7th Conference on Advances in Financial Technologies

AFT 2025, October 8–10, 2025, Pittsburgh, PA, USA

Edited by

Zeta Avarikioti

Nicolas Christin

LIPIcs – Vol. 354 – AFT 2025 www.dagstuhl.de/lipics

Editors

Zeta Avarikioti
TU Wien, Austria
georgia.avarikioti@tuwien.ac.at

Nicolas Christin
Carnegie Mellon University, Pittsburgh, PA, USA
nicolasc@andrew.cmu.edu

ACM Classification 2012
Security and privacy → Mathematical foundations of cryptography; Security and privacy → Distributed systems security; Security and privacy → Privacy-preserving protocols; Security and privacy → Pseudonymity, anonymity and untraceability; Security and privacy → Economics of security and privacy; Theory of computation → Algorithmic game theory and mechanism design; Theory of computation → Cryptographic primitives; Theory of computation → Cryptographic protocols; Applied computing → Digital cash; Applied computing → Economics

ISBN 978-3-95977-400-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-400-0.

Publication date
October, 2025

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists all publications of this volume in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

License
This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): https://creativecommons.org/licenses/by/4.0/legalcode.
In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:
- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.AFT.2025.0

ISBN 978-3-95977-400-0 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

Editorial Board

- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université Paris Cité, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Holger Hermanns (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Daniel Král' (Leipzig University, DE and Max Planck Institute for Mathematics in the Sciences, Leipzig, DE)
- Sławomir Lasota (University of Warsaw, PL)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN – Chair)
- Chih-Hao Luke Ong (Nanyang Technological University, SG)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Pierre Senellart (ENS, Université PSL, Paris, France)
- Alexandra Silva (Cornell University, Ithaca, US)

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface
Zeta Avarikioti and Nicolas Christin 0:ix

Program Committee
..... 0:xi

Steering Committee
..... 0:xiii

External Reviewers
..... 0:xv

List of Authors
..... 0:xvii

Session 1: Cryptography and Privacy in Blockchains

Cache Timing Leakages in Zero-Knowledge Protocols
Shibam Mukherjee, Christian Rechberger, and Markus Schofnegger 1:1–1:26

Zero-Knowledge Authenticator for Blockchain: Policy-Private and Obliviously Updateable
Kostas Kryptos Chalkias, Deepak Maram, Arnab Roy, Joy Wang, and Aayush Yadav 2:1–2:23

Fast, Private and Regulated Payments in Asynchronous Networks
Maxence Brugeres, Victor Languille, Petr Kuznetsov, and Hamza Zarfaoui 3:1–3:24

Proxying Is Enough: Security of Proxying in TLS Oracles and AEAD Context Unforgeability
Zhongtang Luo, Yanxue Jia, Yaobin Shen, and Aniket Kate 4:1–4:24

Blockchain Governance via Sharp Anonymous Multisignatures
Wonseok Choi, Xiangyu Liu, and Vassilis Zikas 5:1–5:24

Session 2: AMM

Modeling Loss-Versus-Rebalancing in Automated Market Makers via Continuous-Installment Options
Srisht Fateh Singh, Reina Ke Xin Li, Samuel Gaskin, Yuntao Wu, Jeffrey Klinck, Panagiotis Michalopoulos, Zissis Poulos, and Andreas Veneris 6:1–6:23

Mechanism Design for Automated Market Makers
T-H. Hubert Chan, Ke Wu, and Elaine Shi 7:1–7:22

Strategic Analysis of Just-In-Time Liquidity Provision in Concentrated Liquidity Market Makers
Bruno Llacer Trotti, Weizhao Tang, Rachid El-Azouzi, Giulia Fanti, and Daniel Sadoc Menasché 8:1–8:22

7th Conference on Advances in Financial Technologies (AFT 2025).
Editors: Zeta Avarikioti and Nicolas Christin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Session 3: Measurements

Money in Motion: Micro-Velocity and Usage of Ethereum's Liquid Staking Tokens
Benjamin Kraner, Luca Pennella, Nicolò Vallarano, and Claudio J. Tessone 9:1–9:18

Do Inflation Expectations Drive Cryptocurrency Investments?
Will Cong, Pulak Ghosh, Jiasun Li, Qihong Ruan, and Artem Streltsov 10:1–10:3

Cryptocurrency Exchange Listings
Jiasun Li, Mei Luo, Muzhi Wang, and Zhe Wei 11:1–11:2

Session 4: Consensus I

How Much Public Randomness Do Modern Consensus Protocols Need?
Joseph Bonneau, Benedikt Bünz, Miranda Christ, and Yuval Efron 12:1–12:19

Composable Byzantine Agreements with Reorder Attacks
Jing Chen, Jin Dong, Jichen Li, Xuanzhi Xia, and Wentao Zhou 13:1–13:23

Optimistic Message Dissemination
Chen-Da Liu-Zhang, Christian Matt, and Søren Eller Thomsen 14:1–14:24

Beyond Optimal Fault-Tolerance
Andrew Lewis-Pye and Tim Roughgarden 15:1–15:23

Session 5: Consensus II

Nakamoto Consensus from Multiple Resources
Mirza Ahad Baig, Christoph U. Günther, and Krzysztof Pietrzak 16:1–16:23

Fully-Fluctuating Participation in Sleepy Consensus
Yuval Efron, Joachim Neu, and Toniann Pitassi 17:1–17:22

From Permissioned to Proof-of-Stake Consensus
Jovan Komatovic, Andrew Lewis-Pye, Joachim Neu, Tim Roughgarden, and Ertem Nusret Tas 18:1–18:26

Two-Tier Black-Box Blockchains and Application to Instant Layer-1 Payments
Michele Ciampi, Yun Lu, Rafail Ostrovsky, and Vassilis Zikas 19:1–19:24

Session 6: Incentives and Tokenomics

Selfish Mining Under General Stochastic Rewards
Maryam Bahrani, Michael Neuder, and S. Matthew Weinberg 20:1–20:23

Pool Formation in Oceanic Games: Shapley Value and Proportional Sharing
Aggelos Kiayias, Elias Koutsoupias, Evangelos Markakis, and Panagiotis Tsamopoulos 21:1–21:24

Single-Token vs Two-Token Blockchain Tokenomics
Aggelos Kiayias, Philip Lazos, and Paolo Penna 22:1–22:22

Session 7: Mechanism Design for Fees and Markets

Transaction Fee Market Design for Parallel Execution
Bahar Acilan, Andrei Constantinescu, Lioba Heimbach, and Roger Wattenhofer .. 23:1–23:25

Multidimensional Blockchain Fees Are (Essentially) Optimal
Guillermo Angeris, Theo Diamandis, and Ciamac Moallemi 24:1–24:23

The Exchange Problem
Mohit Garg and Suneel Sarswat 25:1–25:19

Session 8: Empirical Analysis and MEV

Measuring CEX-DEX Extracted Value and Searcher Profitability: The Darkest of the MEV Dark Forest
Fei Wu, Danning Sui, Thomas Thiery, and Mallesh Pai 26:1–26:23

Unravelling the Probabilistic Forest: Arbitrage in Prediction Markets
Oriol Saguillo, Vahid Ghafouri, Lucianna Kiffer, and Guillermo Suarez-Tangil ... 27:1–27:24

Optimistic MEV in Ethereum Layer 2s: Why Blockspace Is Always in Demand
Ozan Solmaz, Lioba Heimbach, Yann Vonlanthen, and Roger Wattenhofer 28:1–28:24

Session 9: Scalability and Interoperability

Efficient Parallel Execution of Blockchain Transactions Leveraging Conflict Specifications
Parwat Singh Anjana, Matin Amini, Rohit Kapoor, Rahul Parmar, Raghavendra Ramesh, Srivatsan Ravi, and Joshua Tobkin 29:1–29:26

Ticket to Ride: Locally Steered Source Routing for the Lightning Network
Sajjad Alizadeh and Majid Khabbazian 30:1–30:21

Trustless Bridges via Random Sampling Light Clients
Bhargav Nagaraja Bhatt, Fatemeh Shirazi, and Alistair Stewart 31:1–31:24

4-Swap: Achieving Grief-Free and Bribery-Safe Atomic Swaps Using Four Transactions
Kirti Singh, Vinay J. Ribeiro, and Susmita Mandal 32:1–32:22

Session 10: Cryptocurrency Markets and Adoption

Cuttlefish: A Fair, Predictable Execution Environment for Cloud-Hosted Financial Exchanges
Liangcheng Yu, Prateesh Goyal, Ilias Marinos, and Vincent Liu 33:1–33:25

PvpAMM: A Perpetual Market for Unbalanced Long-Short Positions
Zhenhang Shang, Zhenyu Zhao, and Kani Chen 34:1–34:19

On-Chain Decentralized Learning and Cost-Effective Inference for DeFi Attack Mitigation
Abdulrahman Alhaidari, Balaji Palanisamy, and Prashant Krishnamurthy 35:1–35:27

Preface

This volume contains 35 papers selected from a record-high 135 submissions – a 26% acceptance rate – to the Conference on Advances in Financial Technologies (AFT '25) held at Carnegie Mellon University, in Pittsburgh, Pennsylvania, USA, on October 8–10, 2025. This is the seventh year of the conference and the third year that it is independently organized and published in LIPIcs.

This year's edition was hosted in collaboration with the Carnegie Mellon CyLab Security and Privacy Institute, and, more specifically, CyLab's Secure Blockchain Initiative (SBI). CyLab is one of the largest security and privacy academic centers in the United States, and coordinates security and privacy research and education across all university departments at Carnegie Mellon, and SBI is one of its key initiatives. We elected to have this year's conference immediately follow the Secure Blockchain Initiative Summit, to foster collaboration across disciplinary boundaries and professional communities.

The paper selection process followed the typical process used by many computer science conferences. Each submission received at least three, and often four, detailed double-blind reviews by several program committee members and external reviewers. Each accepted paper was presented via a 15-minute live presentation, followed by a 5-minute question/answer period with the audience. We grouped papers in ten topical sessions, covering a broad range of theoretical and practical advances ranging from "Cryptography and Privacy in Blockchains" to "Measurements." Our scientific program was complemented by a keynote speech and an ever-entertaining rump session.

We thank all Program Committee members and external reviewers for their service in selecting the AFT program, and all authors for submitting their work for consideration. We also grateful to the AFT steering committee for their support and guidance throughout the process. We acknowledge the industry sponsors whose financial support was once again essential to running AFT:

Gold level

- a16z crypto

Silver level

- Ava Labs
- IC3
- Input Output
- TRM Labs

We are also deeply thankful for the support by the U.S. National Science Foundation Security, Privacy, and Trust in Cyberspace (SaTC 2.0) program under Grant No. 2420955.

We appreciate the immense efforts of Michael Lisanti, who served as General Co-Chair. Michael and the staff that worked with him – in particular, Ashley Bon, Isabelle Glassmith, and Beth Bucher – tirelessly handled all the logistical questions and were critical in making the event a success.

Vienna, Austria
Pittsburgh, PA, USA
October 2025

Zeta Avarikioti
Nicolas Christin

7th Conference on Advances in Financial Technologies (AFT 2025).
Editors: Zeta Avarikioti and Nicolas Christin
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Aggelos Kiayias, University of Edinburgh and IOG

Akaki Mamageishvili, Offchain Labs

Alberto Sonnino, MystenLabs and University College London

Aljosha Judmayer, University of Vienna

Andrew Lewis-Pye, London School of Economics

Andrew Miller, University of Illinois Urbana-Champaign

Ari Juels, Cornell Tech

Ariel Zetlin-Jones, Carnegie Mellon University

Arthur Gervais, University College London

Aviad Rubinstein, Stanford University

Aviv Yaish, Yale University, IC3

Bernardo David, IT University of Copenhagen

Bernhard Haslhofer, Complexity Science Hub

Bryan Routledge, Carnegie Mellon University

Christian Cachin, University of Bern

Ciamac Moallemi, Columbia University

Clara Shikhelman, Chaincode Labs

Dimitris Karakostas, University of Edinburgh and Common Prefix

Dionysis Zindors, Common Prefix

Fahad Saleh, University of Florida

Fan Zhang, Yale University

Florian Tschorsch, TU Dresden

Geoffrey Ramseyer, Stellar Development Foundation

George Danezis, MystenLabs and University College London

Georgios Piliouras, Google DeepMind and SUTD

Ghassan Karame, Ruhr-University Bochum

Giulia Fanti, Carnegie Mellon University

Guy Goren, Aptos Labs

Hong-Sheng Zhou, Virginia Commonwealth University

Ittay Eyal, Technion

James Chiang, Aarhus University and Common Prefix

Jason Milionis, Columbia University

Jiasun Li, George Mason University

Jing Chen, Tsinghua University

Johnnatan Messias, Max Planck Institute for Software Systems (MPI-SWS)

Julien Prat, IP Paris

Kaihua Qin, Yale University

Kanye Ye Wang, University of Macau

Karl Wüst, Mysten Labs

Krzysztof Pietrzak, ISTA

Kyle Soska, Independent

Lioba Heimbach, ETH Zurich

Lucianna Kiffer, IMDEA Networks

Lukas Aumayr, University of Edinburgh and Common Prefix

Marco Reuter, International Monetary Fund

Marie Vasek, University College London

Matheus Xavier Ferreira, University of Virginia

Philipp Jovanovic, University College London

Pierre-Louis Roman, Independent

Pietro Saggese, IMT School for Advanced Studies Lucca

7th Conference on Advances in Financial Technologies (AFT 2025).
Editors: Zeta Avarikioti and Nicolas Christin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Qiang Tang, The University of Sydney

Rainer Boehme, University of Innsbruck

Robin Fritsch, ETH Zurich

Ruizhe Jia, Stanford University

Sara Tucci-Piergiovanni, CEA LIST
Paris-Saclay University

Sarah Azouvi, Independent

Tarun Chitra, Gauntlet

Tim Roughgarden, Columbia University and
a16z crypto

Will Cong, Cornell SC Johnson

Yonatan Sompolinsky, Harvard University

Steering Committee

Ittai Abraham (co-chair), VMware research

Dan Boneh, Stanford University

Christian Cachin, University of Bern

Ittay Eyal (co-chair), Technion

Maurice Herlihy, Brown University

Maureen O'Hara, Cornell University

Tim Roughgarden, Columbia University

Eli Ben Sasson, Technion

Emin Gun Sirer (co-chair), Cornell University

S. Matthew Weinberg ('23 PC chair), Princeton University

Joseph Bonneau ('23 PC chair), New York University

Rainer Boehme ('24 PC chair), University of Innsbruck

Lucianna Kiffer ('24 PC chair), IMDEA Networks

Nicolas Christin ('25 PC chair), Carnegie Mellon University

Zeta Avarikioti ('25 PC chair), TU Wien and Common Prefix

7th Conference on Advances in Financial Technologies (AFT 2025).
Editors: Zeta Avarikioti and Nicolas Christin

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

7th Conference on Advances in Financial Technologies (AFT 2025).
Editors: Zeta Avarikioti and Nicolas Christin

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Amirreza Sarencheh, University of Edinburgh

Bo Li, Hong Kong Polytechnic University

Burak Öz, TU Munich

Hao Chung, Carnegie Mellon University

Jichen Li, Peking University

Matthias Fitzi, IOHK

Paolo Penna, IOHK

Peter Gaži, IOG

Pyrros Chaidos, IOG

Sandro Coretti-Drayton, IOG

Sotiris Georganas, City University London

Wenrui Zuo, University of Edinburgh

List of Authors

Bahar Acilan (23)
ETH Zürich, Switzerland

Abdulrahman Alhaidari (35)
School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA

Sajjad Alizadeh (30)
Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada

Matin Amini (29)
University of Southern California, Los Angeles, CA, USA

Guillermo Angeris (24)
Bain Capital Crypto, Boston, MA, USA

Parwat Singh Anjana (29)
Supra Research, Udaipur, India

Maryam Bahrani (20)
Ritual, Vancouver, BC, Canada

Mirza Ahad Baig (16)
Institute of Science and Technology Austria, Klosterneuburg, Austria

Bhargav Nagaraja Bhatt (31)
Web3 Foundation, Zug, Switzerland

Joseph Bonneau (12)
New York University, NY, USA

Maxence Brugeres (3)
LTCI, Telecom Paris, Institut Polytechnique de Paris, France; Forvis Mazars, Levallois-Perret, France

Benedikt Bünz (12)
New York University, NY, USA

T-H. Hubert Chan (7)
University of Hong Kong, Hong Kong

Jing Chen (13)
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Kani Chen (34)
Hong Kong University of Science and Technology, Hong Kong SAR, China

Wonseok Choi (5)
DGIST, Daegu, South Korea

Miranda Christ (12)
Columbia University, New York, NY, USA

Michele Ciampi (19)
University of Edinburgh, UK

Will Cong (10)
Cornell University, Ithaca, NY, USA

Andrei Constantinescu (23)
ETH Zürich, Switzerland

Theo Diamandis (24)
Bain Capital Crypto, Boston, MA, USA

Jin Dong (13)
Beijing Academy of Blockchain and Edge Computing (BABEC), China

Yuval Efron (12, 17)
Columbia University, New York, NY, USA

Rachid El-Azouzi (8)
Avignon University, France

Giulia Fanti (8)
Carnegie Mellon University, Pittsburgh, PA, USA

Mohit Garg (25)
Indian Institute of Science, Bengaluru, India

Samuel Gaskin (6)
Independent Researcher, Toronto, Canada

Vahid Ghafouri (27)
Oxford Internet Institute, UK

Pulak Ghosh (10)
Indian Institute of Management, Bangalore, India

Prateesh Goyal (33)
Microsoft Reserach, Redmond, WA, USA

Christoph U. Günther (16)
Institute of Science and Technology Austria, Klosterneuburg, Austria

Lioba Heimbach (23, 28)
ETH Zürich, Switzerland

Yanxue Jia (4)
Purdue University, United States

Rohit Kapoor (29)
Supra Research, New Delhi, India

Aniket Kate (4)
Purdue University, United States; Supra Research, United States

7th Conference on Advances in Financial Technologies (AFT 2025).
Editors: Zeta Avarikioti and Nicolas Christin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Majid Khabbazian (30)
Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada

Aggelos Kiayias (21, 22)
University of Edinburgh, UK;
Input Output Global (IOG), Edinburgh, UK

Lucianna Kiffer (27)
IMDEA Networks, Madrid, Spain

Jeffrey Klinck (6)
University of Toronto, Canada

Jovan Komatovic (18)
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Elias Koutsoupias (21)
University of Oxford, UK

Benjamin Kraner (9)
Blockchain & Distributed Ledger Technologies Group, University of Zurich, Switzerland

Prashant Krishnamurthy (35)
School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA

Kostas Kryptos Chalkias (2)
Mysten Labs, Palo Alto, CA, USA

Petr Kuznetsov (3)
LTCI, Telecom Paris, Institut Polytechnique de Paris, France

Victor Languille (3)
LTCI, Telecom Paris, Institut Polytechnique de Paris, France; EDF R&D, Paris, France

Philip Lazos (22)
London, UK

Andrew Lewis-Pye (15, 18)
London Schoool of Economics (LSE), UK

Jiasun Li (10, 11)
George Mason University, Fairfax, VA, USA

Jichen Li (13)
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Reina Ke Xin Li (6)
University of Toronto, Canada

Vincent Liu (33)
University of Pennsylvania, Philadelphia, PA, USA

Xiangyu Liu (5)
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Chen-Da Liu-Zhang (14)
Lucerne University of Applied Sciences and Arts, Rotkreuz, Switzerland;
Web3 Foundation, Zug, Switzerland

Bruno Llacer Trotti (8)
Federal University of Rio de Janeiro (UFRJ), Brazil

Yun Lu (19)
University of Victoria, Canada

Mei Luo (11)
Tsinghua University, Beijing, China

Zhongtang Luo (4)
Purdue University, United States

Susmita Mandal (32)
Institute for Development and Research in Banking Technology, Hyderabad, India

Deepak Maram (2)
Mysten Labs, Palo Alto, CA, USA

Ilias Marinos (33)
Nvidia, Redmond, WA, USA

Evangelos Markakis (21)
Athens University of Economics and Business, Greece;
Input Output Global (IOG), Athens, Greece

Christian Matt (14)
Primev, Steinhausen, Switzerland

Daniel Sadoc Menasché (8)
Federal University of Rio de Janeiro (UFRJ), Brazil

Panagiotis Michalopoulos (6)
University of Toronto, Canada

Ciamac Moallemi (24)
Graduate School of Business, Columbia University, New York, NY, USA

Shibam Mukherjee (1)
Graz University of Technology, Austria;
Know Center, Graz, Austria

Joachim Neu (17, 18)
a16z Crypto Research, New York, NY, USA

Michael Neuder (20)
Ethereum Foundation, New York, NY, USA

Rafail Ostrovsky (19)
University of California, Los Angeles (UCLA), CA, USA

Mallesh Pai (26)
Rice University, Houston, TX, USA; Tempo Labs, Houston, TX, USA; Paradigm, Houston, TX, USA

Balaji Palanisamy (35)
School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA

Rahul Parmar (29)
Supra Research, Ahmedabad, India

Paolo Penna (22)
IOG, Zurich, Switzerland

Luca Pennella (9)
Department of Economics, Business, Mathematics and Statistics, University of Trieste, Italy

Krzysztof Pietrzak (16)
Institute of Science and Technology Austria, Klosterneuburg, Austria

Toniann Pitassi (17)
Columbia University, New York, NY, USA

Zissis Poulos (6)
York University, Toronto, Canada

Raghavendra Ramesh (29)
Supra Research, Brisbane, Australia

Srivatsan Ravi (29)
Supra Research, Los Angeles, CA, USA; University of Southern California, Los Angeles, CA, USA

Christian Rechberger (1)
Graz University of Technology, Austria; TACEO, Graz, Austria

Vinay J. Ribeiro (32)
Indian Institute of Technology Bombay, India

Tim Roughgarden (15, 18)
Columbia University, New York, NY, USA; a16z Crypto Research, New York, NY, USA

Arnab Roy (2)
Mysten Labs, Palo Alto, CA, USA

Qihong Ruan (10)
Cornell University, Ithaca, NY, USA

Oriol Saguillo (27)
IMDEA Networks, Madrid, Spain

Suneel Sarswat (25)
National Institute of Securities Markets, Mumbai, India

Markus Schofnegger (1)
Fabric Cryptography, San Francisco, CA, USA

Zhenhang Shang (34)
Hong Kong University of Science and Technology, Hong Kong SAR, China

Yaobin Shen (4)
Xiamen University, China

Elaine Shi (7)
Carnegie Mellon University, Pittsburgh, PA, USA

Fatemeh Shirazi (31)
Web3 Foundation, Zug, Switzerland

Kirti Singh (32)
Indian Institute of Technology Bombay, India; Institute for Development and Research in Banking Technology, Hyderabad, India

Srisht Fateh Singh (6)
University of Toronto, Canada

Ozan Solmaz (28)
ETH Zürich, Switzerland

Alistair Stewart (31)
Web3 Foundation, Zug, Switzerland

Artem Streltsov (10)
SUNY Buffalo, Buffalo, NY, USA

Guillermo Suarez-Tangil (27)
IMDEA Networks, Madrid, Spain

Danning Sui (26)
Flashbots, San Francisco, CA, USA

Weizhao Tang (8)
Carnegie Mellon University, Pittsburgh, PA, USA

Ertem Nusret Tas (18)
Stanford University, CA, USA

Claudio J. Tessone (9)
UZH Blockchain Center, University of Zurich, Switzerland

Thomas Thiery (26)
Ethereum Foundation, Lisbon, Portugal

Søren Eller Thomsen (14)
Partisia, Aarhus, Denmark

Joshua Tobkin (29)
Supra Research, Taipei, Taiwan

Panagiotis Tsamopoulos (21)
Athens University of Economics and Business, Greece

Nicolò Vallarano (9)
UZH Blockchain Center, University of Zurich, Switzerland

Andreas Veneris (6)
University of Toronto, Canada

Yann Vonlanthen (28)
ETH Zürich, Switzerland

Joy Wang (2)
Mysten Labs, Palo Alto, CA, USA

Muzhi Wang (11)
Central University of Finance and Economics, Beijing, China

Roger Wattenhofer (23, 28)
ETH Zürich, Switzerland

Zhe Wei (11)
Shantou University, China

S. Matthew Weinberg (20)
Princeton University, NJ, USA

Fei Wu (26)
King's College London, UK

Ke Wu (7)
University of Michigan, Ann Arbor, MI, USA; Carnegie Mellon University, Pittsburgh, PA, USA

Yuntao Wu (6)
University of Toronto, Canada

Xuanzhi Xia (13)
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Aayush Yadav (2)
George Mason University, Fairfax, VA, USA

Liangcheng Yu (33)
Microsoft Research, Redmond, WA, USA; University of Pennsylvania, Philadelphia, PA, USA

Hamza Zarfaoui (3)
LTCI, Telecom Paris, Institut Polytechnique de Paris, France

Zhenyu Zhao (34)
Hong Kong University of Science and Technology, Hong Kong SAR, China

Wentao Zhou (13)
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Vassilis Zikas (5, 19)
Georgia Institute of Technology, Atlanta, GA, USA

Cache Timing Leakages in Zero-Knowledge Protocols

Shibam Mukherjee
Graz University of Technology, Austria
Know Center, Graz, Austria

Christian Rechberger
Graz University of Technology, Austria
TACEO, Graz, Austria

Markus Schofnegger
Fabric Cryptography, San Francisco, CA, USA

Abstract

The area of modern zero-knowledge proof systems has seen a significant rise in popularity over the last couple of years, with new techniques and optimized constructions emerging on a regular basis.

As the field matures, the aspect of implementation attacks becomes more relevant, however side-channel attacks on zero-knowledge proof systems have seen surprisingly little treatment so far. In this paper, we give an overview of potential attack vectors and show that some of the underlying finite field libraries, and implementations of heavily used components like hash functions using them, are vulnerable w.r.t. cache attacks on CPUs.

On the positive side, we demonstrate that the computational overhead to protect against these attacks is relatively small.

2012 ACM Subject Classification Security and privacy → Domain-specific security and privacy architectures

Keywords and phrases zero-knowledge, protocol, cache timing, side-channel, leakage

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.1

Related Version *ePrint Version*: https://eprint.iacr.org/2024/1390

Acknowledgements Special thanks to Andreas Kogler[1] for all the side-channel discussions we had. Also thanks to Erik Kraft and Lukas Giner[2] for the discussions on specific aspects of the attacks. We would also very much like to thank Roman Walch[3] and the members of TACEO and other groups for their useful feedback, especially on the current state of the art ZK scenario.

1 Introduction

Recent years have witnessed a significant development in the area of zero-knowledge proofs (ZKPs) [38] and it application in blockchain [94, 66, 32, 35, 108], decentralized apps [84, 31, 101], anonymous credentials [110], crypto-assets [124, 86, 91], verifiable computation [79, 127], and decentralized storage [109, 112, 104, 72], among others. A ZK protocol allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ the validity of some statement without revealing any additional information about the statement to $\mathcal{V}$. More specifically, most ZKP applications rely on the technique of zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) [19, 3, 39, 121] for sub-linear proof size and verification time. Since these properties are crucial for practicality, many recent developments focus on making these

[1] https://andreaskogler.com/
[2] https://ginerlukas.com/
[3] https://rwalch.at/

© Shibam Mukherjee, Christian Rechberger, and Markus Schofnegger;
licensed under Creative Commons License CC-BY 4.0

proofs even more efficient. This has also led to groups like Fabric [33], Ingonyama [56], and Supranational [111] working on dedicated hardware accelerators for ZK proof generation and verification.

At USENIX'20, Tramer, Boneh and Paterson [114] demonstrated attacks on two privacy-focused cryptocurrencies, namely, Zcash [124] and Monero [86]. They exploited the timing-based side-channel leakages occurring when executing the protocol, allowing an attacker $\mathcal{A}$ to de-anonymize the secret payee (prover $\mathcal{P}$), violating the fundamental purpose of using private transactions. In their attack, the timing leakage stemmed from the high-level protocol properties such as changed communication patterns, or additional operations like commitments being computed only in certain cases. More recently at USENIX'24, [24] took a broader look into the potential security vulnerabilities in SNARKs protocols, in particular, they discuss various threat models and system-level vulnerabilities.

This work looks into the lower level functionalities of the ZK protocols potentially vulnerable to timing attacks [62, 23], in particular *cache-based timing side-channel attacks* [17, 115]. We observed, leakages occurring in the fundamental components of ZK protocols like the underlying field arithmetic libraries used in various constructions like Merkle trees or operations like NTT, polynomial evaluation, among others. More specifically, we looked into the Merkle tree construction when using hash functions with leaky implementation become susceptible to cache timing attacks.

Being the first work in the literature to explore such timing leakages, we only rely on the well-known Flush+Reload (F+R) [122] and Prime+Probe (P+P) [4, 77, 55] attacks and conjecture that other advanced timing attacks such as the ones with pre-fetching [27] or attaining sub-cache level resolution [105, 123] will also work, potentially with even fewer assumptions and higher precision. We show that when constructing Merkle Trees for ZK membership proofs [16] with pairing-based [60, 28, 46] or FRI-based [11] approaches, a non-constant time hash implementation[4] may leak the preimage containing sensitive information about $\mathcal{P}$'s private witness or potentially forfeiting the zero-knowledge property of the ZK protocols.

On scrutinizing the standard field arithmetic libraries like Rust `ff`, `ff-ce`, `ark-ff` crates, Python `galois` library, Typescript `circomlibjs`, we found several potential timing leakages in various field arithmetic operations like modulo addition, multiplication, reduction, exponentiation, among others. The same leaky field arithmetic libraries are used in several public implementation of ZK-friendly hash functions like *Rescue* [11], Monolith [42], and even the de-facto industry-standard Poseidon [43], carrying forward the timing vulnerabilities in the hash functions. It should be emphasized that the above hash functions themselves exhibit a constant-time algorithm, and the leakages occur only due to the use of underlying leaky field arithmetic libraries for the implementation. Additionally, we also looked into the family of lookup-based ZK-friendly hash functions, like Reinforced Concrete (RC) [41] and to some extent Tip5 [113], which are inherently leaky (in the plain evaluation) due to the use of lookup-tables (LUTs) for efficient ZK proofs and verification. In such hash functions the timing leakage may originate both from the hashing algorithm itself (due to timing differences of LUT memory-access) and also the underlying field arithmetic library.

[4] Generally branched implementations are often faster than their constant time counterparts. This may serve as a motivation in the industry to use it for performance benefits.

Main Contributions

We summarize our main contributions in the following.

- For the first time in the literature, we discuss the implications of cache-based timing side-channel attacks on ZK protocols, forfeiting the zero-knowledge property of pairing-based and FRI-based ZK membership proofs.[5]
- We also discuss $\mathcal{P}$'s private witness recovery complexity by $\mathcal{A}$ when using P+P and F+R attacks on the building blocks of ZK protocols, in particular, ZK-friendly hash functions like Poseidon and other ZK-friendly designs like the ones with faster plain hashing performance due to use of lookup tables (Reinforced Concrete hash function is the first such proposal).
- As part of our larger survey, in Table 3, we provide a non-exhaustive list of potentially vulnerable ZK applications found in the wild, discuss mitigations against timing attacks[6] and motivate the need for expedited research in the direction of micro-architecture [76, 27, 85, 105] and power [63, 81, 98, 75, 1, 82, 25] based side channel attacks on ZK applications.
- Finally, we also provide a constant-time implementation effort for some of the standard field libraries (forked) used in various ZK applications discussed in this paper. On the bright side, we found that the constant-time implementations do not necessarily come with significant performance penalties.

2 Background

Here we briefly discuss the fundamentals of cache timing attacks like CPU caches, the concept of shared memory, and the two well-known cache timing attack strategies, namely P+P and F+R. Then we go through the fundamentals of zero-knowledge proofs and give a brief introduction on ZK-friendly hash functions and the role they play in pairing-based and FRI-based membership approaches.

2.1 Caches and Shared Memory

A standard CPU has a limited number of data registers, and thus cannot fit all the data it needs during a process execution. Any data that is required in the future is fetched from a low-latency memory into the CPU register before processing it. In most modern CPUs, caches provide this low-latency memory, however, they have a limited storage capacity and thus any large data must be fetched from the main memory using temporal and spatial data pre-fetching prediction algorithms. When executing a process, if the CPU finds the required data in the cache, we call it a *cache hit*. Otherwise, if the data needs to be fetched from the main memory, we call it a *cache miss*. The latter comes with a larger latency as the data needs to be fetched from the significantly slower main memory into the cache before the CPU can process it. For example, in a standard Intel CPU, the L1 cache has a latency of three to five clock cycles, whereas the L3 cache has a latency of around 30 to 40 clock cycles. In comparison, data fetching from the main memory may take up to 300 clock cycles.

[5] We also found some public discussions, like `https://github.com/facebook/winterfell/issues/9`, taking up the same concerns.

[6] Several micro-architectural measures have been proposed in the past, however, they come with their own trade-off. In this work, we do not focus on these mitigations as it is a topic on its own interest, but only discuss the best constant-time implementation practices.

Caches are usually partitioned into m smaller cache sets, which are further divided into n cache lines of b bytes each. We commonly refer to such caches as *n-way set-associative* caches, where the total cache size is $m \cdot n \cdot b$ bytes.[7] The L1 caches are shared between the sibling cores contained in a physical core, whereas the L3 cache is shared among all the physical cores and their respective sibling cores. Depending on the type of the cache-based timing attack, $\mathcal{A}$ and $\mathcal{P}$ may have to share the same physical core in the threat model.

The concept of shared memory is heavily used by the operating system due to its performance benefits. For instance, several processes relying on the same library can access the shared physical memory where the library is mapped, averting the need for having several copies of the same library in the main memory for each process. Standardized cryptographic implementations like OpenSSL [89], NSS [87], Libgcrypt [74], or even ZKP libraries can be examples of such shared libraries.

2.2 Cache Timing Attacks

Flush+Reload Attack

In 2016, Yarom and Falkner [122] presented the first Flush+Reload (F+R) attack to recover the RSA key. A F+R attack targets the shared instance, like a cryptographic library, mapped on the common cache accessible to both $\mathcal{A}$ and $\mathcal{P}$. $\mathcal{A}$ calls the `clflush` instruction to flush (remove) the cache lines containing the library data, for instance, the S-box lookup table (LUT) of AES or even Reinforced Concrete. $\mathcal{A}$ later reloads the flushed data with `maccess` and measures the reload latency. If the data is not present in the cache, reload requires more time as the data needs to be fetched from the main memory. If a particular section of the flushed data has a smaller reload latency, $\mathcal{A}$ learns that $\mathcal{P}$ accessed that section, as the data was already loaded, potentially leaking the secret input to the cryptographic function, for example, the input to the S-box. Similar to the F+R attack, Gruss, Maurice, Wagner, and Mangard [48] proposed the Flush+Flush (F+F) attack, a window-less low latency high resolution attack where reloading the data is not required as $\mathcal{A}$ can always flush the memory.

Prime+Probe Attack

In 2006, Osvik et al. [90] introduced the Prime+Probe (P+P) attack, followed by later improvements [4, 77]. Compared to F+R, this attack does not require access to shared cryptographic library between $\mathcal{A}$ and $\mathcal{P}$, reducing the security assumptions. In this attack, $\mathcal{A}$ primes (loads) all the cache sets with their eviction set (data) and then waits for $\mathcal{P}$ to load their data on the cache, replacing $\mathcal{A}$'s eviction set. $\mathcal{A}$ then sequentially probes the cache sets and looks for addresses where the eviction set was replaced. If the set was replaced by $\mathcal{P}$, $\mathcal{A}$'s access time to the set will be longer as the data needs to be fetched from the main memory into the cache. P+P attacks are generally more noisy and are limited only to the cache set resolution, giving low information leakage compared to F+R attacks which have a cache line resolution.

[7] For example, our test machine runs on an Intel i7-7600U CPU which has a 4 MB L3 cache with $m = 4096$, $n = 16$ and $b = 64$ bytes.

2.3 Zero-Knowledge Proof

A zero-knowledge proof [38] allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ the validity of a statement, using a public input (public data and the circuit) and potentially private witness data such that the public input reveals no information about the statement. For instance, $\mathcal{P}$ may want to convince $\mathcal{V}$ that they know a preimage x such that $y = H(x)$ for a publicly known value y and a cryptographic hash function H. A similar approach is also used for modern post-quantum signature schemes [26, 9, 30, 8], especially the ones based on MPCitH and VOLEitH ZKP paradigms. Here $\mathcal{P}$ proves the knowledge of their secret key *sk* (used for signing a message) tied to a particular public key *pk* (plaintext-ciphertext pair), where the randomness is generated with a seed and the message to be signed. Two crucial properties of any ZKP protocol are *completeness* and *soundness*. Informally, the former ensures that an honest $\mathcal{V}$ will be convinced by an honest $\mathcal{P}$ and the latter ensures that a dishonest $\mathcal{P}$ cannot convince an honest $\mathcal{V}$ except with some small probability (usually designed to be negligible in the security parameter). Additionally, any ZK protocol must also provide the *zero-knowledge* property.[8] Modern general-purpose ZK protocols can be essentially split into two categories, namely pairing-based [46, 29, 80, 28, 2, 73, 102] and FRI-based [11, 14, 18] approaches. With the former, hash functions are often used to prove membership in zero knowledge, e.g. to prove on-chain coin ownership [124, 86] or show group membership [101, 110]. In contrast, FRI-based approaches often use the hash function primitives internally in recursive proving approaches, which is a crucial building block of various ZK-rollup applications [52, 127, 116, 7].

2.4 ZK-Friendly Hash Functions

Modern ZK protocols often rely on hash functions which exhibit a property referred to as *circuit friendliness*, *ZK-friendliness*, or *arithmetization friendliness*, important for strong performance when proving particular statements in ZK. This essentially means that the algorithmic description of the hash function can be translated to a relatively small system of low-degree constraints, which makes these constructions efficient in proof systems for computational integrity. In the following, we give a brief overview of two such constructions, one of them being among the most popular choices and the other being a more modern approach using lookup arguments in the proof.

Poseidon

Poseidon [43] is a cryptographic hash function designed for efficient implementation as an arithmetic circuit, making it particularly suitable for zero-knowledge proof systems. It operates over a prime field p, where $p \approx 2^n$ and $n \geq 31$, mapping the input messages of arbitrary length to fixed-size digests. The underlying sponge permutation, Poseidon$^\pi$: $\mathbb{F}_p^* \to \mathbb{F}_p^o$, where o is the fixed output length measured in p elements, is based on a substitution-permutation network (SPN) first used for HadesMiMC [44]. It consists of several rounds with two different types of layers, namely *full* (or *external*) and *partial* (or *internal*) rounds. The former includes full nonlinear layers where S-boxes are applied to every word in the state. The latter consists of a partial nonlinear layer where the S-box is only applied to the first word of the state. The idea behind this approach is to provide simple arguments against statistical attacks using consecutive full rounds, while achieving the same degree

[8] We note that in some practical settings the ZK property may not be needed, but instead only the computational integrity property of modern general-purpose proof systems is used.

growth using the more efficient partial rounds. The affine layer $M(\cdot)$ consists of a $t \times t$ MDS matrix multiplication and a round constant addition ARC$(\cdot)$. The nonlinear S-box function is defined as $S(x) = x^\alpha$, where $\alpha \geq 3$ is a small positive integer for which $\gcd(p-1, \alpha) = 1$. An overview of the Poseidon$^\pi$ permutation is shown in Figure 1.

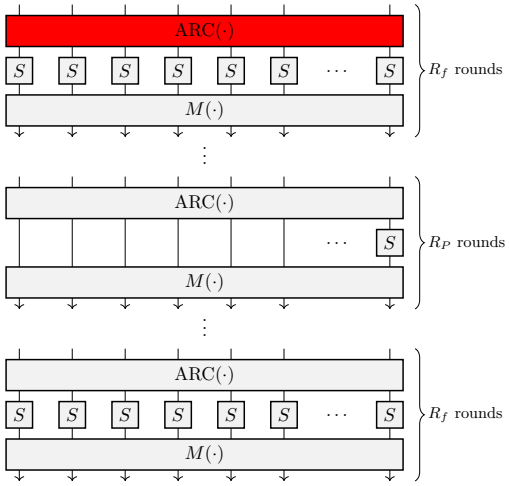


Figure 1 The Poseidon$^\pi$ permutation.

In computational integrity proof systems, the advantages of Poseidon stem from its low number of nonlinear operations, primarily due to the minimal use of S-boxes. This property makes it easier to express the hash function as a circuit and prove knowledge of its preimages within the proving framework. Indeed, nowadays Poseidon is used in various ZK-related applications, including RISC Zero [99], Plonky2 [94], Plonky3 [94], and for many on-chain use cases such as Filecoin [72], Dusk Network [31], Sovrin [107], Loopring [78].[9] We refer the reader to Poseidon [43] for more details.

Reinforced Concrete (RC)

Reinforced Concrete [41] is another ZK-friendly sponge-based hash function which maps $\mathbb{F}_p^3 \to \mathbb{F}_p^3$ for some prime p, operating over two elliptic curves, namely BN254 and BLS12-381 and one special prime field (-ST) crafted for performance. Reinforced Concrete uses lookup-based S-boxes to take advantage of proof systems which use the lookup argument for efficient set membership proofs. The underlying permutation consists of a modified 7-round SPN with Concrete, Bricks and Bars layers, where Concrete $\circ$ Bricks is one round, and a single Bars layer is applied in the middle of the permutation. The Bricks function is a nonlinear permutation. The Concrete function is an MDS matrix multiplication with the state in $\mathbb{F}_p$. The Bar layer consists of composition/decomposition and S-box functions, where the Bar layer is implemented as a lookup table of the functions it contains. RC can be considered as an attractive drop-in replacement for Poseidon due to its increased efficiency for lookup-based protocols like Arya [20], Plookup [36, 92], Halo2 [124], and logUp [49]. A graphical representation of the RC function is provided in Figure 2. We refer the reader to RC [41] for more details.

[9] By "on-chain use cases" we refer to operations and data executed and stored on the main blockchain directly.

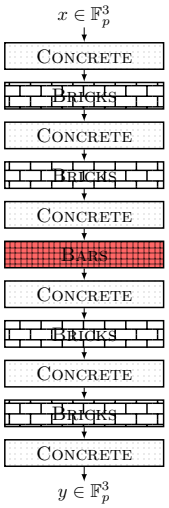


Figure 2 The Reinforced Concrete permutation.

3 Hash Functions in Zero-Knowledge Proofs

We distinguish between two types of general-purpose ZK proof constructions, namely pairing-based and FRI-based ones. Circuit-friendly hash functions are regularly used in both of them, and we give a summary of two main settings below.

For the purpose of this section, we utilize the side-channel technique as a blackbox component in order to describe the setting and the general approach of the attack. In the next section, we go into the details of the side-channel part of the attack specifically.

3.1 Hash Functions in Pairing-Based Proofs

ZK-friendly hash functions defined over pairing-friendly elliptic curves are a powerful tool in many real-world scenarios. In particular, many proof systems [28, 2, 46] (especially those directly employed on chains like Ethereum) are based on this type of elliptic curves in order to support efficient verification of proofs. In this context, a ZK-friendly hash function can be translated to a circuit with minimal overhead.

To give an example, let us consider a simple membership proof where $\mathcal{P}$ wants to convince $\mathcal{V}$ that they hold some secret input (say, a coin). The input is part of a publicly verifiable set fixed by a Merkle root (say, a block on the blockchain containing information about the ownership of the coin). In this case, a hash function can be used to prove knowledge of the secret input without trivially revealing it to $\mathcal{V}$. In order to prove the opening path in the Merkle tree efficiently, a ZK-friendly hash function is used to construct the Merkle tree, resulting in easily verifiable constraints when used together with a pairing-based proof. Here, $\mathcal{V}$ verifies if the path from the leaf to the root of the Merkle tree was computed correctly by checking the correct execution of all the d hash function calls on the path in ZK, where d is the depth of the Merkle tree. In the context of cache timing attacks, any information leakage from the hash function also directly leaks $\mathcal{P}$'s secret witness (leaf) values. Using classical hash functions like SHA-3 makes this approach significantly more expensive in general, as these often result in a more complex and less efficient constraint system when computing the membership proof (i.e., the path from a leaf to the public root).

3.2 Hash Functions in FRI

Similar to pairing-based approaches, ZK-friendly hash functions are also used in FRI-based approaches [10] for constructing Merkle trees during the commitment phase. However, unlike the pairing-based approach, FRI protocols do not directly take the secret witness as the input to the hash function, but instead, at a very high level, a polynomial f is constructed by interpolating the private witnesses, which is then evaluated at some special points (discussed later), and the outputs become the leaves of the Merkle tree fed to the hash functions for computing the Merkle root. Here $\mathcal{P}$ proves to $\mathcal{V}$ that they know a set of points (witnesses) which pass through f of a particular degree d (among other parts in the entire proof). In most applications, these witness points are execution traces of a program running on $\mathcal{P}$'s machine, where $\mathcal{P}$ proves the correct execution of the program to $\mathcal{V}$. The circuit friendliness of a given hash function is particularly important for efficient *recursive proofs*, which essentially include proving verifications of many other proofs.

Currently, FRI-based approaches are used in many popular projects widely adopted in the industry [100, 99, 108, 120, 93], mainly due to their advantages like simple setups and high prover performance. Indeed, many recent developments in the area of zero-knowledge protocols have focused on making FRI-based proving more efficient [51, 6].

Adding Zero Knowledge to FRI

While FRI is often used without the zero knowledge property (e.g., exploiting the succinctness property for arguments of computational integrity), in many cases ZK is needed and activated. For that, various other strategies have to be applied. When focusing on the trace in particular, one part consists of adding random values to an existing witness column in the trace, in order to change the length of the trace to a more suitable number (for example, a power of 2) and to mask the potentially private witness values. We refer to [50] for more details on this approach and a summary of different techniques.

Another step consists of masking the Merkle tree leaves with random paddings, in order to prevent leaking information in the opening part of the proving protocol. A more detailed description of this approach is given in [15]. Since our goal is to find private witnesses, we consider a scenario where zero knowledge is enabled.

Side-Channel Attack on FRI

Let us consider a trace domain $\mathcal{D}$, which we assume to be a multiplicative subgroup of order N generated by ω. The first step of $\mathcal{P}$ is to take the witnesses $\{w_i\}_{i=0}^{N-1}$ in a trace column and to find a polynomial f such that $f(\omega^i) = w_i$. This can be done by an inverse number-theoretic transform (NTT). Note that for simplicity, in this description we omit the random non-witness values added for zero knowledge. We will show that later this makes no difference in the context of our side-channel attack.

After this step, we define the evaluation domain to be a slightly larger domain $\mathcal{D}'$ of $\beta \cdot N$ elements, where β is called the *blowup factor*, generated by $\omega^{1/\beta}$ and shifted by s, i.e., $\mathcal{D}' = \{s \cdot \omega^{i/\beta}\}_{i=0}^{\beta \cdot N-1}$ and $|\mathcal{D}'| = \beta \cdot |\mathcal{D}|$. $\mathcal{P}$ then commits to the evaluations of f over $\mathcal{D}'$ (along with random padding, again added for zero knowledge) using a Merkle tree, where the Merkle root is the commitment. We refer to [10, 12] for more details.

In this scenario, the task of $\mathcal{A}$ is to find the potentially private witness values. This can be done if the evaluations $\{f(s \cdot \omega^{i/\beta})\}_{i=0}^{\beta \cdot N-1}$ are leaked. Indeed, note that the polynomial f originates from the witnesses and, with overwhelming probability, is of degree $N - 1$.

Hence, it suffices to choose any $N-1$ points from $\{(s\cdot\omega^{i/\beta}, f(s\cdot\omega^{i/\beta}))\}_{i=0}^{\beta\cdot N-1}$ and to find a polynomial $f^\star$ which goes through these points. With high probability, $f=f^\star$, and thus evaluating $f^\star$ on $\mathcal{D}=\{\omega^i\}_{i=0}^{N-1}$ is sufficient to find the witness values $\{w_i\}_{i=0}^{N-1}$.

Summarizing, the attack consists of the following steps.

1. Use side-channel leakage to find the evaluations of the trace polynomial f over $\mathcal{D}'=\{s\cdot\omega^{i/\beta}\}_{i=0}^{\beta\cdot N-1}$.
2. Use $|\mathcal{D}|=N$ of these evaluations to find a highly likely candidate $f^\star$ for f (this can be done by interpolation).
3. Evaluate $f^\star$ on $\mathcal{D}$ to find $\{f^\star(\omega^i)=w_i^\star\}_{i=0}^{N-1}$. Again, with high probability, $\{w_i^\star\}_{i=0}^{N-1}=\{w_i\}_{i=0}^{N-1}$, which corresponds to the original witness values.

The last two steps can be handled relatively easily from $\mathcal{A}$'s point of view. However, the first step requires more assumptions, in particular, using high-resolution cache timing attacks, like sub cache line attacks [105, 123], mounted on hash implementations with byte-level leakages, like when using LUTs for the S-boxes in RC hash function. This allows $\mathcal{A}$ to directly learn $\{f(s\cdot\omega^{i/\beta})\}_{i=0}^{\beta\cdot N-1}$ and reconstruct f with trivial complexity as discussed above.

When using low-resolution attacks like F+R and P+P, or if the hash implementation leaks only a few bits of information, $\mathcal{A}$ obtains several candidates for every $\{f(s\cdot\omega^{i/\beta})\}_{i=0}^{\beta\cdot N-1}$ preimage. $\mathcal{A}$ can then iterate through all these candidates in order to reconstruct f. In particular, this is possible if the evaluations of f over $\mathcal{D}'$ exhibit a strong structure, which however is mainly mitigated due to the diffusion properties of the NTT application. Indeed, the evaluation of f over $\mathcal{D}'$ results in a highly unstructured table even for small variations in $\{w_i\}_{i=0}^{N-1}$. An obvious exception occurs if all values in $\{w_i\}_{i=0}^{N-1}$ are the same (i.e., $f(x)=w_0$), which however is an unrealistic scenario both due to the practical use cases for proof systems and due to additional steps, such as padding applied to trace columns.

Zero Knowledge and Side-Channel Resistance

In the context of our cache timing attack, some of the additional techniques to achieve zero knowledge are no obstacle, as recovering the preimage includes both the evaluation points (evaluations of f at publicly known inputs) and the random paddings, which allows $\mathcal{A}$ to fully reconstruct f via interpolation. Once f is reconstructed, $\mathcal{A}$ can trivially retrieve the secret witness by evaluating f at the publicly known input points. The added random values to mask the interpolating polynomial are no obstacle here.

We also emphasize that the initial step of the interpolation is applied to potentially private witness data. It remains a future direction to investigate how this can be exploited in cache attacks with the high-resolution leakages and other side-channel approaches.

4 Cache Timing Attacks

Here we define the threat model under which we mount the F+R and P+P attacks and discuss the preimage recovery complexity when attacking public implementations [45] of Poseidon and RC. We also provide a non-exhaustive list of open source ZK libraries that were found to be using pairing-based and FRI-based ZK approaches with non-constant time Poseidon implementation. It should be emphasized that we only point out the preimage leakage originating from the hash implementation, especially critical for the secure pairing-based approaches. For FRI-based approaches, we are more interested in breaking the zero-knowledge property by recovering some bits of the preimage in the Merkle leaves.

Threat Models

- **First case.** $\mathcal{P}$ is generating the proof for their witness on their personal machine **M**. We assume $\mathcal{A}$ can run side channel measurements on **M** through malware payload delivered to $\mathcal{P}$ or more sophisticated browser-based side-channel exploits [37, 103].
- **Second case.** Assume $\mathcal{P}$ is generating the proof for their witness on a shared **M** (like a cloud), allowing $\mathcal{P}$ to access powerful hardware for generating faster proofs while minimizing the cost by sharing resources[10]. $\mathcal{P}$ trusts cloud **M** to keep its plain unencrypted witness a secret from a third party $\mathcal{A}$. $\mathcal{A}$ (e.g., pretending to be another legit prover), runs their malicious side-channel programs on the shared **M** and learns some (if not all) bits about $\mathcal{P}$'s secret witness through cache-based timing attacks when $\mathcal{P}$ generates the ZK proof for $\mathcal{V}$.
- **Third case.** $\mathcal{P}$ does not trust cloud **M** and sends only encrypted witnesses to **M** which are privately decrypted in the secure enclaves, like Intel SGX, and then the proof is generated. Leaking the witness requires cloud **M** exploiting secure enclave vulnerabilities or performing cache timing attack on the application processor after the private decryption.

Due to the zero-knowledge property, we know that $\mathcal{V}$ cannot extract any new information about the secret witness by only reading the proof sent by $\mathcal{P}$. Likewise, if $\mathcal{A}$ gets access only to the same proof sent to $\mathcal{V}$, $\mathcal{A}$ cannot learn anything new about the witness either, and thus any side-channel attack performed on $\mathcal{V}$'s end is meaningless.

Both $\mathcal{A}$ and $\mathcal{P}$ are required to run on the same CPU, however, they may run on different physical cores as F+R and P+P attacks target the LLC shared by all cores. For the F+R attack we assume that $\mathcal{A}$ and $\mathcal{P}$ share the cryptographic library containing the ZK functionalities like hash functions and arithmetic field operations as a shared object. Also, the `clflush` instruction must be available to $\mathcal{A}$, essential for F+R attacks, otherwise $\mathcal{A}$ may have to perform other similar attacks like Evict+Time [90] or Evict+Reload [47]. All these attacks rely only on measuring the memory access timing difference. For some implementations, as a common practice in the community, when demonstrating proof-of-concept cache-based timing attacks, we use additional `nop` instructions to align the branches on different cache lines. Naturally, such an instruction is equivalent to calling functions from different branches which usually get aligned on different cache lines. Another example is compiler optimization which may put the secret-dependent branched code unfavorably on distinct cache lines, or even cache sets, leading to practical timing attack like in the case of the Kyber constant-time reference implementation.[11] When employing more modern side-channel attacks, one may not require including such `nop` instructions. Before independently verifying the leakages, we used the DATA framework [119][12] to search for any secret-dependent timing leakages in hash function implementations. A summary of the threat model can be found in Table 1.

4.1 Reinforced Concrete Cache Timing Attack

The RC hash function uses lookup tables (LUTs) in the Bars function. Any LUT implementation is vulnerable to cache timing attacks, especially when the LUT lies on multiple cache lines/sets. For the F+R attack, we assume that $\mathcal{A}$ and $\mathcal{P}$ (victim) share the RC

[10] The use-case in consideration mostly determines the computation power required to generate the proof irrespective of the succinct verification protocol. Use-cases like proving root of Merkle tree with $\geq 2^20$ hashes can be resource demanding.

[11] `https://pqshield.com/pqshield-plugs-timing-leaks-in-kyber-ml-kem-to-improve-pqc-implementation-maturity/`

[12] `https://github.com/Fraunhofer-AISEC/DATA/tree/master`

Table 1 A summary of the threat model involving all the 3 parties, $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{V}$ in a ZKP protocol.

Parties	Machine	Cache SCA		Assumptions
		F+R	P+P	
Prover	Personal/Cloud	Yes	No	Shared ZKP library
		N/A	N/A	Additional instructions
		N/A	N/A	Attack resolution
Attacker	Prover's Personal/Cloud	Yes	No	Shared ZKP library
		`clflush,nop`	`nop`	Additional instructions
		Cache line level (64 bytes)	Cache set level (1024 bytes)	Attack resolution
Verifier	N/A	N/A	N/A	N/A

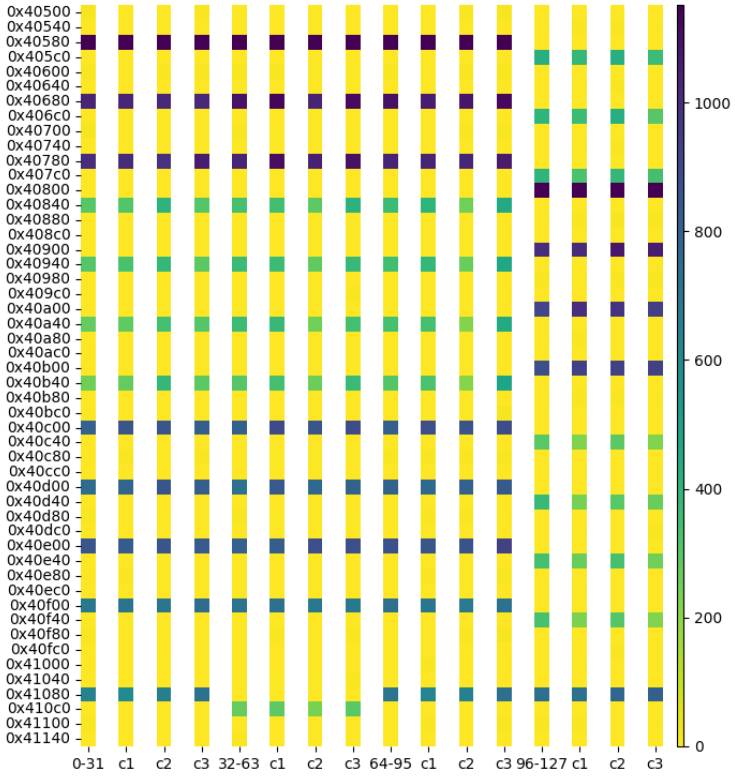
hash function as a shared library including the Bars layer LUT. In RC-BLS12-381, the $\mathbb{F}_p$ elements in the Bars layer are *decomposed* into 27 elements in $\mathbb{F}_{659}$ and then the S-box lookup is applied. Even though $\mathbb{F}_{659}$ can be represented with 10 bits, for efficiency purposes, the elements are mapped as 2-byte elements in the memory, making the LUT 1318 bytes in size, requiring 21 cache lines or 2 cache sets on the LLC of our test machine.

In the F+R attack, with every cache hit $\mathcal{A}$ learns $\approx$ 5 bits of equivalent (total input in bits − possible candidates in bits) S-box input information. In other words, out of 659 elements ($\approx 2^{9.32}$), $\mathcal{A}$ learns the possible $32 = 2^5$ or $19 \approx 2^{4.2}$ (the last cache line) input candidates for the S-box. In P+P attack, however, the recovery resolution is worse than F+R as the resolution is constrained to the cache set level only. In particular, $\mathcal{A}$ learns ≈ 2 bits of equivalent S-box input information, comparable to learning the possible $512 = 2^9$ or $147 \approx 2^7$ (last cache set) input candidates for the S-box. We observe somewhat similar results for RC-BN-254 and RC-ST. However, in the BN-254 implementation there exists a cache line with a single field element, allowing $\mathcal{A}$ to learn all 9.32 bits of equivalent input information in the best case F+R attack scenario occurring with a non-negligible probability of $(642)^{-1} \approx 2^{-9.3}$. We refer to Table 2 for an overview of the preimage recovery complexity. Refer to Figure 3 for illustration of the F+F[13] and P+P attack on the RC hash function.

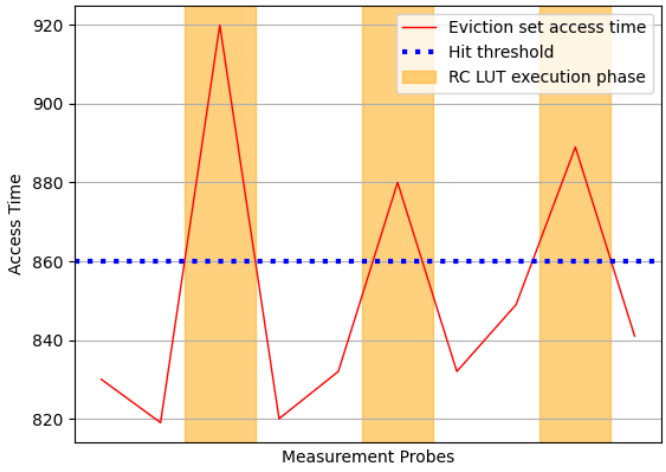
When considering stronger attacks, like the ones with sub-cache line resolution [105], $\mathcal{A}$ may learn all the S-box input bits, fully recovering the preimage without additional search operations. Moreover, for the F+R and P+P attacks, the preimage recovery complexity also decreases when the preimage has some structure to it. This is particularly the case in many ZK applications where the Merkle leaves contain, for example, personally identifiable information like name or dates of birth. Padding some randomness (salt) to the leaves may help against side-channel attacks where $\mathcal{A}$ merely observes the hashes of the tree. However, because the padding is only applied to specific positions of the leaves, this mitigation does not protect against cache timing attacks.

In the following subsections, we discuss some potentially vulnerable Merkle tree constructions used in real-world ZK applications.

[13] We use F+F instead of F+R for attacking RC as the former gives less noisy measurements. The recovery complexity remains the same in both the cases.



(a) F+F cache-hit and cache-miss ratio for Reinforced Concrete BLS-12-381 hash function. Darker blocks represent higher cache-hits in party containing S-box LUT. The input of each S-box starts from `0x0000` and sequentially increases by a multiple of `0x0020` probing the first 4 cache lines, containing 32 inputs each, accessed by $\mathcal{P}$ when generating the Merkle tree. We repeat the attack (c1..c3) demonstrating consistency in the timing leakage when calling S-box inputs contained in the same cache line.



(b) Probe measurements of Reinforced Concrete BLS-12-381 hash function with P+P attack. In the orange phases, the LUT S-box inputs 0-511 are called. We probe the eviction set contained in the cache-set where the 0-511 S-box inputs lie, giving slower probe time compared to not calling the 0-511 S-box inputs.

Figure 3 Access time for F+F and P+P attacks on the Reinforced Concrete hash function. Here $\mathcal{A}$ probes the S-box LUT memory addresses and measures the time to distinguish if the same address was accessed by $\mathcal{P}$ or not.

4.2 Poseidon Cache Timing Attack

As a demonstration, we only attack the Poseidon Goldilocks and Mersenne non-constant time implementations [45] and summarize their leakages.[14] In particular, we attack the ARC(·) function performing modulo reduction after the field addition between the *state* and the round constants *rc*. Depending on the branch, $\mathcal{A}$ can determine if the sum between a state and a round constant is larger than the modulo prime p, where extreme values of round constants (very small or large values in the field) give the most information about the secret state. Additionally, we can also attack functions like $M(\cdot)$ and $S(\cdot)$ containing non-constant time modulo reductions after multiplication operations. However, here we focus on the ARC(·) function, as this is the first layer that directly operates on the preimage. We assume that the `if-else` branches lie on two different cache lines or cache sets for the F+R or P+P attacks respectively.

In the F+R attack, similar to RC, we assume that $\mathcal{A}$ and $\mathcal{P}$ (victim) are sharing the Poseidon library as a shared object, however, $\mathcal{A}$ instead of flushing the LUT addresses, they flush the addresses of the `if-else` branches to detect which branch is accessed by $\mathcal{P}$.

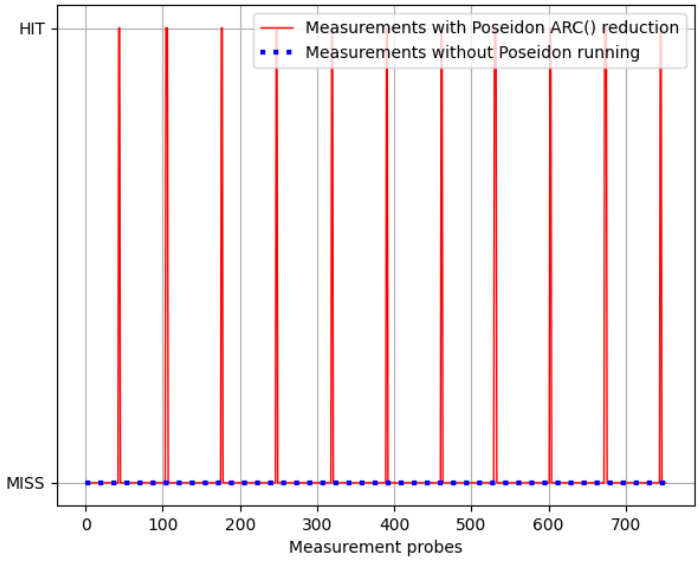
As an input to the Poseidon$^\pi$ permutation, Poseidon-(M, $t = 16$) takes an element in $\mathbb{F}_p^t$, where $p = 2^{31} - 1$ and $t = 16$, where the first layer of Poseidon$^\pi$ is the round constant addition. With the smallest round constant (`0x002f87c1`) in Poseidon-(M, $t = 16$), during a cache hit on the reduction branch, $\mathcal{A}$ learns the most significant ≈ 9 bits (all ones) of $\mathbb{F}_p$ input, where p is `0x7FFFFFFF`.[15] With the largest round constant (`0x7fc1a254`), $\mathcal{A}$ also learns the most significant ≈ 9 bits (all zeros) of the $\mathbb{F}_p$ input. If the round constant *rc* does not "deviate from random" in $\mathbb{F}_p$, then the leakage is only one bit equivalent information. We observed similar leakages when inspecting Poseidon-(M, $t = 24$) and refer to Table 2 for more details and an overview of timing leakages using different versions of the Poseidon hash function.

The generated round constants will be large with an overwhelming probability. However, for large fields like in BLS-12-381 or BN254, usually custom small round constants are preferred as they provide better plain performance (fewer instructions are necessary for the round constant additions). Refer to Figure 4 for the F+R and P+P attack on the Poseidon-GL hash function.

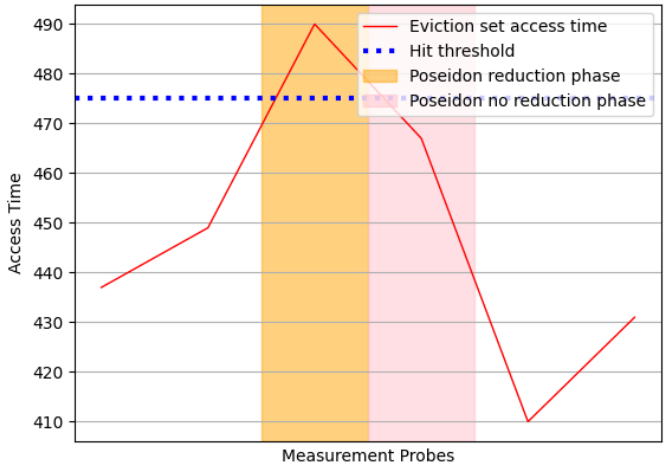
4.3 Other Hash Functions Cache Timing Attack

As the timing leakages mostly stem from the underlying arithmetic library, we also looked into other ZK-fiendly hash function candidates like Tip5, *Rescue*, and Monolith. Due to use of lookups operating over $\mathbb{F}_{2^8+1}$ in one of the Tip5 hash function S-boxes, it may also suffer from the same problems as RC. When performing the F+R attack, $\mathcal{A}$ learns 3 bits of equivalent information in the average case, and all 8 bits in the best case, occurring with a probability of $(2^8 + 1)^{-1}$. We also found some timing leakages in the public implementations of Monolith and *Rescue*. Depending on the *Rescue* implementation, for example, leakage in [67] may occur due to the use of crates like `ff-ce`, similar to Poseidon, or the leakage is only limited to the testing section of the framework like in [120] (`inv()` and `normalize()`). The non-constant time version of the Monolith implementation [45] relies on the same Goldilocks/Mersenne field implementations as Poseidon, thus it might be affected as well. However, Monolith also has a constant time version which is not affected.

[14] The 64-bit Goldilocks prime $2^{64} - 2^{32} + 1$ and the 31-bit Mersenne prime $2^{31} - 1$ define two popular prime fields which are often used for efficient zero-knowledge protocol implementations based on FRI.
[15] Here $\mathcal{A}$ has the same leakage resolution for both F+R and P+P attacks.



(a) Cache-hit and cache-miss ratio for Poseidon-GL hash function with F+R attack. We set the input to the hash function such that modulo reduction (cache hit) is called after every 100 hash calls. We probe the cache address containing the modulo reduction operation branched from the `if-else` check.



(b) Probe measurements of Poseidon-GL hash function with P+P attack. In the orange phase, the round constant addition is followed by the modulo reduction operation. In the pink phase no such reduction is performed. We probe the eviction set contained in the cache-set where the reduction operation instruction also lies, giving slower probe time compared to the phase not executing the modulo reduction.

Figure 4 Access time graph for F+R and P+P attacks on Poseidon-GL hash function. Here $\mathcal{A}$ probes the `if-else` branch memory addresses and measures the time to distinguish if the same addresses were accessed by $\mathcal{P}$.

Table 2 Preimage recovery complexity after F+R and P+P attacks on the Reinforced Concrete and Poseidon hash functions. Here we provide both the average and *best* case input recovery complexity, where the *best* case occurs with a fairly large probability. The preimage leakage is given in terms equivalents bits of information learnt from the full state of the hash function.

Hash function	Timing vulnerability	Preimage leaked (in bits)			
		F+R	% leaked	P+P	% leaked
Poseidon-(M, $t = 16$)	Reduction in ARC(·)	1×16	3.22	1×16	3.22
		$\approx 9 \times 16$	*29.03*	*$\approx 9 \times 16$*	*29.03*
Poseidon-(M, $t = 24$)	Reduction in ARC(·)	1×24	3.22	1×24	3.22
		$\approx 9 \times 24$	*29.03*	*$\approx 9 \times 24$*	*29.03*
Poseidon-(GL, $t = 8$)	Reduction in ARC(·)	1×8	1.56	1×8	1.56
		$\approx 9 \times 8$	*14.06*	*$\approx 9 \times 8$*	*14.06*
Poseidon-(GL, $t = 12$)	Reduction in ARC(·)	1×12	1.56	1×12	1.56
		$\approx 9 \times 12$	*14.06*	*$\approx 9 \times 12$*	*14.06*
RC-(BLS12-381, $n = 27$)	Bars S-box LUT	4.36×27	45.98	0.36×27	3.79
		5.12×27	*54*	*2.17×27*	*22.88*
RC-(BN-254, $n = 27$)	Bars S-box LUT	4.32×27	45.92	0.32×27	3.4
		9.32×27	*100*	*2.31×27*	*24.55*
RC-(ST, $n = 25$)	Bars S-box LUT	4.98×25	49.8	0.98×25	9.8
		5.59×25	*55.9*	*1.02×25*	*10.2*

4.4 Merkle Tree Leaf Distinguisher

Here we discuss the potential attacks that can be mounted on real-world applications, in particular, targeting the Merkle tree construction using non-constant time hash implementations like for Poseidon. One specific aspect we particularly focus on is the entropy of the private values in the Merkle leaves. Low-entropy values, even when padded with randomness at certain points (a common security practice [13, 104]) leak (partial) information about the preimage and hence the secret witness. For example, the Zerocash protocol [13], a foundational work of the Zcash cryptocurrency [124], hashes ($\mathcal{H}(\cdot)$) the coin values $v \in [0, 2^{64} - 1]$ with randomness $k \in [0, 2^{256} - 1]$ as $cm := \mathcal{H}(k \mid\mid 0^{192} \mid\mid v)$, where cm is the commitment value of the coin in the Merkle tree. The authors discuss using SHA-256 as the hash function, however in practice (Zcash), when proving Merkle tree commitments in ZK, Poseidon is the more efficient choice and any non-constant time implementation (Poseidon or other hash function like SHA-256) when constructing the Merkle tree, in this example, will leak the coin value v in accordance with Table 2. Similarly, Sia [104], a decentralized storage protocol, also pads its low-entropy preimage data used in the Merkle tree, such as *time lock*, the number of public keys, and signatures required, by adding a random nonce to the leaves along with the secret input data. Another example is Semaphore [101], a ZK signaling framework, where the public id id_{pub} is padded with a random sequence of bytes $id_{\text{nullifier}}$ when committing to the user identity for generating the Merkle tree membership proofs. These random paddings were adopted to prevent multiple signal broadcasts using the same public id id_{pub}.

In certain conditions, it is even sufficient if $\mathcal{A}$ is able to roughly distinguish the leaves instead of fully recovering the input. For example, a simple ZK use case, where $\mathcal{P}$ convinces $\mathcal{V}$ that they have telephone numbers from certain countries. Let's assume the above Poseidon-(M, $t \in \{16, 24\}$) case where a Merkle tree contains four leaves, the first two leaves contain short phone numbers from Falkland Islands (5 digits, at most 17 bits) and the next two leaves contain phone numbers from Norway (8 digits, at most 27 bits).With the smallest

and largest Poseidon-(M, $t \in \{16, 24\}$) round constants, $\mathcal{A}$ can distinguish between the two groups of the leaves when $\mathcal{P}$ constructs the Merkle tree during the proof generation. Even though leaking only 1 bit of information, this holds a larger privacy implication for ZK protocols.

Table 3 We summarize the ZK frameworks (a non-exhaustive list) that may contain potential witness leakages due to F+R and P+P timing attacks on the various non-constant time Poseidon hash function implementations. We also state the source of the potential leakage affecting both the $ARC(\cdot)$ and $M(\cdot)$ layers.

ZK library (Application) - Source of leakage	Non-constant time implementation	Comments
Matter Labs[66] (Blockchain ZK) - rescue-poseidon [67] → ff-ce crate	Uses `ff-ce` crate with branched `reduce` in `add_assign`	`Readme.md` (`ff-ce`) - Does not provide constant-time guarantees
Lurk Labs [79] (Turing-complete ZK programming language) - neptune-poseidon [64] → ff crate [34]	Uses `ff` crate with branched `reduce` in `add_assign`	`Readme.md` (`neptune-poseidon`) - Has been audited `Readme.md` (`ff`) - Does not provide constant time guarantees
Ingonyama poseidon-hash [56] (Hardware acceleration for ZKP) - poseidon-hash [57] → galois [83]	Uses `Galois` library with branched `add_modular`	`Readme.md` (`galois`) - The library could be vulnerable to timing attacks. Not intended for production.
Sui [110] (ZK login and authentication) - Sui [68] → poseidon-lite [118] → jdk (BigInt) [88]	Uses `poseidon-lite` library with branched addition using `jdk/../BigInteger.java`	`Readme.md` (`poseidon-lite`) - The code has not been audited and the native js `BigInt` is vulnerable to timing attacks.
ZK-kit [125] (General ZK library for developing various applications) - ZK-kit [126]	Field arithmetic `f1-field.ts` contains branched addition.	–
Panther [91] (Virtual asset private trading) - Panther [97] → circomlibjs [53] → ffjavascript [54] → jdk (BigInt) [88]	Uses `circomlibjs` Poseidon implementation using `ffjavascript` for BN-128 and BLS12-381 with branched addition performed with `jdk/../BigInteger.java`.	–
Light Protocol [65] (ZK layer on Solana [106]) - Light Protocol [96] → light-poseidon [95] → ark-ff crate [5]	Uses `light-poseidon` using `ark-ff` crate for arithmetic operations. `add_assign` in `montgomery_backend.rs` has branching	`Readme.md` (`light-poseidon`) - Has been audited `Readme.md` (`ark-ff`) - Academic proof of concept, not ready for production use
Mina Protocol [84] (Decentralized Apps) - O1-labs/proof-systems [69] → ark-ff crate [5]	Uses ark-ff crate, `add_assign` in `montgomery_backend.rs` has branching	`Readme.md` (`proof-systems`) - Security audit missing. `Readme.md` (`ark-ff`) - Academic proof of concept, not ready for production use.
Plonky2/3 [70, 71] (Blockchain ZK) - Plonky2/3	`add` contains branching occurring with a small probability (similar to `reduce128`). In theory leaks the input to the hash function, and we believe SHOULD NOT lead to a practical attack with the attack vectors discussed in this work.	During the time of writing this work Plonky3 finished its security audit and became production ready. We did not find constant time implementation as a part of the threat model.

4.5 Leakages in Zero-Knowledge Protocols

Table 3 shows a non-exhaustive list of ZK frameworks, spread across various ZK applications using various non-constant time Poseidon implementations. Most of the vulnerabilities root from the underlying field arithmetic library, except for ZK-kit, where they implement their own field library. In case of Panther and Sui, the underlying jdk native BigInt implementation is the source of leakage, known since 2020 [22]. The poseidon-lite library acknowledges the timing leakage in its disclaimer, however, in both the Sui and the Panther ZK framework such declarations were found missing. Currently, we looked into only a handful of the ZK frameworks which use such leaky field libraries, however, we suspect that there are several other frameworks using them as well.

Even though most of the underlying field libraries state that they are actually vulnerable to timing attacks and thus should not be used for production, however, at the time of writing this paper, they were being used in many ZK libraries. Moreover, to the best of our knowledge, none of the ZK library security audits considered timing attacks in their threat models, which we believe in the future should be taken into consideration as a standard practice as is done in other areas of cryptographic implementations, including hardware designs.

We also looked into other ZK protocols for potential timing leakages, in particular Plonky2, Plonky3 and Halo2. We found branching in the addition and the subtraction operations in both Plonky2 and Plonky3 using Poseidon-GL. However, the authors claimed, and we also independently verified from their public implementation, that this branching happens with a low probability (however still large in the security parameter). In particular, for this to occur, there must exist a pair of a round constant and an input both $\geq (2^{64} - 1) - (2^{30} - 1)$, and hence it is more efficient to branch. In Halo2, we found the Poseidon S-box layer implementation to contain a non-constant time exponentiation `pow_vartime()`, but the exponent is already public and thus does not affect the security. Special care must be taken in the future to not use this function for any secret dependent operation. We also looked into Aztec [7], and to the best of our knowledge, at the time of writing this paper, we did not find any non-constant time Poseidon implementation.

At the time of writing this paper, we did not attack any ZK proof system in production and thus based on our particular findings we can only recommend switching to constant time implementation to mitigate any potential future risk associated with timing leakages, especially considering that the performance penalties are not significant [42] (Table 4). With this work, we are providing forked versions[16] of the `ff`, `ff-ce` and `ark-ff` crates which will include constant-time implementations of the functions discuss in Table 3. As a future collaboration with (but not limited to) the maintainers of the above libraries, we are currently looking into the possibility of integrating our constant-time proposal into the standard ZK libraries.

End-to-End Attacks

Any serious security mitigation step first requires assessing the practicality of a possible end-to-end attack. This work primarily focuses on hinting towards potential timing leakages in ZK protocols and its implications. No end-to-end attack has been demonstrated in this work, however, after discussing with the side channel community, we understand that once an attack works on an isolated target, that is, in our case the non-constant time field

[16] `https://extgit.isec.tugraz.at/krypto/ffconstzksca`

arithmetic libraries used to implement hash functions in zk protocols, a full attack is merely an engineering problem, similar to other side-channel challenges like handling noisy cloud environment during measurements, among other. As the ZK industry grows with more money at stake, the engineering motivation increases. On the contrary, an end-to-end side channel attack might never be even feasible, as in a personal conversation with a side-channel veteran, it is often cheaper to buy off-the-shelf zero-day exploits with system level or application level vulnerabilities. We believe, research on cache timing attacks, even if not financially practical, still plays an important role in identifying these threats and plug the gaps, like in the case of OpenSSL AES table look-ups [17].

Other Timing Attacks

While this work focuses on the well-known F+R and P+P timing attacks on ZKP protocols and their applications. We conjecture that more sophisticated attacks like [61, 76, 27, 105, 85, 40, 117] will reduce the preimage recovery complexity significantly, especially for lookup-based hash functions like Reinforced Concrete and potentially for Tip5. Power side-channel attacks might be another interesting direction to explore.

Disclosures

This work looks into the potential cache timing side-channel vulnerabilities in the current ZKP systems. Even though we do not show any real-world attack on the existing ZK applications, however, following our footsteps, in the future one may find real-world exploits, potentially causing financial loss and privacy violation, among other damages. We have already informed some of the industry groups about our findings, however, it is infeasible for us to inform every potentially affected group as the scope of our attack spreads across a wide range of ZK applications. The informed industry groups have already acknowledged our findings, and we hope mitigations are out soon.

Conclusion

Non-constant time implementations leak sensitive input data in ZK protocols. Even though attacks like P+P and F+R were introduced several years ago, one still finding potential exploits using them due to non-constant time implementations even in fairly recent protocols like ZKP and their applications. As ZK use-cases mature, side-channel protection will be more crucial. The first step in this direction could be to follow the Intel guidelines on mitigation against timing attacks [58, 59], including constant time implementation of any secret-dependent code. In our context, we recommend switching to a constant-time field arithmetic libraries for implementing constant time algorithm hash functions like Poseidon. While this can be challenging and may not result in the most performant implementation, even though the performance penalties are small, it is important from a security perspective and should be treated as a minimum requirement. In case of non-constant time hash algorithms like Reinforced Concrete and Tip5 (the ones relying on look-up-tables in plain), one can switch to newer generation of look-up-based hash functions like Monolith [42] and Skyscraper [21], with constant time plain S-box evaluation while still supporting look-up feature in ZK. Furthermore, one may also adopt free and easy-to-use tools like the DATA framework [119] for conducting initial checks for any secret-dependent cache-based timing leakages.

It is well-known that constant-time implementations often do not translate into constant-time assembly code due to compiler optimizations which are often unpredictable. The leakages may also originate from other sources like described in [27] and often for better constant time guarantees, protocols, or at least the secret dependent components, can be directly written in `asm`. Moreover, security audits of the ZK libraries may also take into consideration basic cache timing leakages in their security model. One can also switch to dedicated side-channel secure hardware for generating zero-knowledge proofs. Combining multi-party-computation with ZK, also known as coSNARKS can be another potential candidate to protect against side-channel attacks on ZK. Learning from the FRI protocols providing a natural protection against SCA due to the inverse-interpolation step, developing generic ZK side-channel counter measures on a similar path can also be an interesting direction to look into for other protocols.

Table 4 Runtime comparison between the constant time and non-constant time implementation of Poseidon-M and Poseidon-GL [45]. Benchmarked on Intel Core i5-12450H running WSL2.0 with Ubuntu 22.04.3 LTS.

Hash function	Non-constant time	Constant time	% Slower
Poseidon-(M, $t = 16$)	4.577 µs	4.996 µs	9%
Poseidon-(M, $t = 24$)	10.183 µs	10.929 µs	7%
Poseidon-(GL, $t = 8$)	1.997 µs	2.318 µs	16%
Poseidon-(GL, $t = 12$)	3.708 µs	4.057 µs	9%

References

1 Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2002. doi:10.1007/3-540-36400-5_4.

2 Miguel Ambrona, Marc Beunardeau, Anne-Laure Schmitt, and Raphael R. Toledo. a*PlonK*: Aggregated *PlonK* from multi-polynomial commitment schemes. In Junji Shikata and Hiroki Kuzuno, editors, *Advances in Information and Computer Security - 18th International Workshop on Security, IWSEC 2023, Yokohama, Japan, August 29-31, 2023, Proceedings*, volume 14128 of *Lecture Notes in Computer Science*, pages 195–213. Springer, 2023. doi:10.1007/978-3-031-41326-1_11.

3 Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2087–2104. ACM, 2017. doi:10.1145/3133956.3134104.

4 Gorka Irazoqui Apecechea, Thomas Eisenbarth, and Berk Sunar. S$a: A shared cache attack that works across cores and defies VM sandboxing - and its application to AES. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 591–604. IEEE Computer Society, 2015. doi:10.1109/SP.2015.42.

5 ark ff. ark-ff. https://docs.rs/ark-ff/0.4.2/ark_ff/.

6 Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: reed-solomon proximity testing with fewer queries. *IACR Cryptol. ePrint Arch.*, page 390, 2024. URL: https://eprint.iacr.org/2024/390.

7 Aztec. Aztec. https://aztec.network/.

8 Carsten Baum, Ward Beullens, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. One tree to rule them all: Optimizing GGM trees and owfs for post-quantum signatures. *IACR Cryptol. ePrint Arch.*, page 490, 2024. URL: https://eprint.iacr.org/2024/490.

9 Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In Juan A. Garay, editor, *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 266–297. Springer, 2021. doi:10.1007/978-3-030-75245-3_11.

10 Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In *ICALP*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.14.

11 Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, page 46, 2018. URL: http://eprint.iacr.org/2018/046.

12 Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, page 46, 2018. URL: http://eprint.iacr.org/2018/046.

13 Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474. IEEE Computer Society, 2014. doi:10.1109/SP.2014.36.

14 Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, 2019. doi:10.1007/978-3-030-17653-2_4.

15 Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive Oracle Proofs. In *TCC (B2)*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, 2016. doi:10.1007/978-3-662-53644-5_2.

16 Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 1993. doi:10.1007/3-540-48285-7_24.

17 Daniel J Bernstein. Cache-timing attacks on aes, 2005.

18 Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 2025–2038. ACM, 2020. doi:10.1145/3372297.3417893.

19 Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349. ACM, 2012. doi:10.1145/2090236.2090263.

20 Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 595–626. Springer, 2018. doi:10.1007/978-3-030-03326-2_20.

21 Clémence Bouvier, Lorenzo Grassi, Dmitry Khovratovich, Katharina Koschatko, Christian Rechberger, Fabian Schmid, and Markus Schofnegger. Skyscraper: Fast hashing on big primes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(2):743–780, March 2025. doi:10.46586/tches.v2025.i2.743-780.

22 Tegan Brennan, Nicolás Rosner, and Tevfik Bultan. JIT leaks: Inducing timing side channels through just-in-time compilation. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1207–1222. IEEE, 2020. doi:10.1109/SP40000.2020.00007.

23 David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003. URL: https://www.usenix.org/conference/12th-usenix-security-symposium/remote-timing-attacks-are-practical.

24 Stefanos Chaliasos, Jens Ernstberger, David Theodore, David Wong, Mohammad Jahanara, and Benjamin Livshits. Sok: What don't we know? understanding security vulnerabilities in snarks. In Davide Balzarotti and Wenyuan Xu, editors, *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/chaliasos.

25 Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002. doi:10.1007/3-540-36400-5_3.

26 Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1825–1842. ACM, 2017. doi:10.1145/3133956.3133997.

27 Boru Chen, Yingchen Wang, Pradyumna Shome, Christopher W Fletcher, David Kohlbrenner, Riccardo Paccagnella, and Daniel Genkin. Gofetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers. In *Proc. USENIX Secur. Symp*, pages 1–21, 2024.

28 Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768. Springer, 2020. doi:10.1007/978-3-030-45721-1_26.

29 Electric Coin. Halo2. https://zcash.github.io/halo2/index.html.

30 Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. Shorter signatures based on tailor-made minimalist symmetric-key crypto. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 843–857. ACM, 2022. doi:10.1145/3548606.3559353.

31 Dusk. Dusk network. https://dusk.network/.

32 Ethereum. Ethereum: A secure decentralised generalised transaction ledger. ethereum project yellow paper. https://ethereum.github.io/yellowpaper/paper.pdf.

33 Fabric. Fabric cryptography. https://www.fabriccryptography.com/.

34 ff. ff crate. https://docs.rs/ff/0.13.0/ff/.

35 Nil Foundation. Nil;'s zkevm1. https://cms.nil.foundation/uploads/zk_EVM_1_7d6f8caa16.pdf.

36 Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. *IACR Cryptol. ePrint Arch.*, page 315, 2020. URL: https://eprint.iacr.org/2020/315.

37 Lukas Giner, Roland Czerny, Christoph Gruber, Fabian Rauscher, Andreas Kogler, Daniel De Almeida Braga, and Daniel Gruss. Generic and automated drive-by GPU cache attacks from the browser. In Jianying Zhou, Tony Q. S. Quek, Debin Gao, and Alvaro Cardenas, editors, *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2024, Singapore, July 1-5, 2024*. ACM, 2024. doi:10.1145/3634737.3656283.

38 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985. doi:10.1145/22145.22178.

39 Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic snarks for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 193–226. Springer, 2023. doi:10.1007/978-3-031-38545-2_7.

40 Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 955–972. USENIX Association, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/gras.

41 Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Reinforced concrete: A fast hash function for verifiable computation. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1323–1335. ACM, 2022. doi:10.1145/3548606.3560686.

42 Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Hash functions monolith for ZK applications: May the speed of SHA-3 be with you. *IACR Cryptol. ePrint Arch.*, page 1025, 2023. URL: https://eprint.iacr.org/2023/1025.

43 Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 519–535. USENIX Association, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi.

44 Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a generalization of substitution-permutation networks: The HADES design strategy. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 674–704. Springer, 2020. doi:10.1007/978-3-030-45724-2_23.

45 IAIK TU Graz. Zk friendly hash zoo. https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo.

46 Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016. doi:10.1007/978-3-662-49896-5_11.

47 Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A remote software-induced fault attack in javascript. In Juan Caballero, Urko Zurutuza, and Ricardo J. Rodríguez, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment - 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings*, volume 9721 of *Lecture Notes in Computer Science*, pages 300–321. Springer, 2016. doi:10.1007/978-3-319-40667-1_15.

48 Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+flush: A fast and stealthy cache attack. In Juan Caballero, Urko Zurutuza, and Ricardo J. Rodríguez, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment - 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings*, volume 9721 of *Lecture Notes in Computer Science*, pages 279–299. Springer, 2016. doi:10.1007/978-3-319-40667-1_14.

49 Ulrich Haböck. Multivariate Lookups Based on Logarithmic Derivatives. *IACR Cryptol. ePrint Arch.*, page 1530, 2022. URL: https://eprint.iacr.org/2022/1530.

50 Ulrich Haböck and Al Kindi. A note on adding zero-knowledge to STARKs. *IACR Cryptol. ePrint Arch.*, page 1037, 2024. URL: https://eprint.iacr.org/2024/1037.

51 Ulrich Haböck, David Levit, and Shahar Papini. Circle starks. *IACR Cryptol. ePrint Arch.*, page 278, 2024. URL: https://eprint.iacr.org/2024/278.

52 Hermez. Polygon hermez. https://hermez.io/.

53 Iden3. iden3/circomlibjs git. https://github.com/iden3/circomlibjs.

54 Iden3. iden3/ffjavascript git. https://github.com/iden3/ffjavascript.

55 Mehmet Sinan Inci, Berk Gülmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Cache attacks enable bulk key recovery on the cloud. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 368–388. Springer, 2016. doi:10.1007/978-3-662-53140-2_18.

56 Ingonyama. Ingonyama. https://www.ingonyama.com/.

57 Ingonyama. ingonyama-zk/poseidon-hash git. https://github.com/ingonyama-zk/poseidon-hash.

58 Intel. Guidelines for mitigating timing side channels against cryptographic implementations. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/mitigate-timing-side-channel-crypto-implementation.html.

59 Intel. Security best practices for side channel resistance. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/security-best-practices-side-channel-resistance.html.

60 Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010. doi:10.1007/978-3-642-17373-8_11.

61 Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: exploiting speculative execution. *Commun. ACM*, 63(7):93–101, 2020. doi:10.1145/3399742.

62 Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. doi:10.1007/3-540-68697-5_9.

63 Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999. doi:10.1007/3-540-48405-1_25.

64 Lurk Lab. lurk-lab/neptune git. https://github.com/lurk-lab/neptune.

65 Light Protocol Labs. Light protocol. https://docs.lightprotocol.com/.

66 Matter Labs. Matter labs. https://matter-labs.io/.

67 Matter Labs. matter-labs/rescue-poseidon git. https://github.com/matter-labs/rescue-poseidon.

68 Mysten Labs. mystenlabs/sui git. https://github.com/MystenLabs/sui.

69 O1 Labs. o1-labs/proof-systems git. https://github.com/o1-labs/proof-systems.

70 Polygon Labs. 0xpolygonzero/plonky2 git. https://github.com/0xPolygonZero/plonky2.

71 Polygon Labs. Plonky3/plonky3 git. https://github.com/Plonky3/Plonky3.

72 Protocl Labs. Filecoin. https://filecoin.io/filecoin.pdf.

73 Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 1–34. Springer, 2021. doi:10.1007/978-3-030-90453-1_1.

74 Libgcrypt. Libgcrypt. https://gnupg.org/software/libgcrypt/index.html.

75 Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021.

76 Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 973–990. USENIX Association, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp.

77 Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 605–622. IEEE Computer Society, 2015. doi:10.1109/SP.2015.43.

78 Loopring. Loopring. https://loopring.org/#/.

79 Lurk. Lurk. https://lurk-lang.org/.

80 Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2111–2128. ACM, 2019. doi:10.1145/3319535.3339817.

81 Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

82 Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 1999. doi:10.1007/3-540-48059-5_14.

83 mhostetter. galois git. https://github.com/mhostetter/galois.

84 Mina. Mina protocol. https://docs.minaprotocol.com/assets/technicalWhitepaper.pdf.

85 Ahmad Moghimi, Jan Wichelmann, Thomas Eisenbarth, and Berk Sunar. Memjam: A false dependency attack against constant-time crypto implementations. *Int. J. Parallel Program.*, 47(4):538–570, 2019. doi:10.1007/S10766-018-0611-9.

86 Monero. The monero project. https://www.getmonero.org/.

87 NSS. Nss. https://github.com/nss-dev/nss.

88 OpenJDK. openjdk/jdk git. https://github.com/openjdk/jdk/tree/master.

89 openSSL. openssl. https://www.openssl.org/.

90 Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In David Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006. doi:10.1007/11605805_1.

91 Panther. Panther protocol. https://www.pantherprotocol.io/.

92 Luke Pearson, Joshua Brian Fitzgerald, Héctor Masip, Marta Bellés-Muñoz, and Jose Luis Muñoz-Tapia. Plonkup: Reconciling plonk with plookup. *IACR Cryptol. ePrint Arch.*, page 86, 2022. URL: https://eprint.iacr.org/2022/086.

93 Polygon. Polygon miden technologies. https://polygon.technology/polygon-miden.

94 Polygon. Polygon technologies. https://polygon.technology/.

95 Light Protocol. Lightprotocol/light-poseidon git. https://github.com/Lightprotocol/light-poseidon.

96 Light Protocol. Lightprotocol/light-protocol git. https://github.com/Lightprotocol/light-protocol.

97 Panther Protocol. pantherprotocol/panther-core git. https://github.com/pantherprotocol/panther-core.

98 Mark Randolph and William Diehl. Power side-channel attack analysis: A review of 20 years of study for the layman. *Cryptogr.*, 4(2):15, 2020. doi:10.3390/CRYPTOGRAPHY4020015.

99 RISCZero. Risczero. https://www.risczero.com/.

100 Scroll. Scroll. https://scroll.io/.

101 Semaphore. Semaphore. https://semaphore.pse.dev/whitepaper-v1.pdf.

102 Srinath T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737. Springer, 2020. doi:10.1007/978-3-030-56877-1_25.

103 Anatoly Shusterman, Ayush Agarwal, Sioli O'Connell, Daniel Genkin, Yossi Oren, and Yuval Yarom. Prime+probe 1, javascript 0: Overcoming browser-based side-channel defenses. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 2863–2880. USENIX Association, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/shusterman.

104 Sia. Sia. https://sia.tech/sia.pdf.

105 Florian Sieck, Zhiyuan Zhang, Sebastian Berndt, Chitchanok Chuengsatiansup, Thomas Eisenbarth, and Yuval Yarom. Teejam: Sub-cache-line leakages strike back. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(1):457–500, 2024. doi:10.46586/TCHES.V2024.I1.457-500.

106 Solana. Solana. https://solana.com/.

107 Sovrin. Sovrin. https://sovrin.org/.

108 Starkware. Starkware. https://starkware.co/.

109 Storj. Storj. https://www.storj.io/storjv3.pdf.

110 Sui. Sui. https://sui.io/zklogin.

111 Supranational. Supra national. https://www.supranational.net/.

112 Swarm. Swarm. https://www.ethswarm.org/swarm-whitepaper.pdf.

113 Alan Szepieniec, Alexander Lemmens, Jan Ferdinand Sauer, and Bobbin Threadbare. The tip5 hash function for recursive starks. *IACR Cryptol. ePrint Arch.*, page 107, 2023. URL: https://eprint.iacr.org/2023/107.

114 Florian Tramèr, Dan Boneh, and Kenny Paterson. Remote side-channel attacks on anonymous transactions. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2739–2756. USENIX Association, 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/tramer.

115 Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on aes, and countermeasures. *J. Cryptol.*, 23(1):37–71, 2010. doi:10.1007/S00145-009-9049-Y.

116 Tusima. Tusima. https://www.tusima.network/.

117 Stephan van Schaik, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Malicious management unit: Why stopping cache attacks in software is harder than you think. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 937–954. USENIX Association, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/van-schaik.

118 Vimwitch. poseidon-lite git. https://github.com/vimwitch/poseidon-lite.

119 Samuel Weiser, Andreas Zankl, Raphael Spreitzer, Katja Miller, Stefan Mangard, and Georg Sigl. DATA - differential address trace analysis: Finding address-based side-channels in binaries. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 603–620. USENIX Association, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/weiser.

120 Witnerfell. Winterfell git. https://github.com/facebook/winterfell/tree/main.

121 Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 299–328. Springer, 2022. doi:10.1007/978-3-031-15985-5_11.

122 Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 719–732. USENIX Association, 2014. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom.

123 Yuval Yarom, Daniel Genkin, and Nadia Heninger. Cachebleed: A timing attack on openssl constant time RSA. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 346–367. Springer, 2016. doi:10.1007/978-3-662-53140-2_17.

124 Zcash. Zcash protocol specification. https://zips.z.cash/protocol/protocol.pdf.

125 ZK-kit. Zk-kit. https://zkkit.pse.dev/.

126 zk kit. zk-kit git. https://github.com/privacy-scaling-explorations/zk-kit.

127 ZKsync. Zksync. https://zksync.io/.

Zero-Knowledge Authenticator for Blockchain: Policy-Private and Obliviously Updateable

Kostas Kryptos Chalkias
Mysten Labs, Palo Alto, CA, USA

Deepak Maram
Mysten Labs, Palo Alto, CA, USA

Arnab Roy
Mysten Labs, Palo Alto, CA, USA

Joy Wang
Mysten Labs, Palo Alto, CA, USA

Aayush Yadav
George Mason University, Fairfax, VA, USA

Abstract

Transaction details and participant identities on the blockchain are often publicly exposed. In this work, we posit that blockchain's transparency should not come at the cost of privacy. To that end, we introduce *zero-knowledge authenticators* (zkAt), a new cryptographic primitive for privacy-preserving authentication on public blockchains. zkAt utilizes zero-knowledge proofs to enable users to authenticate transactions, while keeping the underlying authentication policies private.

Prior solutions for such *policy-private authentication* required the use of threshold signatures, which can only hide the threshold access structure itself. In comparison, zkAt provides privacy for *arbitrarily complex* authentication policies, and offers a richer interface even within the threshold access structure by, for instance, allowing for the combination of signatures under distinct signature schemes.

In order to construct zkAt, we design a compiler that transforms the popular Groth16 non-interactive zero knowledge (NIZK) proof system into a NIZK with equivocable verification keys, a property that we define in this work. Then, for any zkAt constructed using proof systems with this new property, we show that all public information must be independent of the policy, thereby achieving policy-privacy.

Next, we give an extension of zkAt, called zkAt$^+$ wherein, assuming a trusted authority, policies can be updated obliviously in the sense that a third-party learns no new information when a policy is updated by the policy issuer. We also give a theoretical construction for zkAt$^+$ using recursive NIZKs, and explore the integration of zkAt into modern blockchains. Finally, to evaluate their feasibility, we implement both our schemes for a specific threshold access structure. Our findings show that zkAt achieves comparable performance to traditional threshold signatures, while also attaining privacy for significantly more complex policies with very little overhead.

2012 ACM Subject Classification Security and privacy → Cryptography; Security and privacy → Authentication; Security and privacy → Privacy-preserving protocols

Keywords and phrases Blockchain privacy, authentication schemes, threshold wallets, zero knowledge proofs

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.2

Related Version *Full Version*: `https://ia.cr/2025/921`

Acknowledgements The authors thank Foteini Baldimtsi for her feedback on the manuscript.

© Kostas Kryptos Chalkias, Deepak Maram, Arnab Roy, Joy Wang, and Aayush Yadav;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Blockchain technology, with its decentralized architecture, offers a transparent and immutable ledger of transactions, fostering trust among participants without the need for centralized intermediaries. To authenticate a blockchain transaction, a user creates a digital signature on the transaction. If said signature verifies under the user's signature verification key, then the transaction is permanently added to the blockchain by the validators.

This transparency, however, often comes at the cost of privacy, as transaction details, participant identities, and contract logic are exposed to public scrutiny on most blockchains. While there has been significant progress in enabling confidential transactions with privacy-preserving blockchains, such as Zcash and Monero, and mixing services [28, 20, 19, 30, 27], practical privacy concerns extend beyond just transaction details. For instance, public access to account authentication logic makes it easier for malicious actors to design targeted attacks. Consider, for example, threshold multi-signature [32], which is a popular authentication mechanism supported by most blockchains for protecting high-value transactions. In threshold multi-signature, n different signing keys are generated such that signatures under at least $t \leq n$ of those keys are needed to authorize a transaction. However, this reveals the access structure (n, t) giving critical information to attackers who learn exactly how many accounts they need to compromise.

In this work, we investigate approaches to conceal the account authentication logic (i.e., the policy) on smart-contract supporting public blockchains such as Ethereum, Solana, Aptos and Sui. A popular solution to hide the access structure is to enforce it *off-chain* with threshold signatures [17, 18] issued by a set of custodians or guardians. In such a threshold authenticator[1], a single key is secret-shared between n parties such that any t parties can generate partial signatures, and aggregate them into a complete signature. Crucially, this final signature looks identical to a signature generated by the original key. Thus, threshold authenticators hide the access structure (both t and n) and inherently obscure the identity of the signers within a group against blockchain transaction observers. This additional privacy makes them a preferable alternative to the aforementioned multi-signature authenticators, as noted by a recent user study [38].

The need for complex authentication policies. Threshold-wallets (i.e., cryptographic wallets with authentication enforced via threshold signatures), however, do not support the broad range of authentication needs seen in practice. A recent example is that of *account abstraction* [43] which allows for programmable access to user accounts via smart contracts instead of relying exclusively on private keys. For one, threshold-wallets – by definition – only support a threshold access structure rather than arbitrary boolean formulae. It is also not possible to combine existing keys created under different signature schemes, so for instance, it is not possible to create a 1-out-of-2 access structure between an ECSDA and an EdDSA key under existing threshold signature schemes. Further, some implementations of threshold EdDSA wallets can inadvertently reveal that the wallet is a threshold-wallet rather than a single key[2]!

In practice, authentication policy designers may want to use transaction data to select an appropriate access structure so as to balance usability and security concerns. Which is to say that they might, for instance, want to use 2-out-of-3 keys for high-value transactions above

[1] An authenticator is simply an authentication mechanism which encodes a general access structure in the form of an authentication policy.

[2] Many threshold EdDSA libraries introduce randomness in signatures on retries. So if different signatures for the same message are observed, it can reveal the use of a threshold-wallet [34].

a certain amount, and only 1-out-of-3 keys otherwise; or they might require authorization from a special administrator key for certain transactions. Rich contextual policies like these are commonly seen in smart-contract based authenticators [42, 3] (where the authentication policy is expressed in a smart contract) and off-chain authenticators [8, 24] (where the authentication policy is enforced off-chain by a trusted party). Finally, zero-knowledge proofs offer similarly complex authentication policies with some limited degree of privacy for the inputs to the policy, but not the policy itself [40].

To summarize, existing threshold authenticators offer some amount of privacy but only support very specific authentication policies. On the flip side, smart-contract authenticators leverage the rich language support to encode arbitrarily complex policies, but cannot hide the authentication policy; and zero-knowledge proofs offer only limited privacy but for general authentication policies. This is also summarized in Table 1 below.

This observation leads to the central question that motivates this work: *can we build private authenticators that are capable of hiding arbitrarily complex policies?*

Our results. To address this question, we design a new class of authentication schemes to validate user transactions on the blockchain while hiding the underlying (authentication) policy. Since they leverage zero-knowledge proof systems as the core building block, we refer to these schemes as *zero-knowledge authenticators* (zkAt). We summarize our contributions below:

(1) **Formalizing zero-knowledge authenticators:** we formalize the notion of zero-knowledge authenticators with policy-privacy for policies expressible as general NP statements. Specifically, the property of policy-privacy guarantees that an adversary learns nothing about the underlying policy besides its public inputs.
(2) **Constructing zero-knowledge authenticators:** we give a practical construction for zkAt by building on the seminal work of Groth [29], known colloquially as simply "Groth16". More precisely, we define the property of verification-key equivocation and show that non-interactive zero-knowledge proof systems (NIZK) with this property can be generically used to instantiate zkAt. Then, we modify the Groth16 construction so that it satisfies verification-key equivocation, thus yielding a zkAt.
(3) **Oblivious policy updates:** once policy-privacy has been achieved, a second interesting question emerges: *can policies be updated without leaking any new public information?* As our third contribution, we resolve this question by introducing zkAt$^+$, which extends our standard zkAt by additionally allowing for such oblivious policy updates. We also give a theoretical construction for zkAt$^+$ using recursive NIZKs.
(4) **Demonstrating practicality:** we implement both zkAt and zkAt$^+$, and evaluate them against similar (but less expressive) threshold signatures. Our results demonstrate that zkAt offers comparable performance, even for more complex policy semantics, and as the underlying NIZK is a succinct argument of knowledge (zk-SNARK), the final proof size is independent of the policy.

1.1 Application overview

Our stated goal is to build a private authenticator for transactions on public blockchains. More precisely, we want to build an authenticator for arbitrarily complex policies, such that the underlying policy remains hidden from all parties that did not generate the policy and/or the proof. Before delving into the technical details, we explore two scenarios where zkAt can be applied.

Table 1 A comparison between authenticators. "Expressiveness" refers to whether complex policies can be specified. "Policy anonymity set" indicates the level of policy privacy achieved: None (no privacy), a pre-defined policy set (some privacy), all policies (full privacy).

Type	Expressiveness	Policy anonimity set	Oblivious updates
Multi-signature	Limited	None	N/A
Threshold	Limited	Pre-defined	N/A
Smart-contract	Rich	None	N/A
Zero-knowledge	Rich	Pre-defined	✗
zkAt	Rich	All	✗
zkAt$^+$	Rich	All	✓

(1) **Delegated transactions.** High-level executives at an organization hold shared custody of the organizational finances in the form of a threshold-wallet. The board members of the organization vote on the policies required to authenticate transactions made with the wallet. They might, for instance, require an authentication policy stating that all "large" transactions initiated by the company must be signed by all the executives and at least 50% of the board. For privately-held organizations, it may be in their interest to keep such policies hidden from the public as an added layer of protection for the organizational funds.

(2) **Self-custody solutions.** As another example, consider an individual user who wants to protect their assets using a policy that requires, for instance, the user's valid JSON web tokens (JWT) issued by two-out-of-three OpenID providers (cf. [4] for a discussion on JWT and OpenID) the user has previously registered, in order to authenticate a transaction.

In both examples, zkAt makes it more challenging for attackers to mount a successful attack by hiding all information about the authentication logic and access structure. In particular, in the first example, not only do attackers external to the organization not know how many signatures are needed to authenticate a transaction, they do not even know whether this threshold varies by amount and, if so, what the amount is! Similarly, in the second example, the attackers don't know which OpenID providers the user has registered and, by extension, which accounts they need to compromise.

▶ **Remark 1.1.** We wish to emphasize that, more generally, in the scenarios we envision, the authentication policies are hidden from the validators as well as other (public) third-parties. At first brush, this may seem counter-intuitive – how can a validator validate a transaction without knowing the underlying policy? Our claim is that there is no reason for the validator to know the policy at all! Whatever the policy may be, a validator's only concern is to ensure that the transaction satisfies it, i.e., given the transaction as input, the verification algorithm accepts under the policy issuer's verification key. Importantly, as both our examples illustrate, the policy issuer could be the user (prover) themselves, and it is in their interest to design strong authentication policies.

1.2 Technical overview

To illustrate our technical ideas, we will begin by first considering a simple approach where we only attempt to hide the user's secret credentials.

A simple first approach. The idea is to instantiate a general-purpose zero-knowledge proof system (such as [29, 25]) with the policy circuit as input to the setup. As a concrete example, to create a 1-out-of-2 multi-signature between two existing on-chain accounts while also hiding the identities of the two multi-signature participants, a user can proceed in the following manner:

- **Setup.** Create commitments c_1, c_2 to two addresses, i.e., $c_i = \mathsf{Commit}(\mathsf{addr}_i; r_i)$, where each address addr_i is a signature verification key for some signer. Create a designated-prover NIZK (DP-NIZK) proving key and (proof) verification key pair $(\mathsf{pk}, \mathsf{vk})$ for the policy P given by the relation,

$$\left\{ \begin{array}{l} x := (c_1, c_2, \mathsf{tx_{pb}}), \\ w := (\mathsf{addr}, r, \sigma, \mathsf{tx_{pv}}) \end{array} \quad : \quad \begin{array}{l} \left(\begin{array}{l} c_1 = \mathsf{Commit}(\mathsf{addr}; r) \vee \\ c_2 = \mathsf{Commit}(\mathsf{addr}; r) \end{array} \right) \\ \wedge\ \mathsf{Sig.Verify}(\mathsf{addr}, \mathsf{tx_{pb}} || \mathsf{tx_{pv}}, \sigma) = 1 \end{array} \right\},$$

  where $\mathsf{tx_{pb}}$ (resp. $\mathsf{tx_{pv}}$) is the public (resp. private) part of the transaction. The proof verification key vk acts as the user's on-chain address.
- **Signing.** Split the transaction into public and private components, as determined by the privacy requirements. Collect signature σ on the full transaction $\mathsf{tx_{pb}}||\mathsf{tx_{pv}}$ from one of the two accounts $\mathsf{addr} = \mathsf{addr}_1$ or $\mathsf{addr} = \mathsf{addr}_2$. Generate a ZK proof $\pi = \mathsf{ZK.Prove}(\mathsf{pk}, (c_1, c_2, \mathsf{tx_{pb}}), (\mathsf{addr}, r, \sigma, \mathsf{tx_{pv}}))$ proving that the policy is satisfied or, in other words, the signature verifies with one of the two addresses.

Thus the resulting signature is nothing but a zero-knowledge proof, and the privacy guarantee for the private inputs follows straightforwardly from the zero-knowledge property. Notably, the above sketch already permits more expressive policies than threshold signatures albeit without hiding the policy – the fact that P is a 1-out-of-2 authentication policy is known to the public, however the actual addresses remain hidden thanks to the hiding property of the commitment (as long as r_1 and r_2 are secret), as do the private inputs to the circuit.

Next, in order to hide the *policy* itself, the most obvious solution would be to universalize the circuit. For example, we could make the circuit do $n = \mathsf{poly}(\lambda)$ signature verifications irrespective of the policy. This would allow policies that use up to n signatures to use the same circuit. In effect, the "policy anonymity set" consists of all policies that use up to n signatures. While this straw-man approach works to an extent, it results in poor signing performance (even if someone's policy only uses $O(1)$ signatures, they need to verify all n to generate the signature). Moreover, as policies become more expressive, the policy anonymity set grows exponentially (if there are m possible triggers, a universal circuit would have to verify all the 2^m possibilities). Thus the signing overhead for our simple first approach scales proportionally to the size of the policy anonymity set, resulting in an unsatisfactory trade-off between privacy and performance.

Achieving efficient policy privacy. In order to achieve policy-privacy without sacrificing performance, we must ask a more fundamental question about our sketch above (without the universalized circuit): assuming that the proving key is held privately by the user, do the verification key and proof reveal any non-trivial information about the policy?

A priori, it is not at all obvious whether a zkAt is directly constructible from any existing NIZK schemes. So, as a first step, we must formalize our, presently abstract, policy-privacy property. To that end, we define a new property for NIZK proof systems, which we call *verification key equivocation* (vk-equivocation). At a high level, in the vk-equivocation experiment, an adversary – holding a proof verification key – must identify which of the two

policies (of its choice) was an honest proof created with respect to, for a statement (also of its choice) that satisfies both policies. We find this to be a novel and practically useful property that, to the best of our knowledge, has not been previously considered in designated proof systems. The crucial observation is that if the underlying NIZK has equivocable verification keys, or in other words if the verification key is independent of the underlying relation (which encodes the policy), then a zkAt (instantiated according to our simple first approach) hides the policy since the proof is already zero-knowledge!

Our next task then is to develop such a proof system with vk-equivocation. For this, we turn to the work of Groth [29], that describes a NIZK for quadratic arithmetic programs (QAPs) from pairing-based assumptions (cf. § 2.3 for a description of the scheme). In particular, we find that a simple modification to Groth16 is sufficient to achieve this property. At a high level, our main observation is that the only component linking the verification key to the underlying relation are the evaluations of the QAP polynomials at a random point x; and as it turns out, we can interpolate fresh polynomials that behave exactly as the original ones on the characteristic points (so that they describe the same arithmetic circuit), but additionally, also evaluate at x to *a priori* uniformly chosen values. These fresh polynomials now define an updated QAP. This essentially fixes the evaluation of the polynomials at x independently of the relation, and consequently the resulting verification key is made completely independent of the updated QAP. Most notably, this modification affects no change to the proof verification function, thus making it fully compatible with existing Groth16 verifiers!

The overall workflow for setting up zkAt keys should thus be:

1. Choose a policy and encode it into a circuit,
2. Run the modified Groth16 setup to generate the trapdoor, proving and verification keys,
3. Delete the trapdoor (as we explain shortly, if storing a sensitive secret is acceptable and oblivious policy updates are desired, then the trapdoor can be persisted); and
4. Store the proving key, and publish the verification key as the on-chain address.

Interestingly, unlike in most other use cases of Groth16, the presence of a trusted setup phase is not a problem. This is because it is in the interest of the user, who is the trusted setup generator, to delete the trapdoor safely as otherwise their account could be compromised.

Updating policies obliviously. Policy issuers may want to be able to update their policies for any number of reasons, such as for operational reasons such as periodic key rotation. A trivial way to do this, of course, is to issue a fresh set of keys with respect to the new policy. However, recall that the verification key acts as the user's on-chain address, and thus the trivial approach would require updating user addresses every time a policy is changed. It is therefore desirable to grant policy issuers the ability to update policies without having to change the verification keys. We call this feature *oblivious updateability*.

Interestingly, our Groth16-zkAt already achieves oblivious updateability in a limited sense – the idea is to retain the trapdoor for our modified proof system (in cold storage), so that new proving keys can be generated at will. Old policies and the corresponding proving keys can then be retired by adding a clause within the policy circuit to check that the current time is less than a fixed expiry time (this assumes that the current time is accessible as a public input, a feature commonly available on most blockchains).

However, persisting the trapdoor carries significantly more risk as a leaked trapdoor breaks security. Moreover, we only know our Groth16-zkAt to be securely updateable assuming that an attacker cannot access two different proving keys corresponding to the same verification

key, but in situations such as oblivious policy updates, this is not necessarily the case – a user (prover) with two proving keys for the same verification key can learn non-trivial infomation about the trapdoor and possibly break soundness of the NIZK.

Maliciously-secure oblivious policy updates. The upshot of this is that we must extend our zkAt definition so as to realize maliciously-secure policy updates in the strongest sense, i.e., one where the adversary is given access to an update oracle, which returns updated proving keys (corresponding to the same verification key) for policy updates of the adversary's choice. We call the extended primitive that satisfies the stronger update security, zkAt$^+$.

The main technical challenge in constructing a zkAt$^+$ is to somehow fix the proof verification key across policy updates since it acts as the user's on-chain address while *securely* updating the proving key. As we just explained, however, the Groth16-zkAt approach does not quite work, so we approach this problem from a new direction. As usual, during authentication, the user will still compute a proof π_I that the transaction satisfies the underlying policy using a (not necessarily designated-prover) NIZK. The astute reader may observe that the user cannot send this proof in the clear, since it obviously contains information about the policy. Indeed, instead the user recursively composes this proof with another "outer" NIZK proof π_O, essentially proving that it has a proof that the transaction satisfies the policy. Notably, as this outer NIZK has the same structure for any policy, there need only be a single global common reference string (CRS), crs_O that can be used to generate and verify outer proofs for *all* users.

Observe, also, that the policy can now be updated obliviously, since crs_O does not depend on any policy specific information. Unfortunately, this construction is incomplete, since there are no clear candidate public keys that could play the role of a user address. One might again be tempted to set the "inner" NIZK proof's verification key as the user's address, but remember that this is not possible if we want oblivious updates. Instead of a proof verification key acting as the user's on-chain address, we let it be given by a *signature* verification key vk, for a signing-verification key pair $(\mathsf{sk}, \mathsf{vk})$ generated by the user. Now, in addition to the inner proof π_I, the user must additionally compute a signature σ on the inner proof's CRS, crs_I, and then, in the outer proof, also prove that σ verifies under vk. Thus, the public instance for the outer NIZK is the address vk along with any other public transaction data, and the private witness is crs_I, π_I and σ.

In order to update a policy, the policy issuer simply computes a fresh crs_I for the new policy and signs it with its secret signing key sk. Clearly, this reveals no information regarding the update on the chain. Moreover, this construction has maliciously-secure policy updates by soundness of the both NIZKs.

We remark that in applications where the policy issuers are distinct from the users (which, recall, need not always be the case) there does arise a subtle issue with this approach, namely that a user holding an older proving key can still authenticate with respect to that policy. This, however, is easily circumvented by having the policy issuer additionally sign an arbitrarily chosen tag that the user must prove belongs to a public set of currently accepted tags, and has the benefit of allowing policies to expire gracefully.

1.3 Related Work

Existing blockchain authentication methods. Threshold [17, 18, 33, 6, 41, 2] and multi-signing [39, 9] are commonly used authentication mechanisms in blockchain settings. Our solution offers all the same benefits of threshold-based solutions such as privacy of signers, compact signatures while also capturing more complex policy semantics beyond the threshold.

Moreover, with our zkAt, one can do this with *pre-existing* signing keys and without requiring any expensive distributed key-generation, a pre-requisite for threshold signatures. Smart-contract based authenticators [3, 42] (sometimes called account abstraction wallets [43]) are popular for their flexibility and security features, for example, account recovery, flexible policies for high-value transactions, ability to change policies over time. However, these offer no notion of privacy. Interestingly, zkAt can be used to turn any smart-contract based authenticator private.

Attribute-based authentication. Another common authentication mechanism is based on user attributes satisfying certain pre-specified criteria [37, 35]. Indeed, this is nothing but a policy-based authentication mechanism with the important distinction that the authentication policy need not be private. Put another way, one may view zkAt as an extension of attribute-based authentication that offers stronger privacy guarantees for not just the user's attributes, but also the constraints on those attributes.

Functional commitments. A functional commitment scheme [36] enables a user to succinctly commit to a function (from a specified family), such that the user can later verifiably reveal values of the function at desired inputs. Such a commitment must be *binding* to the function and may additionally also *hide* [10] it. We observe that zkAt realizes a sort of function-private functional commitment scheme for functions with binary outputs, with vk being the commitment to the policy function. Perhaps for the first time, our equivocable Groth16 construction gives a functional commitment where the underlying function is updateable or equivocable (given the trapdoor) without changing the commitment.

2 Preliminaries

In this section, we provide preliminaries needed for this work.

Notation. Let λ denote the security parameter, and PPT denote probabilistic polynomial-time. We use $\leftarrow\!\!\$$ to denote the output of a randomized algorithm, $\leftarrow$ to denote output of a deterministic algorithm, and $\coloneqq$ for assignment. For our security definitions, we use notation similar to [29].

Following common convention we use lowercase bold-face letters to denote vectors and uppercase bold-face letters for matrices. For a vector $\mathbf{x}$, x_i denotes its i^{th} element. In general $\mathbb{G}$ denotes a group and $\mathbb{F}$ a field. Let g_i be the generator of a group $\mathbb{G}_i$, then we write g_i^x for $x \in \mathbb{F}$ as $[x]_i$ and $a[x]_i \coloneqq g_i^{ax}$, for some $a \in \mathbb{F}$. As usual, $\mathbb{Z}$ is the ring of integers and $\mathbb{Z}_p$ is the ring of integers modulo p for some integer $p > 0$. Finally, $\mathbb{Z}_p[X]$ is the ring of polynomials with coefficients in $\mathbb{Z}_p$, and for any polynomial $U(X) \in \mathbb{Z}_p[X]$ the notation $\deg(U)$ denotes its degree.

Bilinear pairings. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be groups of the same prime order q with the generators g_1, g_2 respectively. Using the notation from [29], we denote the pairing map $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ as $=: [a]_1 \cdot [b]_2 = [ab]_T$ for any $a, b \in \mathbb{Z}_q$.

2.1 Cryptographic Building Blocks

We recall the definitions for some common cryptographic primitives that we use in our constructions.

2.1.1 Signature schemes

A signature scheme Sig over message space $\mathcal{M}$ consists of the following polynomial time algorithms:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$. is a randomized algorithm that takes security parameter λ as input and returns a pair of keys $(\mathsf{vk}, \mathsf{sk})$, where vk is the verification key and sk is the signing key.
- $\mathsf{Sign}(\mathsf{sk}, M) \to \sigma$. is a possibly randomized algorithm that takes as input the signing key sk, and a message $M \in \mathcal{M}$, and returns a signature σ.
- $\mathsf{Verify}(\mathsf{vk}, M, \sigma) \to \{0,1\}$. is a deterministic algorithm that takes as input the verification key vk, a message $M \in \mathcal{M}$, and a signature σ. It outputs 1 (accept) or 0 (reject).

A signature scheme satisfies correctness if for all $\lambda \in \mathbb{N}$, $M \in \mathcal{M}$, and every signing-verification key pair $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$, every signature $\sigma \leftarrow\!\$\ \mathsf{Sign}(\mathsf{sk}, M)$, $\mathsf{Verify}(\mathsf{vk}, M, \sigma) = 1$.

▶ **Definition 2.1** (Existential unforgeability under chosen messages). *A signature scheme* $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ *is existentially unforgeable under chosen messages if for every PPT attacker* $\mathcal{A}$ *there exists a negligible function* $\epsilon(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *the following probability is at most* $\epsilon(\lambda)$

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, M^*, \sigma^*) = 1 \ : \ \begin{array}{l}(\mathsf{vk}, \mathsf{sk}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) \\ (M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sk}}}(\mathsf{vk})\end{array}\right]$$

and $\mathcal{A}$ *should never have queried* M^* *to the signing oracle,* $\mathcal{O}_{\mathsf{sk}}(\cdot)$.

2.1.2 Non-interactive zero-knowledge

Let $\mathsf{C}_\lambda := \left\{\mathsf{C} : \{0,1\}^{\mathsf{poly}(\lambda)} \to \{0,1\}\right\}$ be a family of boolean circuits computable in polynomial time. Then, a non-interactive proof system for a circuit C_λ consists of the following polynomial time algorithms:

- $\mathsf{Setup}(1^\lambda, \mathsf{C}) \to (\mathsf{crs}, \tau)$. The setup algorithm takes as input the security parameter λ and a circuit C, and outputs a common reference string crs and a trapdoor τ.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$. The prover algorithm takes as input a crs, an instance x, and a witness w. It outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to 0/1$. The verification algorithm takes as input a crs, an instance x, and a proof π. It outputs 1 (accept) or 0 (reject).

▶ **Definition 2.2** (Non-interactive zero-knowledge). *Given a circuit* C*, a NIZK proof system for an* NP *relation* $\mathscr{R} = \{(x, w) : \mathsf{C}(x||w) = 1\}$ *must satisfy the following properties:*

- **Completeness:** *For every* $\lambda \in \mathbb{N}$*, and every crs computed as* $(\mathsf{crs}, \tau) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{C})$*, any instance and witness pair* $(x, w) \in \mathscr{R}$*,*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \ : \ \pi \leftarrow\!\$\ \mathsf{Prove}(\mathsf{crs}, x, w)\right] = 1.$$

- **Soundness:** *For every PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\epsilon(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *the following probability is at most* $\epsilon(\lambda)$*,*

$$\Pr\left[\ \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \wedge \ x \notin \mathscr{L}_{\mathscr{R}} \ : \ \begin{array}{l}(\mathsf{crs}, \tau) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs})\end{array}\right]$$

where $\mathscr{L}_{\mathscr{R}}$ *is the language specified by* $\mathscr{R}$.

- **Zero-knowledge:** *There exists a* PPT *simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *for every adversary* $\mathcal{A}$ *and such that there is a negligible function* $\epsilon(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$ *and every* $(x, w) \in \mathscr{R}$ *the following probability is at most* $\epsilon(\lambda)$*,*

$$\left| \Pr\left[\begin{array}{r} \mathcal{A}(\mathsf{crs}, x, \pi) = 1 : \\ (\mathsf{crs}, \tau) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ \pi \leftarrow\!\$\ \mathsf{Prove}(\mathsf{crs}, x, w) \end{array} \right] - \Pr\left[\begin{array}{r} \mathcal{A}(\mathsf{crs}, x, \pi) = 1 : \\ (\mathsf{crs}, \tau) \leftarrow \mathcal{S}_1(1^\lambda, \mathsf{C}) \\ \pi \leftarrow \mathcal{S}_2(\tau, x) \end{array} \right] \right|$$

Argument of Knowledge. Further, a NIZK proof system is an *argument of knowledge* if it satisfies the following additional property:

- **(Computational) Knowledge soundness:** For every PPT adversary $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}$ and a negligible function $\epsilon(\cdot)$, such that for all $\lambda \in \mathbb{N}$ the following probability is at most $\epsilon(\lambda)$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x^*, \pi^*) = 1 \wedge\ (x^*, w^*) \notin \mathscr{R} \quad : \quad \begin{array}{l} (\mathsf{crs}, \tau) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ ((x^*, \pi^*); w^*) \leftarrow (\mathcal{A} || \mathcal{E})(\mathsf{crs}) \end{array} \right]$$

  where the notation $((x^*, \pi^*); w^*) \leftarrow (\mathcal{A}||\mathcal{E})(\mathsf{crs})$, taken from [29], is shorthand for $(x^*, \pi^*) \leftarrow \mathcal{A}(\mathsf{crs})$ and $w^* \leftarrow \mathcal{E}(x^*, \pi^*)$ such that and $\mathcal{E}$ gets access to $\mathcal{A}$'s code.

Designated-prover NIZK. A (publicly verifiable) designated-prover NIZK scheme (DP-NIZK) for a circuit C consists of the following polynomial time algorithms:

- $\underline{\mathsf{Setup}(1^\lambda, \mathsf{C}) \to (\mathsf{vk}, \mathsf{pk}, \tau)}$. The setup algorithm takes as input the security parameter λ and a boolean circuit C, and outputs a verification key vk, a proving key pk and a trapdoor τ.
- $\underline{\mathsf{Prove}(\mathsf{pk}, x, w) \to \pi}$. The prover algorithm takes as input the proving key pk, an instance x, and a witness w. It outputs a proof π.
- $\underline{\mathsf{Verify}(\mathsf{vk}, x, \pi) \to 0/1}$. The verification algorithm takes as input a verification key vk, an instance x, and a proof π. It outputs 1 (accept) or 0 (reject).

A DP-NIZK scheme must further satisfy the same properties as described in Definition 2.2 with the crs appropriately replaced by pk and vk.

2.2 Quadratic arithmetic programs

A quadratic arithmetic program (QAP) comprises of a finite field $\mathbb{Z}_p$ for some prime p with $|p| = \lambda$, integers $\ell \leq m$, and polynomials $\{U_i(X), V_i(X), W_i(X)\}_{i=0}^m$ and $T(X)$ in $\mathbb{Z}_p[X]$ with $\deg(U_i), \deg(V_i), \deg(W_i) < \deg(T) = n$ (for all $i \in [0, m]$) such that, for $a_0 \coloneqq 1$, it defines the following relation

$$\left\{ \begin{array}{l} x \coloneqq (a_i)_{i \in [0,\ell]} \in \mathbb{Z}_p^\ell \\ w \coloneqq (a_i)_{i \in [\ell+1, m]} \in \mathbb{Z}_p^{m-\ell} \quad : \\ \quad \sum_{i=0}^m a_i U_i(X) \cdot \sum_{i=0}^m a_i V_i(X) = \sum_{i=0}^m a_i W_i(X) \mod T(X) \end{array} \right\}$$

In this work, we consider proof systems for satisfiability of general arithmetic circuits, which consist of addition and multiplication gates over the finite field $\mathbb{Z}_p$. Gennaro, et al. [26] gave an efficient technique for converting any arithmetic circuit into a QAP, thus allowing us to prove statements encoded as general arithmetic circuits using Groth16.

2.3 Recalling Groth16

Since it will be essential to one of our constructions, we now recall the Groth's NIZK argument for QAPs (and therefore for any arithmetic circuit).

Now, for some prime p such that $|p| = \lambda$, groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ such that the pairing $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map. Consider a QAP,

$$\mathscr{R} = \left\{ p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, g_1, g_2, \ell, \{U_i(X), V_i(X), W_i(X)\}_{i=0}^{m}, T(X) \right\} \ ,$$

that defines a field $\mathbb{Z}_p$ and a language of statements $(a_1, \ldots, a_\ell) \in \mathbb{Z}_p^\ell$ and witnesses $(a_{\ell+1}, \ldots, a_m) \in \mathbb{Z}_p^{m-\ell}$ such that for $a_0 = 1$, polynomials $\{U_i(X), V_i(X), W_i(X)\}_{i=0}^{m}$ and $T(X)$ in $\mathbb{Z}_p[X]$ with $\deg(U_i), \deg(V_i), \deg(W_i) < \deg(T) = n$ (for all $i \in [0, m]$) and $T(X) \coloneqq \prod_{i=1}^{n} (X - r_i)$ for all distinct $r_i \in \mathbb{Z}_p^*$, the following holds

$$\sum_{i=0}^{m} a_i U_i(X) \cdot \sum_{i=0}^{m} a_i V_i(X) - \sum_{i=0}^{m} a_i W_i(X) = H(X)T(X)$$

for some degree $(n-2)$ polynomial $H(X) \in \mathbb{Z}_p[X]$. Then, the Groth16 NIZK argument is given by the following algorithms:

- **Setup.** The setup algorithm accepts the QAP $\mathscr{R}$ as input, sets the secret trapdoor $\tau \coloneqq (\alpha, \beta, \gamma, \delta, x)$ for $\alpha, \beta, \gamma, \delta, x \leftarrow\!\!\$\ \mathbb{Z}_p^*$, and outputs it along with the verifier key vk and the prover key pk where

$$\mathsf{vk} \coloneqq \left(\ [\alpha]_1, \ [\beta]_2, \ [\gamma]_2, \ [\delta]_2, \left\{ [\chi_i]_1 \coloneqq \left[\tfrac{\beta U_i(x) + \alpha V_i(x) + W_i(x)}{\gamma} \right]_1 \right\}_{i=0}^{\ell} \ \right) \ ,$$

$$\mathsf{pk} \coloneqq \left(\begin{array}{c} [\alpha]_1, \ [\beta]_1, \ [\beta]_2, \ [\delta]_1, \ [\delta]_2, \ \left\{ [\theta_j]_1 \coloneqq \left[\tfrac{x^j T(x)}{\delta} \right]_1 \right\}_{j=0}^{n-2} \\ \{[\psi_i]_1 \coloneqq [U_i(x)]_1\}_{i=0}^{m}, \ \{[\varphi_i]_2 \coloneqq [V_i(x)]_2\}_{i=0}^{m}, \\ \left\{ [\zeta_i]_1 \coloneqq \left[\tfrac{\beta U_i(x) + \alpha V_i(x) + W_i(x)}{\delta} \right]_1 \right\}_{i=\ell+1}^{m} \end{array} \right) \ .$$

- **Prove.** The proving algorithm samples $r, s \leftarrow \mathbb{Z}_p^*$ and, using the instance and witness $(a_1, \ldots, a_\ell, a_{\ell+1}, \ldots, a_m) \in \mathbb{Z}_p^m$, sets the polynomial

$$H(X) \coloneqq \frac{\sum_{i=0}^{m} a_i U_i(X) \cdot \sum_{i=0}^{m} a_i V_i(X) - \sum_{i=0}^{m} a_i W_i(X)}{T(X)} \ .$$

  It then computes

$$\pi \coloneqq \left(\begin{array}{c} [A]_1 \coloneqq [\alpha]_1 + r[\delta]_1 + \sum_{i=0}^{m} a_i [\psi_i]_1, \\ [B]_2 \coloneqq [\beta]_2 + s[\delta]_2 + \sum_{i=0}^{m} a_i [\varphi_i]_2, \\ [C]_1 \coloneqq \left(s[\alpha]_1 + r[\beta]_1 + rs[\delta]_1 + \sum_{i=\ell+1}^{m} a_i [\zeta_i]_1 + \sum_{j=0}^{n-2} h_j [\theta_j]_1 \right) \end{array} \right) \tag{1}$$

  given the pk, and outputs π as the proof.

- **Verify.** Given the vk, the instance $(a_1, \ldots, a_\ell) \in \mathbb{Z}_p^\ell$ and a proof $\pi \coloneqq ([A]_1, [B]_2, [C]_1)$, the verifier simply checks whether:

$$[A]_1 \cdot [B]_2 \stackrel{?}{=} [\alpha]_1 \cdot [\beta]_2 + [C]_1 \cdot [\delta]_2 + \left(\sum_{i=0}^{\ell} a_i [\chi_i]_1 \right) \cdot [\gamma]_2$$

  and outputs the result.

3 Zero-knowledge Authenticators with Policy-privacy

We now formalize the notion of a zero-knowledge authenticator for a family of authentication policies and construct such an authenticator under a mild assumption that the underlying NIZK has equivocable (verification) keys, a property which we also define.

3.1 Zero-knowledge authenticator

We define a zero-knowledge authenticator for a family of authentication policies. As mentioned in the introduction, a zkAt can easily capture low-level semantics of an authentication mechanism as a (polynomial-size) circuit.

Let $\Pi = \{f_\lambda : \{0,1\}^{\mathsf{poly}(\lambda)} \to \{0,1\}\}$ be a family of authentication policies, then a *zero-knowledge authenticator* for any policy $\mathsf{P} \in \Pi$ over the message space $\mathcal{M}$ and private input space Ω consists of the following polynomial time algorithms:

- $\underline{\mathsf{Setup}(1^\lambda, \mathsf{P}) \to (\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P})}$. The setup algorithm takes as input the security parameter λ and the authentication policy P. It outputs a public verification key vk_P and a secret proving key[3] pk_P.
- $\underline{\mathsf{AuthProve}(\mathsf{pk}_\mathsf{P}, M, \omega) \to \pi/\perp}$. The proving algorithm takes as input the secret key pk_P, a message $M \in \mathcal{M}$ to be signed and some private input $\omega \in \Omega$. It outputs a proof π or $\perp$.
- $\underline{\mathsf{AuthVfy}(\mathsf{vk}_\mathsf{P}, M, \pi) \to \{0,1\}}$. The verification algorithm takes as input the public verification key vk_P, a message $M \in \mathcal{M}$, and a proof π. It outputs 0 or 1.

▶ **Definition 3.1** (Zero-knowledge Authenticator). *A zero-knowledge authenticator must satisfy the following properties with respect to any policy* $\mathsf{P} \in \Pi$*:*

- **Completeness:** *For every message* $M \in \mathcal{M}$ *and for every string* $\omega \in \Omega$ *such that* $\mathsf{P}(M||\omega) = 1$*,*

$$\Pr\left[\ \mathsf{AuthVfy}(\mathsf{vk}_\mathsf{P}, M, \pi) = 1 \ \ : \ \begin{array}{c}(\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{P}) \\ \pi \leftarrow\!\$\ \mathsf{AuthProve}(\mathsf{pk}_\mathsf{P}, M, \omega)\end{array}\ \right] = 1$$

- **Knowledge soundness***: For every* PPT *adversary* $\mathcal{A}$ *there exists a* PPT *extractor* $\mathcal{E}$ *and a negligible function* $\epsilon(\cdot)$ *satisfying, for all* $\lambda \in \mathbb{N}$ *the following probability is at most* $\epsilon(\lambda)$

$$\Pr\left[\ \begin{array}{l}\mathsf{AuthVfy}(\mathsf{vk}_\mathsf{P}, M^*, \pi^*) = 1 \\ \wedge\ \mathsf{P}(M^*||\omega^*) \neq 1\end{array} \ : \ \begin{array}{r}(\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{P}) \\ ((M^*, \pi^*);\ \omega^*) \leftarrow (\mathcal{A}||\mathcal{E})(\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P})\end{array}\ \right]$$

- **(Perfect) zero-knowledge***: There exists a* PPT *simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that for every* PPT *adversary* $\mathcal{A}$*, every* $\lambda \in \mathbb{N}$*, every* $M \in \mathcal{M}$ *and every string* $\omega \in \Omega$ *such that* $\mathsf{P}(M||\omega) = 1$*,*

$$\Pr\left[\ \begin{array}{l}\mathcal{A}(\mathsf{vk}_\mathsf{P}, M, \pi) = 1 \ \ : \\ \qquad (\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P}) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{P}) \\ \qquad \pi \leftarrow\!\$\ \mathsf{AuthProve}(\mathsf{pk}_\mathsf{P}, M, \omega)\end{array}\ \right] =$$

$$\Pr\left[\ \mathcal{A}(\mathsf{vk}_\mathsf{P}, M, \pi) = 1 \ \ : \ \begin{array}{r}(\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P}, \tau) \leftarrow \mathcal{S}_1(1^\lambda, \mathsf{P}) \\ \pi \leftarrow \mathcal{S}_2(\tau, M)\end{array}\ \right]$$

[3] We can assume that pk_P also implicitly contains P.

- **Policy privacy**: *For every stateful PPT adversary $\mathcal{A}$, there is a negligible function $\epsilon(\cdot)$ such that the following probability is at most $1/2 + \epsilon(\lambda)$*

$$\Pr\left[\begin{array}{l} \forall \hat{b} \in \{0,1\} \ : \ \mathsf{P}_{\hat{b}}(M^* || \omega^*_{\hat{b}}) = 1 \\ \wedge \ \mathcal{A}(\pi) = b \end{array} \ : \ \begin{array}{r} \{0,1\} \leftarrow\$ \ b \\ \mathsf{P}_0, \mathsf{P}_1 \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{vk}_{\mathsf{P}_b}, \mathsf{pk}_{\mathsf{P}_b}) \leftarrow\$ \ \mathsf{Setup}(1^\lambda, \mathsf{P}_b) \\ (M^*, \omega^*_0, \omega^*_1) \leftarrow \mathcal{A}(\mathsf{vk}_{\mathsf{P}_b}) \\ \pi \leftarrow\$ \ \mathsf{AuthProve}(\mathsf{pk}_{\mathsf{P}_b}, M^*, \omega^*_b) \end{array}\right]$$

Before giving a formal construction, we must define a new object called designated-prover NIZK schemes (DP-NIZK) with *equivocable verification keys*. At a high level, the property of verification-key equivocation guarantees that the verification key of a DP-NIZK scheme is independent of its circuit. Looking ahead to our construction, when instantiated with a DP-NIZK with this property, we will be able to reduce the policy privacy of our zkAt to the verification key equivocation of the DP-NIZK.

3.1.1 Verification-key equivocation

Informally, the vk-equivocation game models and adversary who, given a verification key and a proof for a statement in languages specified by both circuits of its choice, must decide which of the said circuits does the key (and proof) correspond to.

▶ **Definition 3.2** (vk-equivocation)**.** *A publicly verifiable DP-NIZK scheme has equivocable verification keys if for every stateful PPT adversary $\mathcal{A}$, there is a negligible function $\epsilon(\cdot)$ such that the following probability is at most $\frac{1}{2} + \epsilon(\lambda)$,*

$$\Pr\left[\begin{array}{l} \forall \hat{b} \in \{0,1\} \ : \ \mathsf{C}_{\hat{b}}(x^* || w^*_{\hat{b}}) = 1 \\ \wedge \ \mathcal{A}(\pi) = b \end{array} \ : \ \begin{array}{r} \{0,1\} \leftarrow\$ \ b \\ \mathsf{C}_0, \mathsf{C}_1 \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{vk}_b, \mathsf{pk}_b) \leftarrow\$ \ \mathsf{Setup}(1^\lambda, \mathsf{C}_b) \\ (x^*, w^*_0, w^*_1) \leftarrow \mathcal{A}(\mathsf{vk}_b) \\ \pi \leftarrow\$ \ \mathsf{Prove}(\mathsf{pk}_b, x^*, w^*_b) \end{array}\right]$$

where each circuit C_b encodes an NP relation $\mathscr{R}_b = \{(x,w) \ : \ \mathsf{C}_b(x||w) = 1\}$

3.2 Construction

We now provide our construction. It requires a publicly-verifiable DP-NIZK scheme *with vk-equivocation* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ for the relation $\mathscr{R}_\mathsf{P} = \{(x,w) : \mathsf{P}(x||w) = 1\}$.

- $\underline{\mathsf{Setup}(1^\lambda, \mathsf{P}) \to (\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P})}$. Give the security parameter λ and the policy P as input, the setup algorithm runs the NIZK setup to obtain the verification and the prover keys, i.e., $(\mathsf{vk}_{\mathsf{zk}}, \mathsf{pk}_{\mathsf{zk}}) \leftarrow\$ \ \mathsf{NIZK.Setup}(1^\lambda, \mathsf{P})$. It then outputs $\mathsf{vk}_\mathsf{P} := \mathsf{vk}_{\mathsf{zk}}$, and $\mathsf{pk}_\mathsf{P} := \mathsf{pk}_{\mathsf{zk}}$.
- $\underline{\mathsf{AuthProve}(\mathsf{pk}_\mathsf{P}, M, \omega) \to \pi/\perp}$. It parses pk_P to obtain $\mathsf{pk}_{\mathsf{zk}}$ and then computes the proof $\pi \leftarrow\$ \ \mathsf{NIZK.Prove}(\mathsf{pk}_{\mathsf{zk}}, x := M, w := \omega)$ and outputs it.
- $\underline{\mathsf{AuthVfy}(\mathsf{vk}_\mathsf{P}, M, \pi) \to \{0,1\}}$. It returns the output of $\mathsf{NIZK.Verify}(\mathsf{vk}_\mathsf{P}, M, \pi)$.

Security. We claim that the above construction is a zkAt. The knowledge soundness and zero-knowledge properties of zkAt follow directly from the underlying DP-NIZK so we only focus on policy privacy, which we claim follows when the DP-NIZK proof has equivocable verification keys. The formal theorem statement and proof are presented in the full version of this article [14].

4 An Equivocable Groth16

In Section 3.1.1 we defined proofs with equivocable verification keys that are required to instantiate our zkAt construction. We now explain how to concretely build this primitive from the DP-NIZK of Groth [29].

A crucial observation towards achieving policy-privacy is that the Groth16 verification key can be equivocated – in the sense that one can perform the Groth16 setup in a way that guarantees that vk can be sampled independently of the relation. We give a bootstrapping compiler to build an equivocable Groth16 scheme given the non-equivocable version.

Now, for some prime p such that $|p| = \lambda$, groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ such that the pairing $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map, consider a QAP,

$$\mathscr{R} = \left\{ p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, g_1, g_2, \ell, \{U_i(X), V_i(X), W_i(X)\}_{i=0}^{m}, T(X) \right\} \quad , \tag{2}$$

that defines a field $\mathbb{Z}_p$ and a language of statements $(a_1, \ldots, a_\ell) \in \mathbb{Z}_p^\ell$ and witnesses $(a_{\ell+1}, \ldots, a_m) \in \mathbb{Z}_p^{m-\ell}$ such that for $a_0 = 1$, polynomials $\{U_i(X), V_i(X), W_i(X)\}_{i=0}^{m}$ and $T(X)$ in $\mathbb{Z}_p[X]$ with $\deg(U_i), \deg(V_i), \deg(W_i) < \deg(T) = n$ (for all $i \in [0, m]$) and $T(X) \coloneqq \prod_{i=1}^{n} (X - r_i)$ for all distinct $r_i \in \mathbb{Z}_p^*$, the following holds

$$\sum_{i=0}^{m} a_i U_i(X) \cdot \sum_{i=0}^{m} a_i V_i(X) - \sum_{i=0}^{m} a_i W_i(X) = H(X)T(X)$$

for some degree $(n-2)$ polynomial $H(X) \in \mathbb{Z}_p[X]$. Then, we have the following constructive procedure:

1. Sample $x, \{y_{U,i}\}_{i=0}^{m}, \{y_{V,i}\}_{i=0}^{m}, \{y_{W,i}\}_{i=0}^{m} \leftarrow\!\$\, \mathbb{Z}_p^*$.
2. For every symbol $S \in \{U, V, W\}$ and for every $i \in [0, m]$ interpolate the polynomial $\tilde{S}_i(X) \in \mathbb{Z}_p[X]$ over coordinates

$$\tilde{S}_i(x) = y_{S,i} \quad \text{and} \quad \forall j \in [n] \;:\; \tilde{S}_i(r_j) = S_i(r_j) = S_{i,j} \quad .$$

Given this, we claim that the modified QAP:

$$\tilde{\mathscr{R}} = \left\{ p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathsf{e}, g_1, g_2, \ell, \left\{\tilde{U}_i(X), \tilde{V}_i(X), \tilde{W}_i(X)\right\}_{i=0}^{m}, T(X) \right\} \tag{3}$$

defines the same relation as $\mathscr{R}$. Formally,

▶ **Lemma 4.1.** *$T(X)$ divides $\sum_{i=0}^{m} a_i \tilde{U}_i(X) \cdot \sum_{i=0}^{m} a_i \tilde{V}_i(X) - \sum_{i=0}^{m} a_i \tilde{W}_i(X)$ if and only if $(a_1, \ldots, a_m)$ is a satisfying assignment for $\mathscr{R}$.*

Please refer to the full version of this article [14] for the proof of this lemma.

In short, by interpolating fresh polynomials $\left\{\tilde{U}_i, \tilde{V}_i, \tilde{W}_i\right\}_i$ that behave exactly as $\{U_i, V_i, W_i\}_i$ (respectively, for each i), but also evaluate at x to uniformly chosen values $\{y_{U,i}, y_{V,i}, y_{W,i}\}_i$ (respectively, for each i), we have essentially fixed the evaluation of the polynomials at x independently of the relation. The result is that the verification key generated during setup will now be made completely independent of the updated QAP which, as we have just shown, defines the same relation as the original QAP. Accordingly, let us now define the modified setup algorithm (prove and verify algorithms do not change):

- **Setup.** The setup algorithm accepts a QAP $\mathscr{R}$ as explained by (2), sets the secret trapdoor $\tau \coloneqq (\alpha, \beta, \gamma, \delta, x, \{y_{U,i}\}_{i=0}^{m}, \{y_{V,i}\}_{i=0}^{m}, \{y_{W,i}\}_{i=0}^{m})$ for all values in τ sampled uniformly from $\mathbb{Z}_p^*$, and outputs it along with the verifier key vk and the prover key $\left(\mathsf{pk}, \tilde{\mathscr{R}}\right)$ where $\tilde{\mathscr{R}}$ is of the form described in (3) and,

$$\mathsf{vk} := \left([\alpha]_1,\ [\beta]_2,\ [\gamma]_2,\ [\delta]_2, \left\{ [\chi_i]_1 := \left[\frac{\beta y_{U,i} + \alpha y_{V,i} + y_{W,i}}{\gamma} \right]_1 \right\}_{i=0}^{\ell} \right) ,$$

$$\mathsf{pk} := \left(\begin{array}{c} [\alpha]_1,\ [\beta]_1,\ [\beta]_2,\ [\delta]_1,\ [\delta]_2,\ \left\{ [\theta_j]_1 := \left[\frac{x^j T(x)}{\delta} \right]_1 \right\}_{j=0}^{n-2} \\ \left\{ [\psi_i]_1 := [y_{U,i}]_1 \right\}_{i=0}^{m},\ \left\{ [\varphi_i]_2 := [y_{V,i}]_2 \right\}_{i=0}^{m}, \\ \left\{ [\zeta_i]_1 := \left[\frac{\beta y_{U,i} + \alpha y_{V,i} + y_{W,i}}{\delta} \right]_1 \right\}_{i=\ell+1}^{m} \end{array} \right) .$$

Security. The main theorem for this construction states that it is a NIZK proof system with vk-equivocation. In addition, completeness and zero-knowledge follow straightforwardly, so we only show that this construction is also knowledge-sound. This is described and proven formally in the full version of this article [14].

5 Obliviously Updateable Policy-privacy

In this section, we introduce zkAt with *oblivious updates* wherein, given a pk_P with respect to an existing policy $\mathsf{P} \in \Pi$, a policy issuer can update pk_P to pk'_P for some new policy $\mathsf{P}' \in \Pi$ without updating the corresponding vk_P. Following the formal definition, we give a generic construction for zkAt using recursive NIZKs.

An *obliviously updateable* zkAt, that we call zkAt$^+$, additionally consists of the following polynomial time algorithms:

- $\underline{\mathsf{Gen}(1^\lambda) \to \mathsf{pp}}$. The parameter generation algorithm as input the security parameter λ, and outputs public parameters pp for the protocol. All algorithms of zkAt$^+$ take pp as input, but we omit it for brevity. This algorithm is run once and for all.
- $\underline{\mathsf{Setup}(1^\lambda, \mathsf{P}) \to (\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P}, \kappa)}$. The setup algorithm takes as input the security parameter λ and the authentication policy P. It outputs a public verification key vk_P, a secret proving key pk_P and a secret update key κ.
- $\underline{\mathsf{PolUpdate}(\mathsf{pk}_\mathsf{P}, \kappa, \mathsf{P}') \to \mathsf{pk}'_\mathsf{P}/\perp}$. The policy update algorithm takes as input the secret proving key pk_P, a secret update key κ, and an updated policy P'. It outputs the updated secret proving key $\mathsf{pk}_{\mathsf{P}'}$.

▶ **Definition 5.1** (Obliviously updateable Policy-private Zero-knowledge Authenticator)**.** *A zkAt$^+$ must additionally satisfy the following properties for policies $\mathsf{P}, \mathsf{P}' \in \Pi$,*

- **Update completeness:** *For every message $M \in \mathcal{M}$ and for every string $\omega \in \Omega$ such that $\mathsf{P}'(M||\omega) = 1$, we must have that*

$$\Pr\left[\mathsf{AuthVfy}(\mathsf{vk}_\mathsf{P}, M, \pi) = 1 \ : \ \begin{array}{r} \mathsf{pp} \leftarrow\!\$\ \mathsf{Gen}(1^\lambda) \\ (\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P}, \kappa) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{P}) \\ \mathsf{pk}'_\mathsf{P} \leftarrow\!\$\ \mathsf{PolUpdate}(\mathsf{pk}_\mathsf{P}, \mathsf{P}') \\ \pi \leftarrow\!\$\ \mathsf{AuthProve}(\mathsf{pk}'_\mathsf{P}, M, \omega) \end{array} \right] = 1$$

- **Update knowledge soundness***: For every PPT adversary $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}$ and a negligible function $\epsilon(\cdot)$, for all $\lambda \in \mathbb{N}$ such that the following probability is at most $\epsilon(\lambda)$,*

$$\Pr\left[\begin{array}{l} \mathsf{AuthVfy}(\mathsf{vk}_\mathsf{P}, M^*, \pi^*) = 1 \ \wedge \\ \forall \mathsf{P}' \in Q_\mathsf{P} \ : \ \mathsf{P}'(M^*||\omega^*) \neq 1 \end{array} \ : \ \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{P} \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P}, \kappa) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, \mathsf{P}) \\ ((M^*, \pi^*);\ \omega^*) \leftarrow (\mathcal{A}^{\mathcal{O}_\kappa}||\mathcal{E})(\mathsf{pp}) \end{array} \right]$$

where the oracle $\mathcal{O}_\kappa(\cdot)$ *takes as input an updated policy* $\mathsf{P}^{(i)} \in \Pi$ *in its* i^{th}*-query. It adds* $\mathsf{P}^{(i)}$ *to the set* Q_P *(initially set to* $\{\mathsf{P}\}$*) and outputs the signing key* $\mathsf{pk}_\mathsf{P}^{(i)}$ *under* $\mathsf{P}^{(i)}$ *by running* $\mathsf{PolUpdate}(\mathsf{pk}_\mathsf{P}, \kappa, \mathsf{P}^{(i)})$.

▶ **Remark 5.2.** Since the verification key vk_P is independent of the policy updates, the view of the policy privacy adversary remains unaltered from that in Definition 3.1. So, policy privacy is actually implicit in the definition.

5.1 Construction

In this section, we describe our generic zkAt$^+$ construction for *disjunctive* policy updates, which we explain next.

Disjunctive policy updates. Let $\mathsf{P} \in \Pi$ be an existing policy. Then, an update $\mathsf{P}' \in \Pi$ is a disjunctive update if and only if P' is a disjunction of P with some other valid policy. Thus, for each $\mathsf{P} \in \Pi$, we can define the predicate $\mathsf{Adm}_\mathsf{P}(\mathsf{P}') = 1 \Leftrightarrow \mathsf{P}' \in \{\mathsf{P} \vee f \ : \ \forall f \in \Pi\}$.

Barring the trivial idea of re-running the setup and distributing fresh keys under the new policy, obliviously performing *general* policy updates, which is to say non-disjunctive updates, appears to be somewhat challenging. This is because it would require a way to revoke a proving key under an older policy without also revoking the corresponding verification key. Nevertheless, as we explain show in the full version of this article [14], allowing the older proving key to expire via a simple tagging mechanism suffices for general policy updates with only minor generic modification to the overall construction.

At a high level, our zkAt$^+$ construction, recursively composes NIZK proofs with the "inner" proof corresponding to the policy predicate, and the "outer" proof to the *knowledge* of a valid proof to the policy predicate as well as a signature on the inner verification key for soundness. Thus a verifier simply verifies this outer proof and is convinced of the user's authenticity. Like in our Groth16-zkAt construction, our zkAt$^+$ construction also requires maintaining a sensitive secret κ that gets used whenever the policies need to be updated, so that when a new policy is created, the issuer simply issues a fresh signature on the new inner verification key (using κ). Importantly, no change is made to the outer keys so that all third parties remain unaware of the update.

Tools required. The construction below utilizes a signature scheme $\mathsf{Sig} = (\mathsf{Sig.Setup}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$, an inner NIZKAoK scheme $\mathsf{NIZK}_I = (\mathsf{NIZK}_I.\mathsf{Setup}, \mathsf{NIZK}_I.\mathsf{Prove}, \mathsf{NIZK}_I.\mathsf{Verify})$ which encodes the policy, and an outer NIZKAoK scheme, $\mathsf{NIZK}_O = (\mathsf{NIZK}_O.\mathsf{Setup}, \mathsf{NIZK}_O.\mathsf{Prove}, \mathsf{NIZK}_O.\mathsf{Verify})$ for the following relation

$$\mathscr{R}_O = \left\{ \begin{array}{c} x \coloneqq (\mathsf{vk}_\sigma, M) \\ w \coloneqq (\mathsf{crs}_I, \pi_I, \sigma) \end{array} \ : \ \begin{array}{c} \mathsf{NIZK}_I.\mathsf{Verify}(\mathsf{crs}_I, M, \pi_I) = 1 \\ \wedge\ \mathsf{Sig.Verify}(\mathsf{vk}_\sigma, \mathsf{crs}_I, \sigma) = 1 \end{array} \right\}$$

- $\underline{\mathsf{Gen}(1^\lambda) \to \mathsf{pp}}$. Given the security parameter λ as input, the parameter generation algorithm generates the NIZK crs as $\mathsf{crs}_O \leftarrow\!\$\ \mathsf{NIZK}_O.\mathsf{Setup}(1^\lambda, \mathsf{C}_O)$, where C_O is the circuit encoding $\mathscr{R}_O$, and outputs $\mathsf{pp} \coloneqq \mathsf{crs}_O$. This algorithm is run once and for all.
- $\underline{\mathsf{Setup}(1^\lambda, \mathsf{P}) \to (\mathsf{vk}_\mathsf{P}, \mathsf{pk}_\mathsf{P}, \kappa)}$. Given the security parameter λ and the policy P as input, the setup algorithm runs the inner NIZK setup to obtain $\mathsf{crs}_I \leftarrow\!\$\ \mathsf{NIZK}_I.\mathsf{Setup}(1^\lambda, \mathsf{P})$. Next, it creates the signing and verification keys for the signature scheme as $(\mathsf{vk}_\sigma, \mathsf{sk}_\sigma) \leftarrow\!\$\ \mathsf{Sig.Setup}(1^\lambda)$, and then signs crs_I to obtain $\sigma \leftarrow\!\$\ \mathsf{Sig.Sign}(\mathsf{sk}_\sigma, \mathsf{crs}_I)$. Finally, it outputs $\mathsf{vk}_\mathsf{P} \coloneqq \mathsf{vk}_\sigma$, $\mathsf{pk}_\mathsf{P} \coloneqq (\mathsf{crs}_I, \mathsf{vk}_\sigma, \sigma)$, and $\kappa \coloneqq \mathsf{sk}_\sigma$.

- $\underline{\mathsf{AuthProve}(\mathsf{pk_P}, M, \omega) \to \pi/\perp}$. It first parses $\mathsf{pk_P}$ as $(\mathsf{crs}_I, \mathsf{vk}_\sigma, \sigma)$ and continues only if $\mathsf{Sig.Verify}(\mathsf{vk}_\sigma, \mathsf{crs}_I, \sigma) = 1$, otherwise it aborts and outputs $\perp$. It then computes the proof $\pi_I \leftarrow\$ \mathsf{NIZK}_I.\mathsf{Prove}(\mathsf{crs}_I, x := M, w := \omega)$ and then outputs the final proof π computed as $\pi \leftarrow\$ \mathsf{NIZK}_O.\mathsf{Prove}(\mathsf{pp}, x := (\mathsf{vk}_\sigma, M), w := (\mathsf{crs}_I, \pi_I, \sigma))$.
- $\underline{\mathsf{AuthVfy}(\mathsf{vk_P}, M, \pi) \to \{0,1\}}$. Returns the output of $\mathsf{NIZK}_O.\mathsf{Verify}(\mathsf{pp}, x := (\mathsf{vk_P}, M), \pi)$.
- $\underline{\mathsf{PolUpdate}(\mathsf{pk_P}, \kappa, \mathsf{P}') \to \mathsf{pk}'_\mathsf{P}/\perp}$. If $\mathsf{Adm_P}(\mathsf{P}') \neq 1$, it outputs $\perp$ and aborts. Otherwise, it parses $\mathsf{pk_P}$ as $(\mathsf{crs}_I, \mathsf{vk}_\sigma, \sigma)$. Then, the update algorithm re-runs the inner NIZK setup with this input to obtain $\mathsf{crs}'_I \leftarrow\$ \mathsf{NIZK}_I.\mathsf{Setup}(1^\lambda, \mathsf{P}')$. Next, it signs crs'_I to obtain $\sigma' \leftarrow\$ \mathsf{Sig.Sign}(\kappa, \mathsf{crs}'_I)$. Finally, it outputs $\mathsf{pk}'_\mathsf{P} := (\mathsf{crs}'_I, \mathsf{vk}_\sigma, \sigma')$.

▶ **Remark 5.3.** Depending on the application scenario, the setup could be combined with the parameter generation algorithm so that both are performed once, and the rest of the protocol proceeds identically. This could be potentially useful in a situation where, for instance, an organization has an authorized list of users who can create transactions on behalf of the organization, while keeping their individual identities private. Moreover, this gives a *maximally* private authentication scheme while still demanding accountability from the users. On the other hand, if a protocol requires identities from individual users, one can perform the setup for each user and set the (hash of) vk_σ as their corresponding addresses.

Security. The main security theorem and the corresponding proof for this section are provided in the full version of this article [14].

6 Application: Private On-chain Authentication

We now discuss how our zkAt construction can be integrated into a blockchain. First, recall that zkAt$^+$ uses existing NIZKs in a black-box manner, so its instantiation is relatively straightforward. Even our concrete Groth16-zkAt uses the standard Groth16 proving and verification algorithms. Therefore, the only major requirement for integrating our zkAt constructions is the support for on-chain NIZK verification. Fortunately, many smart-contract supporting chains like Ethereum, Aptos and Sui already support on-chain NIZK verification for Groth16 (among others), and can thus readily integrate any of our constructions. Therefore, zkAt can act as a drop-in replacement for any existing authenticator used on public blockchains including multi-signature, threshold signatures and smart-contract based authenticators.

Next, we give a practical instantiation of our zkAt. As a concrete example we will consider the policy – "*require t-out-of-n signatures*," where t, n and the n signature verification keys are all to be hidden with the zkAt. The concrete zkAt would look as follows:

- **Setup.** Create commitments $c_1, c_2, \ldots, c_n$ to the n account addresses, i.e., $c_i \leftarrow \mathsf{Commit}(\mathsf{addr}_i; r_i)$ where address addr_i is a signature verification key for the i^{th} signer. Then, create a Merkle tree digest of the n commitments, $\mathsf{root} \leftarrow \mathsf{MT.Commit}(\{c_1, \ldots, c_n\})$. Run the Groth16-zkAt setup to create a (designated-prover) NIZK proving key and (proof) verification key pair $(\mathsf{pk_P}, \mathsf{vk_P}) \leftarrow\$ \mathsf{Setup}(1^\lambda, \mathsf{P})$ for the policy P given by the relation,

$$\left\{ \begin{array}{l} x := (\mathsf{root}, \mathsf{tx_{pb}}), \\ w := \left(\{\mathsf{addr}_i, r_i, c_i, \sigma_i\}_{i=1}^{t}, \mathsf{tx_{pv}}\right) \end{array} : \begin{array}{r} \forall i \neq k \in [t]\ ,\ \mathsf{addr}_i \neq \mathsf{addr}_k \\ \wedge\ c_i = \mathsf{Commit}(\mathsf{addr}_i;\ r_i) \\ \wedge\ \mathsf{Sig.Verify}(\mathsf{addr}_i, \mathsf{tx_{pb}} || \mathsf{tx_{pv}}, \sigma_i) = 1 \\ \wedge\ \mathsf{MT.Includes}(\mathsf{root}, c_i) = 1 \end{array} \right\},$$

  where $\mathsf{tx_{pb}}$ (resp. $\mathsf{tx_{pv}}$) is the public (resp. private) part of the transaction. The proof verification key vk acts as the user's on-chain address.

- **Sign.** Split the transaction into public and private components, as determined by the privacy requirements. Collect signatures $\{\sigma_1, \sigma_2, \ldots, \sigma_t\}$ on the full transaction $\mathsf{tx_{pb}}||\mathsf{tx_{pv}}$ from t accounts $\{i_1, i_2, \ldots, i_t\}$. Finally, generate the authentication proof with public input $M := (\mathsf{root}, \mathsf{tx_{pb}})$ and private input $\omega := \left(\{\mathsf{addr}_i, r_i, c_i, \sigma_i\}_{i=1}^{t}, \mathsf{tx_{pv}}\right)$, i.e., $\pi \leftarrow\$ \mathsf{AuthProve}(\mathsf{pk_P}, M, \omega)$ proving that the user has t valid signatures on the transaction from the set of n signers committed in the Merkle tree.

In particular, this design only requires implementing t signature verifications in the circuit. Assuming the use of a NIZK-friendly signature scheme which only requires a few thousand R1CS constraints per verification, this is concretely efficient (cf. § 7). Using standard signature schemes (which might be necessary if we want to use existing accounts) can lead to costlier circuits. Concretely, each secp256k1 verification requires 1.5M constraints [1] and Ed25519 verification requires 2.5M constraints [23]. However, proving only takes a few seconds on cloud machines[4], so these are still practical for reasonable t values seen in practice.

▶ **Remark 6.1.** Successfully integrating zkAt into a blockchain, will also require some standardization effort with regards to the zkAt's public inputs so as not to unintentionally leak some information about the policy by virtue of using any specific public input. Below, we provide a brief list of the required public inputs to a zkAt. Note that a blockchain designer may decide to support all or a subset of these.

- **Transaction details ($\mathsf{tx_{pb}}$):**
  - **Digest:** specifying just a transaction digest however requires parsing the transaction within a zero-knowledge proof. Transaction parsing can be simplified by carefully designing the format of a transaction, e.g., structure it in the format of a Merkle tree. We leave concrete specification for future work.
  - **Amount:** if supporting a specific type of transaction is enough, e.g., amount transfers, then exposing the transaction amount as a public input can make the signature generation process very efficient.
  - **Other fields:** similarly, other common fields of a transaction, e.g., the sender address, can be exposed as public inputs.
- **Time:** most blockchains support some notion of time. Including time allows specifying time-based policies, e.g., use a certain set of credentials before market close and another after. If the trapdoor is persisted in a safe place (e.g., cold storage), then time allows oblivious policy updates. Say we want to rotate keys once a month, then we can embed an expiry date after a month, and use the trapdoor to generate a new policy when needed.
- **Support for web2 credentials:** certain blockchains support authentication based on existing credentials issued by web2 providers, e.g., e-Passports [31], OpenID Connect credentials [4]. To support these, the chains use oracles to fetch the public keys of an issuer. In this case, a Merkle root of all the authorized public keys can serve as a public input.

7 Implementation and Evaluation

We implemented, both, our zkAt construction by instantiating with standard Groth16 (as a proxy for the Groth16-zkAt), as well as zkAt$^+$ constructions in Go using the `gnark` library [11][5] (we chose this library since it supports both Groth16 and recursive composition). All our experiments were done on a laptop equipped with an Apple M3 Pro chip and we report means over 100 executions.

[4] It only takes 6s to verify Ed25519 signatures on a 16-core 32G RAM machine [23].
[5] GitHub: `https://github.com/Consensys/gnark`

Table 2 Comparison of zkAt with threshold signatures. Signer time is equivalently the prover time in zkAt.

	Signer time (ms)	Verifier time (ms)
zkAt for P	50.97	0.89
2-of-3 threshold	0.03	0.07

Policy choice. We restrict our implementations to the policy P described abstractly in the previous section as: "*require* 1-*out-of*-3 *signatures for transaction amounts under* 1000 *units, otherwise require* 2-*out-of*-3". We find that this sufficiently captures complex policy semantics not offered by, for instance, a traditional (t-out-of-n) threshold scheme while also giving a reasonable basis for comparison between the two.

We remark that our scheme generalizes to arbitrary circuits, and for zkAt, the proving time is no worse than a standard zk-SNARK which is very commonly used in practice so we expect an identical scalability profile since the prove and verify algorithms do not change. Our intention for comparing with a 2-of-3 scheme is simply to demonstrate that the computational cost for additional privacy is nominal. Of course, one can formulate more complex policies with greater number of constraints, but we note here that even as the prover time increases for more complex policies, the proof size and the verification time remain the same due to the use of zk-SNARK. Moreover, since the proof computation would be performed offline, and the (online) verification time is reasonably efficient, and depends essentially only on the size of the public input, we do not expect a significant bottleneck to the scalability of our protocol.

7.1 Evaluation of zkAt

In our proof-of-concept implementation a prover first draws an integer valued transaction amounts uniformly, creates the required number of EdDSA signatures [7] over a ZK-friendly curve and then constructs the zkAt proof with the transaction data as the public input and the signatures and their corresponding verification keys as the private input. In short, the NIZK verification circuit checks whether one or two of the signatures verify depending on the transaction amount.

Concretely, the basic zkAt was implemented over the BN254 curve and the resulting R1CS had 24,564 constraints. Table 2 compares our implementation against a 2-of-3 threshold scheme. While the signer (prover) time for zkAt is noticeably higher relative to the threshold scheme, it is still small in absolute terms. The verification times are within an order of magnitude.

7.2 Evaluation of zkAt$^+$

As before, in our simple proof-of-concept implementation a prover first draws an integer valued transaction amounts uniformly, creates the required number of EdDSA signatures [7]. It then constructs the inner proof that the necessary number of verifying signatures were obtained for the for the given transaction amount. Next, using the transaction data and its signature verification key (which can be thought of as the prover's address) as the public input, it constructs the outer proof that (i) the inner NIZK circuit accepts and; (ii) it has a verifying signature (with respect to the signature verification key corresponding to its address) on the proving key used to compute the inner proof. In our implementation,

we instantiate the proof system with the Groth16 due to its compact proof size as well as support for its recursive composition inside `gnark`, although other choices of proof systems such as [25, 16, 15] are equally valid.

We implement the zkAt$^+$ over the SNARK-friendly 2-chain[6] of BLS12-377 inner curve [5, 12] and BW6-761 [13, 21] outer curve which was shown to be highly efficient for Groth16 [21, 22]. The inner and outer R1CSs have 24,172 and 40,474 constraints respectively. Table 3 gives the prover and verifier times for our implementation.

Table 3 Prover and verifier times for zkAt$^+$. The preprocessing time is given for each signer.

Inner prover time (ms)	**Preprocessing**	325.15
	Proof generation	78.55
Outer prover time (ms)		644.54
Verifier time (ms)		7.47

References

1 0xPARC. Big integer arithmetic and secp256k1 ecc operations in circom. `https://github.com/0xPARC/circom-ecdsa`, 2024. Accessed: 2025-02-19.

2 Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ECDSA via pseudorandom correlation generators. In *2022 IEEE Symposium on Security and Privacy*, pages 2554–2572, San Francisco, CA, USA, May 22–26 2022. IEEE Computer Society Press. `doi:10.1109/SP46214.2022.9833559`.

3 Argent. Smart wallet features. `https://www.argent.xyz/blog/smart-wallet-features`, 2024. Accessed: 2024-10-11.

4 Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Yan Ji, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Mahdi Sedaghat, and Joy Wang. zkLogin: Privacy-preserving blockchain authentication with existing credentials. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024: 31st Conference on Computer and Communications Security*, pages 3182–3196, Salt Lake City, UT, USA, October 14–18 2024. ACM Press. `doi:10.1145/3658644.3690356`.

5 Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02: 3rd International Conference on Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267, Amalfi, Italy, September 12–13 2003. Springer Berlin Heidelberg, Germany. `doi:10.1007/3-540-36413-7_19`.

6 Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 517–550, Santa Barbara, CA, USA, August 15–18 2022. Springer, Cham, Switzerland. `doi:10.1007/978-3-031-15985-5_18`.

7 Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, September 2012. `doi:10.1007/s13389-012-0027-1`.

8 BitGo. Policy builder overview. `https://developers.bitgo.com/guides/policy-builder/overview`, 2024. Accessed: 2024-10-11.

[6] A 2-chain is a pair of pairing-friendly elliptic curves such that the base field of one curve is equal to the scalar field of the other. This enables efficient proof composition of up to one level [12].

9 Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464, Brisbane, Queensland, Australia, December 2–6 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-030-03329-3_15.

10 Dan Boneh, Wilson Nguyen, and Alex Ozdemir. Efficient functional commitments: How to commit to private functions. Cryptology ePrint Archive, Report 2021/1342, 2021. URL: https://eprint.iacr.org/2021/1342.

11 Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. Consensys/gnark: v0.9.0, February 2023. doi:10.5281/zenodo.5819104.

12 Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy*, pages 947–964, San Francisco, CA, USA, May 18–21 2020. IEEE Computer Society Press. doi:10.1109/SP40000.2020.00050.

13 Friederike Brezing and Annegret Weng. Elliptic curves suitable for pairing based cryptography. *Designs, Codes and Cryptography*, 37(1):133–141, 2005. doi:10.1007/s10623-004-3808-4.

14 Kostas Kryptos Chalkias, Deepak Maram, Arnab Roy, Joy Wang, and Aayush Yadav. Zero-knowledge authenticator for blockchain: Policy-private and obliviously updateable. Cryptology ePrint Archive, Paper 2025/921, 2025. URL: https://eprint.iacr.org/2025/921.

15 Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530, Lyon, France, April 23–27 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-30617-4_17.

16 Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768, Zagreb, Croatia, May 10–14 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-45721-1_26.

17 Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127, Santa Barbara, CA, USA, August 16–20 1988. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-48184-2_8.

18 Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Santa Barbara, CA, USA, August 20–24 1990. Springer, New York, USA. doi:10.1007/0-387-34805-0_28.

19 Jiajun Du, Zhonghui Ge, Yu Long, Zhen Liu, Shifeng Sun, Xian Xu, and Dawu Gu. MixCT: Mixing confidential transactions from homomorphic commitment. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022: 27th European Symposium on Research in Computer Security, Part III*, volume 13556 of *Lecture Notes in Computer Science*, pages 763–769, Copenhagen, Denmark, September 26–30 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-17143-7_39.

20 Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy*, pages 106–123, San Francisco, CA, USA, May 19–23 2019. IEEE Computer Society Press. doi:10.1109/SP.2019.00020.

21 Youssef El Housni and Aurore Guillevic. Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20: 19th International Conference on Cryptology and Network Security*, volume 12579 of *Lecture Notes in Computer Science*, pages 259–279, Vienna, Austria, December 14–16 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-65411-5_13.

22 Youssef El Housni and Aurore Guillevic. Families of SNARK-friendly 2-chains of elliptic curves. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 367–396, Trondheim, Norway, May 30 – June 3 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-07085-3_13.

23 Electron Labs. Ed25519 implementation in circom. https://github.com/Electron-Labs/ed25519-circom, 2024. Accessed: 2024-10-11.

24 Fireblocks. Fireblocks governance and policy engine. https://www.fireblocks.com/platforms/governance-and-policy-engine/, 2024. Accessed: 2024-10-11.

25 Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. URL: https://eprint.iacr.org/2019/953.

26 Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645, Athens, Greece, May 26–30 2013. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-38348-9_37.

27 Noemi Glaeser, Matteo Maffei, Giulio Malavolta, Pedro Moreno-Sanchez, Erkan Tairi, and Sri Aravinda Krishnan Thyagarajan. Foundations of coin mixing services. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 1259–1273, Los Angeles, CA, USA, November 7–11 2022. ACM Press. doi:10.1145/3548606.3560637.

28 Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 473–489, Dallas, TX, USA, October 31 – November 2 2017. ACM Press. doi:10.1145/3133956.3134093.

29 Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12 2016. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_11.

30 Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. TumbleBit: An untrusted bitcoin-compatible anonymous payment hub. In *ISOC Network and Distributed System Security Symposium – NDSS 2017*, San Diego, CA, USA, February 26 – March 1 2017. The Internet Society. doi:10.14722/ndss.2017.23086.

31 International Civil Aviation Organization. Basics of epassport cryptography, 2024. Accessed: 2025-02-17. URL: https://www.icao.int/Security/FAL/PKD/BVRT/Pages/Basics.aspx.

32 K. Itakura. A public-key cryptosystem suitable for digital multisignatures, 1983.

33 Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020: 27th Annual International Workshop on Selected Areas in Cryptography*, volume 12804 of *Lecture Notes in Computer Science*, pages 34–65, Halifax, NS, Canada (Virtual Event), October 21-23 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-81652-0_2.

34 Kostas Kryptos Chalkias. Soft privacy-related leak in threshold eddsa wallets. https://x.com/kostascrypto/status/1703594584100700641, 2023.

35 Jin Li, Man Ho Au, Willy Susilo, Dongqing Xie, and Kui Ren. Attribute-based signature and its applications. In Dengguo Feng, David A. Basin, and Peng Liu, editors, *ASIACCS 10: 5th ACM Symposium on Information, Computer and Communications Security*, pages 60–69, Beijing, China, April 13–16 2010. ACM Press. doi:10.1145/1755688.1755697.

36 Benoît Libert, Somindu C. Ramanna, and Moti Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016: 43rd International Colloquium on Automata, Languages and Programming*,

volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:14, Rome, Italy, July 11–15 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2016.30.

37 Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. Attribute-based signatures. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 376–392, San Francisco, CA, USA, February 14–18 2011. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-19074-2_24.

38 Easwar Vivek Mangipudi, Udit Desai, Mohsen Minaei, Mainack Mondal, and Aniket Kate. Uncovering impact of mental models towards adoption of multi-device crypto-wallets. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023: 30th Conference on Computer and Communications Security*, pages 3153–3167, Copenhagen, Denmark, November 26–30 2023. ACM Press. doi:10.1145/3576915.3623218.

39 Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001: 8th Conference on Computer and Communications Security*, pages 245–254, Philadelphia, PA, USA, November 5–8 2001. ACM Press. doi:10.1145/501983.502017.

40 Microchain Labs. Zk session keys. https://docs.microchain.microchainlabs.xyz/blog/second-post, 2024. Accessed: 2025-02-21.

41 Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 2551–2564, Los Angeles, CA, USA, November 7–11 2022. ACM Press. doi:10.1145/3548606.3560583.

42 Safe Core. Safe core protocol whitepaper. URL: https://github.com/5afe/safe-core-protocol-specs/blob/main/whitepaper.pdf, 2024. Accessed: 2024-10-11.

43 Qin Wang and Shiping Chen. Account abstraction, analysed. In *2023 IEEE International Conference on Blockchain (Blockchain)*, pages 323–331, 2023. doi:10.1109/Blockchain60715.2023.00057.

Fast, Private and Regulated Payments in Asynchronous Networks

Maxence Brugeres
LTCI, Telecom Paris, Institut Polytechnique de Paris, France
Forvis Mazars, Levallois-Perret, France

Victor Languille
LTCI, Telecom Paris, Institut Polytechnique de Paris, France
EDF R&D, Paris, France

Petr Kuznetsov
LTCI, Telecom Paris, Institut Polytechnique de Paris, France

Hamza Zarfaoui
LTCI, Telecom Paris, Institut Polytechnique de Paris, France

Abstract

We propose a decentralized asset-transfer system that enjoys *full privacy*: no party can learn the details of a transaction, except for its issuer and its recipient. Furthermore, the recipient is not aware of the sender's identity. Our system does not rely on consensus or synchrony assumptions, and therefore, it is *responsive*, since it runs at the actual network speed. Under the hood, every transaction creates a consumable *coin* equipped with a non-interactive zero-knowledge proof (NIZK) that confirms that the issuer has sufficient funds without revealing any information about her identity, the recipient's identity, or the payment amount. Moreover, we equip our system with a *regulatory enforcement* mechanism that can be used to regulate transfer limits or restrict specific addresses from sending or receiving funds, while preserving the system's privacy guarantees.

Finally, we report on *PaxPay*, our implementation of Fully Private Asset Transfer (FPAT) that uses the Gnark library for the NIZKs. In our benchmark, PaxPay exhibits better performance than earlier proposals that either ensure only partial privacy, require some kind of network synchrony or do not implement regulation features. Our system thus reconciles privacy, responsiveness, regulation enforcement and performance.

2012 ACM Subject Classification Security and privacy; Computer systems organization → Dependable and fault-tolerant systems and networks; Computing methodologies → Distributed computing methodologies

Keywords and phrases Anonymous, Asset Transfer, Asynchronous System, BFT, CBDC, NIZK, Payment System, Privacy, Regulation, Scalability, zk-SNARK

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.3

Related Version *Full Version*: `https://eprint.iacr.org/2025/098.pdf`

Funding *Maxence Brugeres*: Forvis Mazars Group
Victor Languille: EDF R&D
Petr Kuznetsov: CHIST-ERA-22-SPiDDS-05 (REDONDA project) and Agence Nationale de la Recherche (ANR-23-CHR4-0009)

1 Introduction

Payments, with and without consensus. Bitcoin [33] revolutionized the world of finance by proposing a fully decentralized asset-transfer system that allows an open set of users to consistently exchange assets without any mutual trust. Instead of relying on a central authority, users repeatedly engage in a form of *consensus* [22, 18] to maintain a replicated

© Maxence Brugeres, Victor Languille, Petr Kuznetsov, and Hamza Zarfaoui;
licensed under Creative Commons License CC-BY 4.0

ledger – an ever-growing record of all transactions. It was later observed [27, 26] that, strictly speaking, global consensus, a notoriously hard task [16, 17] that requires partial synchrony [22, 12], is not necessary in payments. In most common cases, when every account is maintained by a dedicated user, asset transfer can be implemented in an asynchronous, *responsive* way on top of the *reliable-broadcast* primitive [10] (instead of consensus).

Reliable broadcast is weaker than consensus as it allows the correct users to eventually and consistently agree on the *set* of issued transactions but not on their order. However, it turned out to be strong enough to prevent *double spending*, a major issue in asset-transfer systems. This observation gave rise to a series of purely asynchronous, *consensus-free* asset-transfer systems [14, 5, 31].

Privacy and regulations in payment systems. However, a decentralized asset-transfer system, as any replicated service, still faces a vital challenge of preserving privacy of its users. Indeed, in all the mentioned payment protocols, all transactions are inherently *public* and every user's activities are traceable. Hiding the amount of a payment offers *confidentiality* and hiding the identities of the sender and receiver offers *anonymity*. Fully *private* payment systems ought to offer both confidentiality and anonymity. Furthermore, many important payment applications, such as Central Bank Digital Currency (CBDC) [4], require mechanisms for *regulatory enforcement*, such as limiting transaction amounts.

Existing distributed asset-transfer systems ([6, 7, 37, 38, 41, 44, 45], to name a few) arbitrate a trade-off between the four following aspects: **(a) Privacy guarantees**, **(b) Model assumptions**, **(c) Regulation enforcement**, and **(d) Performance**. Reconciling these four aspects is a onerous challenge, as strengthening one aspect often weakens another. Table 1 provides a detailed comparison of several payment systems across these dimensions. This table and its content are detailed and discussed in Section 8.

FPAT and PaxPay. In this paper, we address this challenge. We introduce the abstraction of *Fully Private Asset Transfer* (FPAT) that maintains conventional safety and liveness properties of a payment system (informally, no asset is spent twice and every transaction takes effect), while at the same time making sure that no transaction's detail is leaked to a non-involved party. We then describe the *PaxPay* protocol, a FPAT implementation over an *asynchronous* network. PaxPay has been implemented in Golang using Gnark library [8].

As in UTXO transactions [33], to execute a transfer, the sender *spends* a set of *old coins* it has received earlier and *generates* a set of *new coins* of the same total value. Every coin is uniquely identified by its *serial number*. Every coin should be signed by a sufficiently large set (a 2/3 *quorum*) of *validators*, i.e., dedicated parties that maintain lists of spent coins and make sure that no user spends a same coin twice.

The protocol operates in an asynchronous network, where any number of users and less than one third of validators can be *Byzantine* (deviating arbitrarily from its algorithm). The rest of the validators are considered *semi-honest* (following their algorithm but possibly sharing their states with the adversary). To ensure that the transfer is legal, several assertions have to be checked: the total value of the spent coins equals the total value of the new coins, the user that calls the transfer is the owner of the spent coins, all the coins are properly signed, etc.

To provide *full privacy* (confidentiality and anonymity), PaxPay leverages several cryptographic primitives. The legality of transactions is verified within a non-interactive zero-knowledge proofs (NIZK). A *blind* signature scheme is used at the validators' side and these signatures are verified within the NIZK, allowing not to reveal the coins or their signatures

when spending them. To verify these signatures efficiently, we implemented the NIZK using the Groth16 [25] scheme and employed a pairing-based signature scheme [35]. We then selected a pair of elliptic curves that form a chain to instantiate these cryptographic primitives, allowing efficient verification of the signature within the NIZK (we refer to Section 7 for more details).

Regulation. We also describe and implement a regulation enforcement mechanism that can be built on top of our system. Similar to PEReDi and PARScoin [37, 38], our approach allows setting limits on individual transfer amounts or the cumulative amount transferred by a user since they joined the system. However, regulations for CBDCs mandate stringent measures for Anti-Money Laundering (AML) and Countering the Financing of Terrorism (CFT). These measures go beyond simply limiting transaction amounts.

PEReDi and PARScoin [37, 38] address these requirements and enable validators to reveal the details of transfers if needed. This could put at risk the privacy of the users. Their approach relies on additional trust assumptions, requiring non-Byzantine validators to be *honest* (i.e., not to share any information with the adversary) to maintain privacy. We avoid such extra assumptions by equipping the users with a mechanism to generate cryptographic proofs for arbitrary statements about the histories of their transactions. It enables extracting a requested information without disclosing the full content of the transaction histories.

Additionally, we incorporate a novel mechanism, empowering the regulator to impose *sanctions*, precluding specific addresses from interactions with other users. This ensures compliance with traditional financial sanctions emitted by the central banks or intergovernmental organizations (e.g., the UN[1]) and explicitly required by the European Central Bank for the digital Euro [4].

The regulatory mechanism is implemented by introducing an additional type of coin, referred to as the *compliance coin*. At a time, each user may possess several regular coins, but only one unique compliance coin. In each transfer, the compliance coin of the sender must be spent together with the regular coins. Then a new compliance coin containing the information of the transfer is generated (i.e., signed by a quorum of validators). The compliance coin thus allows us to keep track records of all the interactions of a user with the system, which can be seen as a commitment to the transaction history and some aggregated values. To generate an initially valid compliance coin, before joining the system, the user must be registered with a *regulatory authority*.

Our regulation framework incurs very low impact on the transaction throughput. It mainly affects the transaction proving time (by doubling it compared to the non-regulated case), which we do not consider a primary performance factor. In contrast, such a mechanism could be difficult and performance-intensive to implement in other protocols that rely on Sigma protocols [15], such as UTT [41], Zef [6], Peredi [37], and Parscoin [38], which use these protocols to construct their NIZKs.

Performance. Finally, we evaluated the performance of PaxPay, in comparison with state-of-the-art payment systems that provide privacy-preserving and/or regulatory-enforcement features (cf Table 1). In our benchmark (detailed in Section 7.2), we witness that PaxPay can process at least 8 times more transactions per second than any of these systems. The benchmark on AWS EC2 instances demonstrates a throughput of 925 transactions per second. The performance gains can be explained by the use of Groth16 as a succinct NIZK with low

[1] `https://main.un.org/securitycouncil/en/sanctions/information`

verification cost. In return, this comes at the cost of heavy computations required to generate a proof. The time required to create a transfer request with limited computational resources in PaxPay on a one-core CPU is up to $7s$ for example. However, given the throughput gains, this appears to be a good trade-off. Moreover, PaxPay's design allows for pushing the throughput even further by parallelizing the validator's load on several machines.

Full version. The full version of this paper [11] incorporates more details including the algorithms that implement PaxPay, the security proofs, the link toward PaxPay implementation and additional details on the benchmarks.

Summary. To sum up our contributions:

- We formalize the problem by introducing the *Fully Private Asset Transfer* abstraction (FPAT).
- We propose PaxPay, a protocol that implements a FPAT object, and prove its correctness.
- We implement PaxPay in Golang, mainly leveraging on the Gnark [8] library.
- PaxPay achieves a true reconciliation of **(a) Privacy**, **(b) Model Assumptions** (network synchrony and trust assumptions for validators), **(c) Regulation Enforcement**, and **(d) Performance**. Unlike earlier works that propose trade-offs, PaxPay excels in each of these aspects simultaneously, bringing together the best of all these dimensions:
  - Regarding **Privacy**, transactions in PaxPay are *confidential*, *fully anonymous*, and *unlinkable*, ensuring the most robust privacy guarantees.
  - Regarding **Model Assumptions**, the system tolerates up to one-third Byzantine validators, with the remaining validators only required to be *semi-honest*, and does not rely on the network synchrony.
  - Regarding **Regulation**, PaxPay provides a comprehensive regulation framework by enabling users to generate cryptographic proofs about their transaction history, setting spending limits, and incorporating a sanction mechanism preventing sanctioned addresses to receive or spend funds.
  - Finally, our implementation outperforms state-of-the-art protocols. Also, we show that the transaction throughput of PaxPay scales with the computational power of validators. The system can further increase its supported throughput by distributing validators across additional machines.
- PaxPay can find a variety of use cases such as a standalone payment system, a *layer 2* solution on top of a blockchain (brings scalability, compliance, or privacy) or a CBDC.

2 Related work

We overview payment systems related to our work, the summary of our comparative analysis is presented in Table 1 compares this work with the related works. Additional details about the comparison can be found in the full version of the paper.

Consensus-free payment systems. Gupta [27] and Guerraoui *et al.* [26] proposed a payment system that replaces the conventional consensus-based synchronization with the secure broadcast primitive [10, 32]. Systems like Astro [14], FastPay [5], Zef [6], PARScoin [38] and UTT [41] extend this approach, operating without consensus in asynchronous networks.

[2] Fully anonymous among all the users (Full) or anonymous among an anonymity set (AS).
[3] Semi-Honest (SH) or Honest (H).

Table 1 Comparison of PaxPay vs. Zcash [19], Lelantus [29], Quisquis [21], Zef [6], UTT [41], PRCash [44], PEReDi [37] and PARScoin [38].

	Zcash	Lelantus	Quisquis	Zef	UTT	PRCash	PEReDi	PARScoin	PaxPay
PRIVACY PROPERTIES									
Confidential transfers	●	●	●	◑	●	◑	●	●	●
Sender-anonymous transfers	●	●	●	❍	●	◑	◑	◑	●
Receiver-anonymous transfers	●	●	●	●	●	●	●	●	●
Unlinkable transfers	●	●	●	❍	●	◑	●	●	●
Anonymity strategy[2]	Full	AS	AS	Full	Full	Full	Full	Full	Full
MODEL ASSUMPTIONS									
Asynchronous network	❍	❍	❍	●	●	❍	❍	●	●
Correct validators[3]	SH	SH	SH	SH	SH	SH	H	H	SH
REGULATION FEATURES									
Limited held amount per user	❍	❍	❍	❍	❍	❍	●	❍	❍
Limited spendable amount per tx	❍	❍	❍	❍	❍	◑	●	●	●
Limited spendable amount in total	❍	❍	❍	❍	●	❍	●	●	●
Full asset tracing	❍	❍	❍	❍	❍	❍	●	●	❍
Sanctions	❍	❍	❍	❍	❍	❍	❍	❍	●
Provable transaction history	❍	❍	❍	❍	❍	❍	❍	❍	●
PERFORMANCE									
Transaction throughput (tx/s)	25	∅	∅	88	235	∅	∅	∅	925
Transaction latency (s)	1000	∅	∅	< 1	< 1	∅	∅	∅	< 1
NIZK Proving time (ms)	21K	2378	2110	438	60	100	3100	392	6959
NIZK Verification time (ms)	9	40	251	142	49	96	518	159	5

Private payment systems without regulation. Zerocash [7], Zexe [9], Monero [34], Quisqus [21] and Lelantus [29] are payment systems built on top of blockchains and consensus. They provide *privacy* via NIZKs or ring signatures. Zerocash and Zexe [7, 9] use *commitments* stored in Merkle trees and nullifiers to prevent double spending. Each transaction is accompanied by an NIZK to provide privacy but these systems offer inherently low transaction throughput. Monero [34] and Quisquis [21] relies on ring signatures to provide anonymity but only limits the sender and receiver's identity to a small subset of the users, known as the *anonymity set*. Compared to Monero, Quisquis offers stronger privacy guarantees in certain scenarios preventing possible de-anonymization.

Lelantus [29, 30] offers privacy via NIZK requires less advanced cryptographic assumptions than Zerocash or Zexe, for instance it does not require any trusted setup. It also proposes batch verification for validators.

All payment systems mentioned so far are built on a blockchain and, thus, require consensus. Baudet et al. [6] describe Zef, a private payment system, that assumes an asynchronous network but only provides *partial* anonymity and confidentiality. To improve performance, the transaction data in Zef is distributed over a set of *authorities*. Each authority can be sharded – i.e., distributed over several machines, as in Astro [14] and in this work.

Private payment systems with regulation. PRCash [44], Platypus [45], PEReDi [37], PARScoin [38] and UTT [41] incorporate certain regulatory features into private payment systems. A common element in their designs is the reliance on NIZK.

Garman et al. [23] were among the first to address regulation by enforcing *policies* such as fixing a spending limit on a system with a similar construction as Zerocash. Their system has not been implemented and, as it is built on top of Zerocash, it shares the same major drawbacks.

PRCash [44], as Zerocash, is blockchain-based and operates with commitments but offers only partial privacy. Regulation features proposed by PRCash are limited: users cannot spend more than a predefined amount set by the regulator within a certain time window.

UTT [41], a concurrent work to our paper, proposes an asynchronous private payment system with privacy guarantees, a low transaction latency and a high transaction throughput. Both UTT and PaxPay protocols follow a similar approach, using coin commitments and consistent broadcast to create and spend coins in a fully private manner. However, the regulation features offered by UTT are restricted to the *anonymity budget* that limits the amount of anonymous coins that a user can transfer. Compared to UTT, PaxPay offers a more comprehensive set of regulation features and exhibit better performance, due to improved NIZK construction.

Platypus [45] offers regulation features of holding limits, receiving limits, and spending limits. It uses NIZK with low verification costs to increase transaction throughput. However, unlike other payment systems described in this section, Platypus is a centralized payment system rather than a distributed one.

PEReDi [37] is an account-based system. The regulation framework enforces spending and receiving limits for each transaction and imposes restrictions on the maximum amount a user can hold. Additionally, the system allows validators to trace funds and reveal details of certain payments. Users encrypt their transaction details with a threshold encryption scheme so that a quorum of validators can decrypt them. PEReDi needs a synchronous network and *correct* (*i.e.*, non-Byzantine) validators must be *honest* (*i.e.*, strictly follow the protocol and avoid seeking additional information). In contrast, most systems discussed here (as well as ours) are designed to tolerate *semi-honest* correct validators.

Concurrently with this work, the same authors have recently proposed PARScoin [38]. PARScoin enhances PEReDi by allowing asynchronous transactions yet still requires honest validators and faces limited throughput. PARScoin's protocol is similar to the one employed by UTT and our work, relying on Byzantine consistent broadcast. Our work differs from PARScoin in the NIZK construction which offers better performance, fewer assumptions on the validators and additional regulations features. Also, neither PRCash, PARScoin nor PEReDi have conducted latency tests to evaluate the maximum transaction throughput of their systems, they only provide benchmarks of their primitives (such as the time to prove or verify a transaction).

Other decentralised privacy-preserving payment systems adopt this auditability approach such as [3, 36, 13].

Specifications for a responsive and private payment system. In this paper, we introduce a new specification (inspired by the formalism recently proposed by Albouy et al. [1]), as existing ones do not address our particular problem. They describe either private payment systems that cannot be implemented in asynchronous networks [19], asynchronous payment systems that lack anonymity [26, 14, 31], or non-sequential specifications that do not align with how specifications are typically defined in the distributed systems literature [37, 38, 41].

3 Model

3.1 Participants and adversary

The system is composed of two types of participants:

- A set $\mathcal{U}$ of U *users* of the payment system.
- A set $\mathcal{V}$ of $N = 3f + 1$ *validators*.

Here f denotes the maximum number of validators that can go *Byzantine*, i.e., deviate from the protocol. A non-Byzantine (faithful following the protocol) is called *correct*. Correct validators may, however, be *semi-honest* [20]: a correct party may try to learn as much as possible from the messages they receive from other parties, which may involve colluding and pooling their views together. In contrast, *honest* validators that are not trying to learn any information. The correct participants are thus following the protocol but they may exchange information with any other participant (Byzantine or not) to learn as much as possible. As we shall see, Byzantine participants pose challenges to all correctness aspects of our system (safety, liveness and privacy), while semi-honest participants affect only privacy.

The adversary $\mathcal{A}$ thus controls any number of users and all validators, but at most f validators can deviate from the protocol. $\mathcal{A}$ can be seen as a hybrid adversary between honest and semi-honest. We assume that $\mathcal{A}$ is *static* and *network-ignorant*: the set of Byzantine participants controlled by $\mathcal{A}$ is chosen a priori, and $\mathcal{A}$ has no information about message delay between the validators and the other participants. It can delay the messages but must eventually deliver them. Conventionally, the adversary $\mathcal{A}$ is computationally bounded. More precisely, $\mathcal{A}$ is *probabilistic polynomial-time* (PPT).

Every participant is provided with a pair of distinct public and secret addresses denoted *apk* and *ask*. These addresses are used to identify users and validators. The public addresses of the validators are known to every participant. Otherwise, a user only needs to know the identifiers of the users she engages in transfers with.

3.2 Network and communications

Concerning the network, we assume *asynchronous* but *reliable* communication. The participants can communicate via *anonymous*, *secure*, and *asynchronous network channels*. The channels do not modify or create messages. If a correct participants sends a message to a correct one, the message is eventually received, though we assume no bounds on the communication delays. The sender's identity is not known to the receiver, but the receiver can still respond to the sender. No other participant can tell who is the sender or the receiver, or tamper with the message content. We could use a network based on Syverson et al. [40] work. We consider that the latency distribution is the same for all the channels.

3.3 Cryptographic tools

Our protocol makes use of several cryptographic tools that we list below. Due to the space constraints, we only give informal definitions and refer to [24] for more details. Also, in the algorithms mentioned below, we omit the security parameters taken as inputs. Some primitives require a setup phase, which will be handled by a trusted third party $\mathcal{T}$. Note that these setups could be carried out via MPC [20] between the validators.

(k, N)-Threshold Blind Signature Scheme. Tuple of algorithms $\Pi_{\mathsf{sig}} = (\mathsf{SETUP}_{\mathsf{Sig}}, \mathsf{BLIND}, \mathsf{SIGN}, \mathsf{UNBLIND}, \mathsf{AGGREGATE}, \mathsf{VERIFY}_{\mathsf{Sig}})$ allowing to divide a secret key sk in n fragments $[sk_i]_{i=1}^{N}$ between n signers, such that valid signatures from any subset of k signers

can be aggregated into a valid signature on behalf of the corresponding public key pk_{agg}. Each fragment sk_i has a corresponding public key pk_i so that a partial signature σ_i generated with sk_i can be verified with respect to pk_i. Moreover, the signers sign a blinding version $\tilde{m}$ of a message m such that no information on m can be derived from $\tilde{m}$. A valid signature with respect to m can then be computed. Signature are unforgeable, which means that no PPT adversary can forge a valid partial signature σ_i of a message m that correctly verifies against pk_i without the knowledge of sk_i. As a blind signature, this should not also be possible for a signer to eventually make the link between the signature is has issued, and the final signature (after UNBLIND and AGGREGATE).

- $\mathsf{SETUP}_{\mathsf{Sig}}(k, N) \rightarrow ([sk_i]_{i=1}^N, [pk_i]_{i=1}^N, pk_{agg})$: Randomized algorithm run by a trusted party. Takes as input threshold parameters (k, N) and returns a list of N signing keys $[sk_i]_{i=1}^N$ and one corresponding verification key pk_{agg}.
- $\mathsf{BLIND}(m, b) \rightarrow (\tilde{m})$: Takes as input a message m and a blinding factor b, returns a blinded message $\tilde{m}$.
- $\mathsf{SIGN}(\tilde{m}, sk_i) \rightarrow \tilde{\sigma_i}$: Takes as input a blinded message $\tilde{m}$ and a secret key sk_i, and returns a partial blinded signature $\tilde{\sigma_i}$.
- $\mathsf{UNBLIND}(\tilde{m}, \tilde{\sigma}_i, b) \rightarrow \sigma_i$: Takes as input a blinded message $\tilde{m}$, the blinding factor b used to blind m and the corresponding partial blinded signature $\tilde{\sigma_i}$, and returns a valid partial signature σ_i for the original message m.
- $\mathsf{AGGREGATE}([\sigma_i]_{i=1}^k) \rightarrow \sigma$: Takes as input a list of k signatures $[\sigma_i]_{i=1}^k$ and produce a signature σ such that σ is a valid signature on behalf of the public key pk_{agg} if and only if all σ_i are valid signatures for distinct public keys pk_i.
- $\mathsf{VERIFY}_{\mathsf{Sig}}(m, \sigma, pk) \rightarrow b$: Takes as input a partial or aggregated signature σ of a message m and returns a bit b of value 1 if σ is valid with respect to pk and 0 otherwise. The signature can be a partial or aggregated signature.

***Collision-Resistant* and *Preimage-Resistant* Pseudorandom Function Family [24].** A family of functions $\mathsf{PRF} = \{\mathsf{PRF}_s : \{0,1\}^* \rightarrow \{0,1\}^{O(|s|)}\}_s$, where s denotes a seed, computationally indistinguishable from a random function family. *Collision-Resistance* means here that it is computationally infeasible to find couples $(s, x) \neq (s', x')$ such that $\mathsf{PRF}_s(x) = \mathsf{PRF}_{s'}(x')$. *Preimage-Resistance* means that it is computationally infeasible, given y, to find (s, x) such that $\mathsf{PRF}_s(x) = y$.

Non-Interactive Zero-Knowledge Proof (NIZK) [24]. A tuple of algorithms $\Pi_{\mathsf{proof}} = (\mathsf{SETUP}_{\mathsf{NIZK}}, \mathsf{PROVE}, \mathsf{VERIFY}_{\mathsf{NIZK}})$ allowing a *prover* to prove to a *verifier* that, given a statement defined by an NP relation $\mathcal{R}(a, b)$ and an instance public_input, she knows a witness private_input such that $\mathcal{R}(\mathsf{public_input}, \mathsf{private_input})$.

- $\mathsf{SETUP}_{\mathsf{NIZK}}(\mathcal{R}) \rightarrow \mathsf{pp}_{\mathsf{NIZK}}$: Randomised algorithm run by a trusted party. Takes as input a relation $\mathcal{R}$ and outputs public parameters $\mathsf{pp}_{\mathsf{NIZK}}$ (also known as common reference string). These public parameters are taken as input by the two following algorithms PROVE and VERIFY, but we omit it to lighten the notation.
- $\mathsf{PROVE}(\mathsf{public_input}, \mathsf{private_input}, \mathsf{pp}_{\mathsf{NIZK}}) \rightarrow \pi$: Randomised algorithm. Takes as inputs instance and witness. Outputs a proof π such that :
  $\mathcal{R}(\mathsf{public_input}, \mathsf{private_input})$.
- $\mathsf{VERIFY}_{\mathsf{NIZK}}(\mathsf{public_input}, \mathsf{pp}_{\mathsf{NIZK}}, \pi) \rightarrow \mathsf{b}$: Takes as input an instance public_input and a proof π. Outputs 1 if the proof is valid and 0 otherwise.

Incremental commitment scheme. A tuple of algorithms $\Pi_{com} = (\mathsf{COM}, \mathsf{INCR})$ that allows us, given a sequence of messages $[m_i]_{i=1}^n$ and a sequence of randomness $[r_i]_{i=1}^n$, to produce a *commitment* c. The commitment is *hiding*, i.e., no information on $[m_i]_{i=1}^n$ can be derived from c without prior knowledge on $[r_i]_{i=1}^n$. The commitment is also *binding*, meaning that given a sequence of couples $[(m_i, r_i)]_{i=1}^n$ that commits to a value c, it should be computationally infeasible for a PPT adversary to compute $[(m'_i, r'_i)]_{i=1}^n$ such that $[(m'_i, r'_i)]_{i=1}^n \neq [(m_i, r_i)]_{i=1}^n$ also commits to c. The commitment scheme is also *incremental*: let us consider a sequence $[(m_i, r_i)]_{i=1}^n$, its commitment c and a couple (m', r'). Given the knowledge of c, m' and r', it is possible to compute c' that commits to $[(m_1, r_1), (m_2, r2), ..., (m_n, r_n), (m', r')]$:

- $\mathsf{COM}([(m_i, r_i)]_{i=1}^n) \rightarrow c$ takes as input a sequence of messages and randomness and returns the corresponding commitment. COM is hiding and binding.
- $\mathsf{INCR}(c, m', r') \rightarrow c'$ takes as input a commitment c of a sequence $[(m_i, r_i)]_{i=1}^n$, a new message m' and a randomness r'. It returns c', the commitment to the sequence $[(m_1, r_1), (m_2, r2), ..., (m_n, r_n), (m', r')]$.

4 Fully private asset transfer (FPAT)

State and interface. At the abstract level, the state of the FPAT object is represented as an array of U positive integer values $[v_k]_{k=1}^U$, one for each user, interpreted as the current balances of the users' accounts. Let $[v_k^{init}]_{k=1}^U$ be the *initial state* of the object. The object exports two operations: *transfer* and *balance*. Assuming that a user u invokes an operation, they are defined as follows:

- $transfer_u(v, w)/r$ takes as inputs a value v and a user identifier w. It transfers the amount v from u to w: updates a state $[v_k]_{k=1}^U$ to the state $[v'_k]_{k=1}^U$ where $v'_k = v_k + v$ if $k = w \wedge u \neq w$, $v'_k = v_k - v$ if $k = u \wedge u \neq w$ and $v'_k = v_k$ otherwise. It returns a binary response $r = \mathsf{confirm}$ if it succeeds and $r = \mathsf{fail}$ otherwise.
- $balance_u()/v_u$ takes no inputs and returns the value v_u stored at location u of the state $[v_k]_{k=1}^U$.

Let $\mathcal{O}$ be a set of FPAT *operations* – invocations of *transfer* and *balances* provided with matching responses, each associated with a distinct user. Let $transfer_u(*, *)/C$ (resp., $transfer_u(*, *)/F$) denotes a transfer operation invoked by a user u that returns *confirm* (resp. *fail*). For each user u, we define a function $total_i(\mathcal{O})$ as follows:

$$total_u(\mathcal{O}) = v_u^{init} + \sum_{transfer_*(v,u)/C \in \mathcal{O}} v \quad - \sum_{transfer_u(v,*)/C \in \mathcal{O}} v$$

$total_u(\mathcal{O})$ is thus defined as the current *balance* of u after all successful transfers in $\mathcal{O}$ complete: the initial amount owned by u, minus all the funds sent by u plus all the funds received by u in the set of operation $\mathcal{O}$.

A *sequential history* S (of FPAT) is a totally ordered set of FPAT operations, let $\prec_S$ be this order. Let $S|_u$ denotes the sub-sequence of S consisting of the events of user u. We say that S is *legal* if:

1. $\forall o = transfer_u(v, w)/C \in S \quad v \leq total_u(\{o' \in S : o' \prec_S o\})$
2. $\forall o = balance_u()/v \in S \quad v \leq total_u(\{o' \in S : o' \prec_S o\})$
3. If an operation $balance_u()$ returning v_1 directly precedes a $transfer_u(v_2, w)$ operation in $S|_u$, and $v_2 \leq v_1$, then the transfer operation cannot return fail.

Histories and serializations. Consider an *execution* of a FPAT algorithm: a sequence of *events* produced by the algorithm, such as invocations and responses of FPAT operations, sending and receiving messages, etc. A *local history* L_i of a user i is the sequence of *operations* invoked by i in that execution. We assume that correct users are *well-formed* – they never invoke a new operation before the previous one returns, and thus if u is correct, then L_u is sequential. In case the last operation of L_u is *incomplete* (not followed by a response), we can add any matching response and get a *completion* of L_u. Now a *history* H is a vector $(L_1, L_2, ..., L_{|\mathcal{U}|})$ of local histories, one for each user. Notice that if the history is produced by an execution of a FPAT algorithm, only the local histories of correct users are of interest for us: a Byzantine user is not obliged to follow the protocol.

A sequential (FPAT) history S is a *serialization* of a history $H = (L_1, L_2, ..., L_{|\mathcal{U}|})$ if for each *correct* user u, there exists $\hat{L}_u$, a completion of L_u, such that $S|_u = \hat{L}_u$ ($\prec_S$ respects the local order $\prec_{\hat{L}_u}$). S can be seen as a *global interpretation* of the local histories of correct users. Notice that we allow any operations to be executed by Byzantine users in S, as long as it "makes sense" to the correct ones.

FPAT-Safety. An implementation of a FPAT-object is *FPAT-Safe* if and only if, for any finite history of execution H it produces, there exists a serialization S of H which is legal.

FPAT-Liveness. Liveness ensures that (1) every operation invoked by a correct user eventually completes and (2) considering two correct users u and w, if w transfers money to u, then u eventually receives this money. Let us consider the existence of a global time during the execution of an implemented FPAT object. An implementation of a FPAT-object is *FPAT-Live* if:

1. All *transfer* and *balance* operations terminate for correct users.
2. Consider a correct user u. For each operation $transfer_*(*, u)$ completed during the execution at time t, there exists a time $t' \geq t$ such that any operation *balance* inserted at time $t'' \geq t'$ will return a value v such that:

$$v \geq \sum_{\substack{transfer_*(v_+,u) \\ \text{invoked by correct users} \\ \text{completed before time } t}} v_+ \; - \sum_{\substack{transfer_u(v_-,*) \\ \text{completed before time } t''}} v_-$$

FPAT-privacy. We capture the privacy guarantees of FPAT through a *distinguishing game* $\mathcal{G}^{priv}$ described in details the full version of the paper. *FPAT-privacy* has important implications that can be expressed as a collection of properties we use here. Let ϵ be a negligible function and λ the security parameter. Consider a protocol execution, let H be its history and $\mathcal{U}_{Byz}$ the set of users controlled by the adversary, among the U users of the system. Let $o = transfer_u(v, w) \in H$ be any *transfer* operation such that u is honest. Then for each guess (u', v', w') made by an adversary with no prior knowledge, where u' is the payment sender, v' is the value, and w' is the payment recipient, the following properties hold:

1. **Sender-anonymity**: $\mathbb{P}(u' = u) \leq \frac{1}{U - |\mathcal{U}_{Byz}|} + \epsilon(\lambda)$, i.e., the adversary only knows that the sender is not itself.
2. **Receiver-anonymity**: If w is honest, then $\mathbb{P}(w' = w) \leq \frac{1}{U - |\mathcal{U}_{Byz}|} + \epsilon(\lambda)$, i.e., the adversary only knows that the receiver is not itself.
3. **Confidentiality**: If w is honest, then $\mathbb{P}(v' = v) \leq \epsilon(\lambda)$, i.e.,the adversary cannot guess the amount of the transaction.

These three properties together constitute **full privacy**. Additionally, *FPAT-privacy*, as captured by the *distinguishing game* $\mathcal{G}^{priv}$, implies **unlinkability**: given two transfers, no adversary can determine whether the sender, receiver, or amount is the same in both transfers.

5 PaxPay Protocol

We overview our FPAT implementation below and delegate detailed algorithms and proofs of correctness to the full version of the paper.

Setup. Every user is identified by her public address *apk* that is derived from a secret address *ask* as follows: $apk = PRF_{ask}(0)$ [4]

The protocol uses a $(2f+1, N)$-threshold blind signature scheme, as defined in Section 3.3. The secret signing keys $[sk_i]_{i=1}^{N}$ are held by each of the validators. The aggregated public key is denoted by pk_{agg} The protocol will also make use of NIZK, as defined in Section 3.3. The setup algorithms for these primitives are again described in [11].

Coin structure. A *coin* is a tuple $\mathbf{c} = (v, apk, \rho)$ with:

- v the (integer) value of the coin.
- *apk* the public address of the owner of the coin.
- ρ the seed of the coin, from which the serial number is derived.

Coins are similar to unspent transactions (UTXO) in Bitcoin: to make a payment, a user "spends" *old* coins and creates *new* coins whose owners are the payment recipients.

Coin validity. Using a quorum of validators signatures (inspired by Byzantine Consistent Broadcast), we decide whether a coin is valid or not. In concrete terms, a coin is valid if it has been signed by $2f + 1$ validators. During a transfer, several old coins $[\mathbf{c}_i^{old}]_{i\in[1,n]}$ are spent to create new coins $[\mathbf{c}_j^{new}]_{j\in[1,m]}$. To make sure that a coin it not spent twice, a unique *serial number* is derived from the coin's seed ρ as $sn = \mathsf{PRF}_{ask}(\rho)$. Each validator maintains a list snList of the old coins' serial numbers. The coins $[\mathbf{c}_i^{old}]_{i\in[1,n]}$ are considered spent when a quorum of $2f + 1$ validators have appended the corresponding serial numbers $[sn_i^{old}]_{i\in[1,n]}$ to their snList.

Intuitively, as the computation of sn_i^{old} is using the PRF and the secret address of the payer, no party can link sn_i^{old} to ρ_i^{old} or apk_i^{old}, and thus the sender-anonymity property is preserved. The full version of the paper details an algorithm that initializes the balances for the initial set of users, by creating coins for each of them.[5]

Local variables. User storage consists of:

- apk/ask: its public/secret address pair
- coinList $= [(\mathbf{c}_i, \sigma_i)]_i$: the set of coins owned by the user associated with the aggregated signature of each coin.
- rhoList: the list of all seed ρ corresponding to coins received by the user

[4] The address generation is detailed in [11] and shows the address generation process by sampling a random *ask* and deriving the *apk*.

[5] As discussed in Section 8, depending on the use cases of the system, this algorithm might not be executed. Instead, users might freely join the system and call a Mint algorithm that provides them with new coins.

Validator storage consists of:

- sk: its secret signing key
- snList: the list of serial numbers identifying spent coins.
- signedCoinList: the list of all blinded coins signed by the replica.

Transfer. Assuming a user has enough funds, she can issue payments to several recipients $[apk_j^{new}]_{j=1}^m$ in one transfer by sending v_j^{new} to apk_j^{new}. The full algorithm is described in [11]. The user first chooses n (old) coins denoted $[\mathbf{c}_i^{old}]_{i=1}^n$ and their signatures $[\sigma_i^{old}]_{i=1}^n$ from her coinList, where $\mathbf{c}_i^{old} = (v_i^{old}, apk_i^{old}, \rho_i^{old})$, such that the total value of the old coins does not fall below the total value payed in the transfer: $\sum_{i=1}^n v_i^{old} \geq \sum_{j=1}^m v_j^{new}$. (Otherwise, the transfer operation returns fail.) The user then computes the serial number $[sn_i^{old}]_{i=1}^n$ of every old coin as follows: $sn_i^{old} = \mathsf{PRF}_{ask_i^{old}}(\rho_i^{old})$.

The j new coins are then produced by the user according to the values and addresses to pay: $\forall j \in [1, m]$, $\mathbf{c}_j^{new} = (v_j^{new}, apk_j^{new}, \rho_j^{new})$. ρ_j^{new} is derived as $\rho_j^{new} = \mathsf{PRF}_{\rho_{seed}}(sn_1^{old}||...||sn_n^{old}||j)$, where ρ_{seed} is sampled randomly. This binds each new coin to the old coins spent for its creation. This way, we make sure that the validators signing $\mathbf{c}^{new}$ mark the same old coins as spent. Since the value of the chosen old coins might exceed the value of the new coins, the user also produces a *redeem coin* $\mathbf{c}_{m+1}^{new} = (v_{m+1}^{new}, apk_{m+1}^{new}, \rho_{m+1}^{new})$, with $apk_{m+1}^{new} = apk$ and v_{m+1}^{new} the exceeding value. The coins $[\mathbf{c}_j^{new}]_{j=1}^{m+1}$ are then *blinded*. The blinding of $\mathbf{c}_j^{new}$ is denoted $\tilde{\mathbf{c}}_j^{new} = \mathsf{BLIND}(\mathbf{c}_j^{new}, b_j)$, where b_j is a randomly sampled blinding factor.

The sender of the payment then computes an NIZK with:

- public_input : $([sn_i^{old}]_{i=1}^n, [\tilde{\mathbf{c}}_j^{new}]_{j=1}^{m+1})$
- private_input : $([\mathbf{c}_i^{old}]_{i=1}^n, [\sigma_i^{old}]_{i=1}^n, [ask_i^{old}]_{i=1}^n, [\mathbf{c}_j^{new}]_{j=1}^{m+1}, [b_j]_{j=1}^{m+1}, \rho_{seed})$

This NIZK $\pi_{transfer}$ proves the following relations:

1. Serial numbers $[sn_i^{old}]_{i=1}^n$ are correctly derived from the old coins $[\mathbf{c}_i^{old}]_{i=1}^n$.
2. New coins $[\mathbf{c}_j^{new}]_{j=1}^{m+1}$ are correctly derived from the old coins $[\mathbf{c}_i^{old}]_{i=1}^n$.
3. Blinded coins $[\tilde{\mathbf{c}}_j^{new}]_{j=1}^{m+1}$ are correctly derived from $[\mathbf{c}_j^{new}]_{j=1}^{m+1}$ and $[b_j]_{j=1}^{m+1}$.
4. Signatures $[\sigma_i^{old}]_{i=1}^n$ of the coin $[\mathbf{c}_i^{old}]_{i=1}^n$ are correct.
5. Private addresses $[ask_i^{old}]_{i=1}^n$ match the public address $[apk_i^{old}]_{i=1}^n$.
6. The sum of the values of the old coins $[\mathbf{c}_i^{old}]_{i=1}^n$ equals the sum of the values of the new coins $[\mathbf{c}_j^{new}]_{j=1}^{m+1}$.

The user can now send the NIZK $\pi_{transfer}$ along with the public inputs to all the validators. The detailed algorithm run by the validators is detailed in [11]. Once a validator receives the proof:

1. It checks that none of the $[sn_i^{old}]_{i=1}^n$ appears in its snList;
2. It checks that the proof π is correct;
3. If the last two conditions are fulfilled, it adds all $[sn_i^{old}]_{i=1}^n$ to snList, add all $[\tilde{\mathbf{c}}_j^{new}]_{j=1}^{m+1}$ to signedCoinList, it signs all the coins $[\tilde{\mathbf{c}}_j^{new}]_{j=1}^{m+1}$ and returns the blinded partial signatures.
4. If any of the previous conditions is not fulfilled, the validator checks if the blinded coins $[\tilde{\mathbf{c}}_j^{new}]_{j=1}^{m+1}$ appear in signedCoinList. If they all do, then the validator still sends back the signatures. This is done because a Byzantine validator p might receive a transfer request from a user u and send it to other validators using a private channel. As a result, other validators might answer p before answering u, preventing u from receiving the signatures while the old coins are actually spent (their serial numbers would have been added to snList already).

Once the user receives $2f+1$ valid partial signatures for the coins $[\tilde{\mathbf{c}}_j^{new}]_{j=1}^{m+1}$ from $2f+1$ validators, she can unblind and aggregate them to form $m+1$ signatures $[\sigma_j]_{j=1}^{m+1}$ that are valid signatures for $[\mathbf{c}_j^{new}]_{j=1}^{m+1}$. She can now send each couple $(\mathbf{c}_j^{new}, \sigma_j^{new})$ to apk_j^{new}. The user apk_j^{new} only accepts the payment if $\rho_j^{new} \notin \mathsf{rhoList}_{apk_j^{new}}$ and $\mathsf{VERIFY}_{\mathsf{Sig}}(\mathbf{c}_j^{new}, \sigma_j^{new}, pk_{agg}) == 1$.[6]

Double spending. Consider a coin $\mathbf{c}^{\text{old}}$ that has been spent. Hence, a set $\mathcal{V}_1$ of $2f+1$ validators have claimed to have added sn^{old} to their snList, and at least $f+1$ out of them are correct. If a Byzantine user tries to spend $\mathbf{c}^{\mathbf{old}}$ again, it should collect confirmations from a set $\mathcal{V}_2$ of $2f+1$ validators that will have to add sn^{old} in their snList again. Since there are $3f+1$ validators in total, $\mathcal{V}_1$ and $\mathcal{V}_2$ have at least one correct validator in common. It must refuse to sign the second transaction because sn^{old} is already in its snList, so double spending is prevented and FPAT-Safety is guaranteed. The full proof is presented in [11].

Example. Figure 1 depicts a transfer:

1. Alice wants to pay 3 recipients with 2 coins. She generates 3 blinded coins and a matching NIZK, and sends them to the validators.
2. Each validator checks the validity of the NIZK and that the old coins are not in its snList, and, if so, sign the blinded coins and add the serial numbers to their snList.
3. Alice receives the signatures.
4. For each coin: once $2f+1$ blinded partial signatures are received, Alice unblinds and then aggregates them into one aggregated signature.
5. Alice sends the coin tuples and their signatures to their recipients.

6 Regulatory enforcement

PaxPay is designed as a private payment system. As in PEReDi [37], PRCash [44] or Platypus [45], the protocol can be enhanced to be *regulation-compiant*. The use of succinct proofs limits the impact of regulatory enforcement on the system's performance. This section describes how to build such a regulatory enforcement on top of the protocol described above.

Our regulatory enforcement is achieved via *compliance coins*. Each user owns its unique compliance coin. The coin commits to some data related to the user and all his transactions. The user must attach the compliance coin to each of it transactions as an input and the validators will sign the updated compliance coin. Well-formedness of the updated compliance coin is proved through NIZK. As for a classical coin, the old compliance coin is spent by revealing its serial number. As a result, a user has one one valid compliance coin at a time and this coin acts as an append-only tracing mechanism. We introduce the following regulation features, shown in Table 1:

- *Limited held amount per user:* Users cannot hold more than V_{held}^{max}.
- *Full asset tracing:* A trusted authority (centralized or decentralized) can reveal the content of transaction to trace back assets and users activities.
- *Limited spendable amount per tx:* Users cannot spend more than V_{sent}^{max} in one transfer.
- *Limited spendable amount in total:* Users cannot spend more than V_{total}^{max} in total.

[6] For convenience, we allow the TRANSFER algorithm to send money to several recipients in a single call, in contrast with the specification, which allows only one recipient at a time. As explained in the security proof in [11], this more generic transfer can be seen as a batch of successive transfer calls, each destined to a single user.

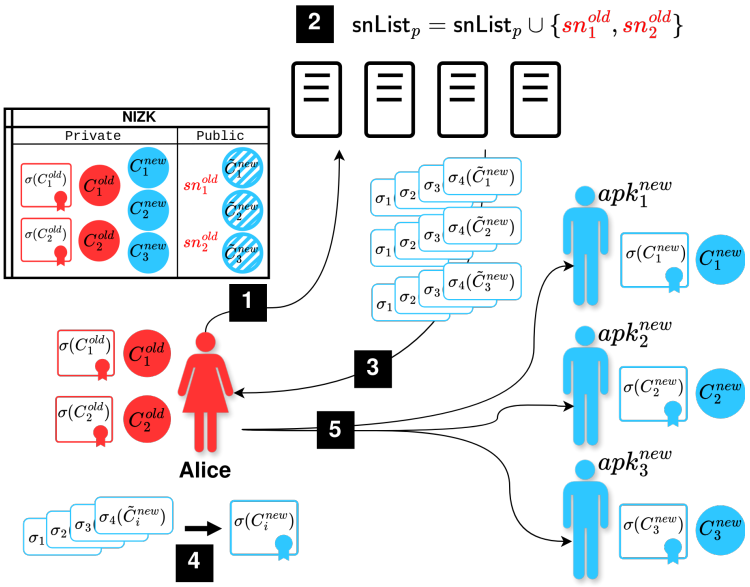


Figure 1 PaxPay example: Alice pays 3 recipients with 2 old coins.

- *Sanctions:* Users in the sanction list cannot send or receive funds.
- *Provable transaction history:* A user can prove arbitrary statements about her transactions. For instance, reveal all of her transactions and prove that the revealed set is complete (no transactions are missing).

Our regulatory construction enforce *Limited spendable amount per tx*, *Limited spendable amount in total*, *Sanction list* and *Provable transaction history*. V_{sent}^{max}, V_{total}^{max} and *SancList* are public parameters chosen and potentially updated by the regulator. The sanction list *SancList* is represented as a sorted Merkle tree, which allows efficient proof of non-membership.

Compliance coin. The compliance coin of a user u is defined as a tuple $cc_u = (apk, \rho, v, com)$ where:

- apk is u's sole public address
- ρ the coin seed, as for a normal coin
- v is the total amount of money sent so far by u
- com is the commitment to the list of all the transfer done by u so far. For each coin created, it commits the tuple (apk, v) with apk the public key of the receiver and v the value sent. It uses the incremental commitment introduced in Section 3.3

com allows to trace u's activities and allows u to prove additional statements on her transaction history in case of an update of the regulation.

Registration. Adding a user registration process is essential for regulatory enforcement, to comply with the *know-your-customer* (KYC) and *customer-due-diligence* (CDD) checks. Additionally, it ensures the uniqueness of the compliance coin for each physical user.

To this end, the validator storage is provided with a new list registeredList, the list of the enrolled users associated with their public key. registeredList is used to make sure that a physical user can only register once on the system. The registration process is as follows for the user u:

1. Generate a random ρ and computes $cc_u = (apk_u, \rho, 0, 0)$ and its blinding $\tilde{cc}_u$ (with blinding factor b).
2. Computes an NIZK $\pi_{register}$ that takes $(\tilde{cc}_u, apk_u)$ as public input and (ask_u, cc_u, b) as private input. It proves that:
   a. $\tilde{cc}_u = \mathsf{BLIND}((apk_u, \rho, 0, 0), b)$.
   b. ask_u and apk_u form a correct secret/public address pair.
3. Send to each validator $\pi_{register}$ and $(\tilde{cc}_u, apk_u)$.
4. Upon correct KYC and CDD proving u's identity, each validator check if $\pi_{register}$ is correct and that: $(u, *) \notin \mathsf{registeredList}$ (u has registered no public address yet). If so, they sign $\tilde{cc}_u$, send it back to u, and add (u, apk_u) to $\mathsf{registeredList}$.
5. Once u has received $2f + 1$ partial signature, she can unblind and aggregates the partial signatures to form a valid signature for cc_u.

The detailed algorithm is described in [11].

Transfer. Section 5 described how a transfer is handled by a user u, by spending n old coins $[\mathbf{c}_i^{old}]_{i=1}^{n}$ and creating $m + 1$ new coins $[\tilde{\mathbf{c}}_j^{new}]_{j=1}^{m+1}$ to pay m recipients (m coins are created to pay the recipient and a redeem coin is also created with index $m + 1$ to get the excess of money back to u). Now the user also has to update a compliance coin at each payment and prove that their transfer is compliant. The following additional steps are thus required. She generates m random values: $[r_j]_{j=1}^{m}$. The user provides an additional proof π_{comply}. This proof takes the following public inputs:

- $\tilde{cc}_u^{new}$ the blinding of the new compliance coin
- sn_{cc}^{old} the serial number of the old compliance
- V_{sent}^{max} the maximum amount that can be transferred in one transfer
- V_{total}^{max} the total amount of money that u can transfer in her use of the system
- $SancList$ the list of addresses under sanctions

π_{comply} takes the following as private input:

- $cc^{old} = (apk_{cc}^{old}, \rho_{cc}^{old}, v_{cc}^{old}, com^{old})$ the old compliance coin
- $cc^{new} = (apk_{cc}^{new}, \rho_{cc}^{new}, v_{cc}^{new}, com^{new})$ the new compliance coin
- ask_{cc}^{old} the secret address corresponding to apk_{cc}^{old}

π_{comply} should verify the following clauses:

1. $\forall i \in [1, n], apk_i^{old} = apk_{cc}^{old}$
2. $apk_{cc}^{old} = \mathsf{PRF}_{ask_{cc}^{old}}(0)$
3. $apk_{m+1}^{new} = apk_{cc}^{old}$
4. $apk_{cc}^{new} = apk_{cc}^{old}$
5. $\rho_{cc}^{new} = \mathsf{PRF}_{\rho_{seed}}(sn_1^{old}||...||sn_n^{old}||m + 2)$
6. $v_{cc}^{new} = v_{cc}^{old} + \sum_{1 \le j \le m} v_j^{new}$
7. $com_0 = com^{old} \wedge \forall j \in [1, m], com_j = \mathsf{INCR}(com_{j-1}, apk_j^{new}, v_j^{new}, r_j) \wedge com^{new} = com_m$
8. $\sum_{1 \le j \le m} v_j^{new} \le V_{sent}^{max}$
9. $v_{cc}^{new} \le V_{total}^{max}$
10. $apk_{cc}^{old} \notin SancList$
11. $\forall j \in [1, m], apk_j^{new} \notin SancList$

Conditions (1) ensures that all the input coins belong to the owner of the input compliance coin. Condition (2) ensures that u, the user that invokes *transfer*, is the owner of the compliance coin. Condition (3) ensures that the redeem coin will belong to u. Conditions (4), (5), (6) and (7) ensure that the new compliance coin is well-formed. Condition (8), (9) and (10) ensure that the transfer is compliant. (8) verifies that the amount of the transfer does not exceed that maximum allowed for a single transfer. (9) verifies that the total amount of

money sent by u does not exceed that maximum amount. (10) verifies that the sender is not a user under sanctions. (11) verifies that none of the receivers of the transfer is under sanction.

The user storage is also augmented with a list comList, in which she stores the tuples (apk, v, r) that she has used at each transfer. At the end of the transfer described above, she thus append to comList the following: $\forall j, j \in [1, m], (apk_j^{new}, v_j^{new}, r_j)$. She can later use comList and her compliance coin to prove any statement about her transaction.

Upon receiving correct proofs $(\pi_{comply}, \pi_{transfer})$ and their public parameters, the validators execute the same algorithm as for a non regulated transfer, except that they verify both π_{comply} and $\pi_{transfer}$. They can then sign the news coins and then send the signature back. The detailed algorithm is described in [11].

7 PaxPay implementation and performance

PaxPay implementation in Golang is available on Github. We developed with two implementations, with and without regulation support. Below we overview the cryptographic tools we employed and report on the performance of our protocols.

7.1 Cryptographic building blocks

NIZK. NIZK proofs are implemented using the Groth16 [25] protocol. Groth16 allows for constructing succinct non-interactive arguments of knowledge (SNARKs) with constant verification complexity, regardless of the complexity of the relation $\mathcal{R}$ it attests to. The relation $\mathcal{R}$ is represented by an arithmetic circuit. We instantiate this scheme with the BW6-761 [28] curve, designed to cycle with the BL12-377 [28] curve, allowing for efficient pairing verification over BL12-377 within the SNARK.

In the regulated version, each transfer requires two NIZK. For performance reasons, these two proofs $\pi_{transfer}$ and π_{comply} are merged into a single proof with a unique circuit. The proof of non-membership in the merkle tree is implemented as follows: the sanction list has the following form: $SancList = [s_1, s_2, ..., s_\xi]$. $SancList$ is supposed to be published sorted by the regulator. To verify the non-inclusion of an address apk in the list, we prove that there exist s_i and s_j such that:

- s_i and s_j are elements of $SancList$.
- $j = i + 1$ (s_i and s_j are consecutive elements of $SancList$).
- $s_i < apk < s_j$ (s_1 and s_ξ are set to $0x00..00$ and $0xFF..FF$ to make sure this constraint can always be fulfilled).

Hash functions and PRF. To optimize the computational cost of the proving algorithm in SNARK, we rely on MiMC [2]. MiMC is an arithmetization-oriented, collision-resistant and preimage-resistant hash function that is efficiently represented as arithmetic circuits. We use it to derive the the seeds ρ of created coins from the serial numbers sn of spent coins. The pseudorandom function family $\mathsf{PRF}_x()$ is derived from a MiMC function H as $\mathsf{PRF}_x(m) := \mathsf{H}(m||x)$.[7]

[7] The natural way to implement a PRF from a hash function is to consider $\mathsf{PRF}_x := \mathsf{H}(m||x)$, however, replacing concatenation by an addition in the field does not compromise security in our case while reducing the computational cost.

Incremental commitment scheme. The incremental commitment scheme is also derived from the MiMC function H. Given a commitment com^{old} and a message m to be added to the commitment with a randomness r, $\mathsf{INCR}(com^{old}, m, r)$ is defined as follows: $\mathsf{INCR}(com^{old}, m, r) = \mathsf{H}(com^{old}||m||r)$. The commitment function COM is then an iteration of increments INCR over the sequence of messages and randomnesses provided.

(k, N)-threshold blind signature. Our signature scheme is based on a slightly modified version of the Coconut scheme [39], which itself is a variant of the Pointcheval and Sanders signature scheme [35]. Indeed, certain checks originally performed using a sigma protocol in Coconut are instead handled in the Groth16 proof in our implementation, reducing the overall verification complexity. We instantiate this scheme with the BLS12-377 curve so that the verification algorithm, which involves pairings, is efficient in the Groth16 proof. More details about the construction are given in [11].

7.2 Performance analysis

We now report on our comparative performance analysis.

NIZK Benchmark. We benchmarked our NIZK, implemented using a Groth16 SNARK, on an Intel i7 @ 2.6 GHz CPU running Ubuntu 22.04. The results are summarized in Table 2. Each value is based on the average of 10 proving and verification runs. A comparison with related protocols is provided in Table 1. The data in this table was gathered from the respective research papers, as the source codes of these protocols were not always publicly available. However, the CPU used for our benchmark has comparable performance to the machines reported in those works: they all use Intel Core i7 CPUs except for Zef and UTT. For Zef and UTT, AWS EC2 instances were used for the benchmark. These two types of instance have similar CPUs (Intel Xeon Platinum 8175 and 8124).

PaxPay has two implementations: with and without the regulatory feature. The benchmark was conducted on both. The regulated version, running on a single core, serves as a reference point. The results indicate that our SNARK verification time is notably short, taking only 5.6 **ms**. As shown in Table 1, this verification time is between 8 and 100 times shorter than the NIZK verification times reported in other works (except for Zcash, which takes approximately the same time but suffers from other major issues discussed in Section 8). The verification time is a key metric since slow verification limits the transaction throughput of the system.

This fast verification comes at the cost of relatively slow proving, requiring nearly 7 **s** to generate a proof for a transfer on a single core. However, with 6 cores, the proving time drops to 2 **s**.

Table 2 SNARK Proving and Verification Times on an Intel i7 2.6 GHz CPU, running Ubuntu 22.04.

	Regulated		Not Regulated	
	1 Core	**6 Cores**	**1 Core**	**6 Cores**
Proving time (ms)	**6959** (± 13)	2001 (± 13)	3080 (± 6)	882 (± 5)
Verification time (ms)	**5.61** (± 0.20)	5.38 (± 0.28)	5.22 (± 0.15)	4.71 (± 0.85)

Latency test and transaction throughput. Apart from Zcash, which is already deployed in real-world use, the only payment systems in the related work to report on their latency and throughput are Zef [6] and UTT [41]. The performance analyses of UTT and Zef were performed on different types of AWS EC2 instances (c5.4xlarge for UTT and m5.8xlarge for Zef). Since the micro-benchmarks show better performance for PaxPay on UTT's EC2 instance than on the one used by Zef, we chose to use the same instance type as in the Zef paper for consistency. [8]

We thus run our PaxPay implementation on multiple m5.8xlarge instances, assigning one validator per instance. For each version of PaxPay (regulated and non-regulated), we conducted two tests: one with each validator running on a single CPU core, and another with each validator using 16 CPU cores. The results are presented in Figures 2a and 2b. We consider that the system support a given throughput if the corresponding transaction latency is smaller than 500 ms. Our regulated implementation achieves the throughput of 100 **tx/s** (transactions per second) in the single-core setup and 925 **tx/s** in the 16-core setup. Under the same conditions, Zef processes 5 **tx/s** and 88 **tx/s**, respectively. UTT processes 235 **tx/s** using 16 cores.[9] For the non-regulated PaxPay implementation, the throughput reaches 115 **tx/s** in the single-core setup and 1200 **tx/s** in the 16-core setup.

These promising results can be attributed to the low verification time of the SNARK presented in the full version of the paper. In the reference setting (regulated implementation with 1-core validators), the verification of the SNARK takes 9 **ms** in our implementation while it takes 142 **ms** in Zef with the same setting [11].

The difference in performance between the regulated and non-regulated versions is due to the additional public parameters in the regulated SNARK. This requires validators to perform more point multiplications on the elliptic curve used in the Groth16 construction.

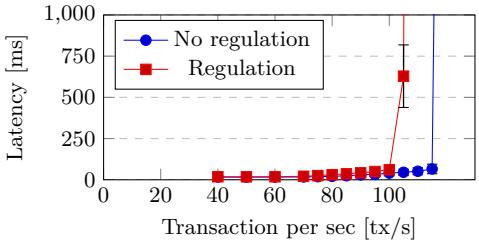
To improve scalability, validators can be *sharded* (run on multiple parallel machines) as in Zef and UTT. The same approach could be applied to our system, though it has not been implemented. A load balancer could efficiently distribute transfer requests among the validators by assigning a specific range of serial numbers to each shard of a validator. Upon receiving a request, the load balancer would determine the appropriate shard to handle it based on these predefined ranges. Our experiments thus show that PaxPay is scalable. The throughput can be improved even further by allocating more computational power to the validators, either by using more CPU cores per validator or by distributing the workload across additional machines.

8 Discussion and comparison with related protocols

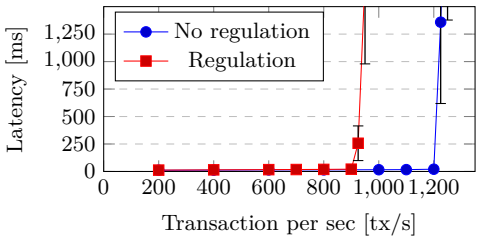
Comparative analysis. Table 1 provides a comparison between PaxPay and the related protocols. Lelantus appears to outperform Monero across all metrics in the table and offers larger anonymity sets. We therefore decided to omit Monero from the table. Similarly, Platypus was not included in the table, as we focus exclusively on decentralized payment systems. Details and explanations regarding the data presented in the table can be found in [11].

[8] We were unable to build and test the UTT code due to dependencies that broke over time. Since the UTT authors chose an EC2 instance type on which PaxPay performs better compared to the m5.8xlarge instance, we consider the results reported in the UTT paper to be suitable for direct comparison.

[9] As stated in Zef [6], each authority is distributed over multiple shards, each running in a single core. The 16-core result is extrapolated from the linear relationship between throughput and the number of shards per validator, as shown in their paper.



(a) Average transaction latency with respect to the transaction throughput. $f = 3, N = 10$. Each validator runs on 1 CPU core.



(b) Average transaction latency with respect to the transaction throughput. $f = 3, N = 10$. Each validator runs on 16 CPU cores.

Figure 2 Latency tests on AWS machines within the same region.

Privacy. As shown in Table 1, PaxPay provides the strongest privacy guarantees, comparable to those provided by Zcash, whereas some protocols offer reduced privacy guarantees. This is especially the case for Zef and PRCash.

To ensure full privacy, it is important to hide coin details when creating them with blind signatures. Yet, when the signatures are verified by validators, some users may still learn information on the payment.[10] UTT circumvents this problem by using a re-randomizable signature scheme (where both the signature and the signed message can be randomized), but this solution increases the workload for verifying transactions. Our solution is to verify the signature within the NIZK, which also decreases the validators workload.

Model assumptions. PaxPay is responsive, meaning it can be implemented in an asynchronous network. Among the related protocols only Zef, UTT, and PARScoin also provide responsiveness. All the four protocols are built on some form of Byzantine consistent broadcast. However, PARScoin requires correct validators (those who follow the protocol) to be honest, ensuring they do not share any information with the adversary. In contrast, PaxPay tolerates correct validators being semi-honest, meaning they can share any information they receive or transmit during the protocol's execution without compromising the system's privacy guarantees.

Regulation. As displayed in Table 1, PaxPay does not support the *Limited held amount per user* feature. PEReDi [37] implements this feature, requiring synchrony between sender and receiver. PARScoin [38] claims a different approach: the sender initiates a transaction to reduce their balance, and the receiver can asynchronously claim funds later, provided their balance remains within the imposed limit. However, this method merely limits the amount a user can hold at a given time, as receivers can bypass it by spending excess funds before claiming the pending funds they have received. In practice, this mechanism has the same effect as setting a limit on the amount that can be spent in a single transaction. We believe that implementing a *Limited held amount per user* feature is particularly challenging in a private and asynchronous payment system. Such a feature would require that transfers atomically change the balances of both the sender and the receiver, to prove that receiver

[10] Suppose that, as in Zef [6], we resort to only using blind signatures. Let a user A creates a coin $\mathbf{c}$ for a user B during a transfer o_1. A requests the validators to sign the blinded coin $\tilde{\mathbf{c}}$, and then sends the aggregated signature to B. B now spends $\mathbf{c}$ during a transfer o_2, and in process reveals $\mathbf{c}$ and the (randomized) signature. Even though validators cannot reveal that the coin spent in o_2 was created in o_1, A knows $\mathbf{c}$ and, thus, can deduce the sender of o_2 and a lower bound on the amount spent during o_2.

balance does not exceed a certain amount after executing the transfer. However, to affect the balance of a user, privacy typically requires this user to interact with another user or the validators, in order to provide some secret data known only to him (such as coin data or account commitments). Yet, asynchrony precludes live protocols from relying on interaction between senders and receivers. Indeed, in an asynchronous setting, the sender who gets no response from the receiver cannot distinguish whether this absence of response is caused by network delays or by the receiver refusing to respond. Implementing *Limited Held Amount per User* seems hard when ensuring both asynchrony and privacy.

PaxPay also avoids *Full Asset Tracing* due to privacy risks. Instead, it allows users to generate privacy-preserving proofs about their transactions. Let us give a real life example. A user can prove that her account received funds from multiple addresses, each transfer being under a certain threshold, demonstrating legitimate income patterns. This proof could demonstrate that the income pattern of a business is legitimate and consistent with regular business operations, precluding money laundering (which might involve receiving a large sum from a single address).

Performance. As mentioned in Section 7.2, PaxPay outperforms all other protocols in terms of transaction throughput. This performance advantage is directly attributed to its fast NIZK verification, which surpasses all existing protocols. The only exception is Zcash, which also benefits from a fast NIZK verification but is constrained by the limitations of its underlying consensus protocol.

Additionally, the system's scalability, that is achieved by increasing the computational power allocated to each validator, further strengthens its practical applicability. The Visa payment system processed an average of $7,388$ transactions per second in 2024 [43], which could be supported by PaxPay by distributing each validator across 8 m5.8xlarge EC2 instances each[11].

As mentioned in Section 7.2, this high throughput comes at the cost of a relatively slow transfer proving, requiring between 2 and 7 seconds depending on the user computational power to prove a transaction. We believe that this delay is acceptable from the user's perspective, especially since multiple payments can be aggregated into a single transfer. As a result, even if a user needs to make numerous payments, she can prove them all within a single transaction, requiring only one proof. It is worth noting that the statement being proven in such a case would be larger than for a proof of a single payment, which would still increase the proving time, though less significantly compared to generating a separate proof for each payment.

Use cases. Our system is versatile and adaptable to various use cases. It can function as a standalone payment system, a CBDC, or a scaling solution for an existing blockchain (a so-called Layer 2). It can be enriched with a Mint and Redeem operations that user would call to put or retrieve money in/from the system. Depending on the use case, Mint can be equipped with a proof showing that the user has the right to mint a new coin. For example, if PaxPay were used as a scaling solution for Bitcoin, the Mint operation would take some SPV (Simplified payment verification) [33] as input to prove to the validators that some value has been locked on Bitcoin. The implementation of the Mint and Redeem operations

[11] This estimation assumes a direct linear relationship between throughput and the computational power of the validators, as shown in Zef [6].

would vary depending on whether the system is deployed as a CBDC or as a Layer 2 solution for a blockchain. The performance of PaxPay (Section 7.2) and its scaling capacity are key factors that enable our system to effectively address these use cases.

Bad user behavior. If a user tries to pass two transactions spending the same input coin, her account might get blocked because she will not get enough signatures for either of the transactions. In this case, consensus is required to unlock the user's account [26, 42].

Future work. Our protocol can be reused with some slight modifications to implement the same functionalities as in Zexe [9], hence not only offering a payment system but *private computation*. We can also considerably improve performance of our system by introducing sharding in the implementation, or improve the usability by moving to a different type of NIZK with a universal trusted setup.

9 Conclusion

In this paper, we consider the problem of fully-private asset transfer (FPAT). We propose PaxPay, an asynchronous FPAT protocol, prove its correctness, detail its implementation in Golang, and analyze its performance. PaxPay leverages succinct non-interactive zero-knowledge proofs and threshold blind signatures. Our system demonstrates significant improvements over existing systems, processing transactions at a high rate, while maintaining full anonymity for the users and responsiveness. PaxPay also ensures regulatory enforcement, including some features that no other private payment system has managed to provide so far. The flexibility offered by succinct NIZKs opens avenues for incorporating other regulatory compliance features with minimal impact on the system's performance. These elements make PaxPay suitable for a wide range of applications, from a scaling solution of blockchain to a standalone private payment system that can be used as a technical layer for CBDCs.

References

1 Timothé Albouy, Emmanuelle Anceaume, Davide Frey, Mathieu Gestin, Arthur Rauch, Michel Raynal, and François Taïani. Asynchronous BFT asset transfer: Quasi-anonymous, light, and consensus-free, 2024. doi:10.48550/arXiv.2405.18072.

2 Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, 2016. doi:10.1007/978-3-662-53887-6_7.

3 Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Björn Tackmann. Privacy-preserving auditable token payments in a permissioned blockchain system. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, AFT '20, pages 255–267, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3419614.3423259.

4 European Central Bank. Report on a digital euro, 2020. URL: https://www.ecb.europa.eu/pub/pdf/other/Report_on_a_digital_euro~4d7268b458.en.pdf.

5 Mathieu Baudet, George Danezis, and Alberto Sonnino. Fastpay: High-performance byzantine fault tolerant settlement. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 163–177, 2020. doi:10.1145/3419614.3423249.

6 Mathieu Baudet, Alberto Sonnino, Mahimna Kelkar, and George Danezis. Zef: Low-latency, scalable, private payments. In Bart P. Knijnenburg and Panos Papadimitratos, editors, *Proceedings of the 22nd Workshop on Privacy in the Electronic Society, WPES 2023, Copenhagen, Denmark, 26 November 2023*, pages 1–16. ACM, 2023. doi:10.1145/3603216.3624952.

7 Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014. doi:10.1109/SP.2014.36.

8 Gautam Botrel, Thomas Piellard, Youssef El Housni, Ivo Kubjas, and Arya Tabaie. Consensys/gnark: v0.11.0, September 2024.

9 Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. *IACR Cryptol. ePrint Arch.*, page 962, 2018. URL: https://eprint.iacr.org/2018/962.

10 Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987. doi:10.1016/0890-5401(87)90054-X.

11 Maxence Brugeres, Victor Languille, Petr Kuznetsov, and Hamza Zarfaoui. Fast, private and regulated payments in asynchronous networks. Cryptology ePrint Archive, Paper 2025/098, 2025. URL: https://eprint.iacr.org/2025/098.

12 Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM (JACM)*, 43(4):685–722, 1996. doi:10.1145/234533.234549.

13 Panagiotis Chatzigiannis and Foteini Baldimtsi. Miniledger: Compact-sized anonymous and auditable distributed payments. In *European Symposium on Research in Computer Security*, pages 407–429. Springer, 2021. doi:10.1007/978-3-030-88418-5_20.

14 Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xygkis. Online payments by merely broadcasting messages. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020*, pages 26–38. IEEE, 2020. doi:10.1109/DSN48063.2020.00023.

15 Ronald Cramer. *Modular design of secure yet practical cryptographic protocols*. PhD thesis, U. of Amsterdam, 1996.

16 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985. doi:10.1145/2455.214112.

17 Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. doi:10.1137/0212045.

18 Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony (preliminary version). In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 103–118. ACM, 1984. URL: https://dl.acm.org/citation.cfm?id=1599406.

19 Electric Coin Company and Zcash Contributors. Zcash: Protocol and reference implementation. https://github.com/zcash/zcash, 2023. Accessed: 01/09/2024.

20 David Evans, Vladimir Kolesnikov, and Mike Rosulek. *A Pragmatic Introduction to Secure Multi-Party Computation*. Now Publishers Inc, 2018. doi:10.1561/3300000019.

21 Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. Cryptology ePrint Archive, Paper 2018/990, 2018. URL: https://eprint.iacr.org/2018/990.

22 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. In Ronald Fagin and Philip A. Bernstein, editors, *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23, 1983, Colony Square Hotel, Atlanta, Georgia, USA*, pages 1–7. ACM, 1983. doi:10.1145/588058.588060.

23 Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, volume 9603 of *Lecture Notes in Computer Science*, pages 81–98. Springer, 2016. doi:10.1007/978-3-662-54970-4_5.

24 Oded Goldreich. *Foundations of Cryptography*, volume 1. Cambridge University Press, 2001. doi:10.1017/CBO9780511546891.

25 Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016. doi:10.1007/978-3-662-49896-5_11.

26 Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. The consensus number of a cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19. ACM, July 2019. doi:10.1145/3293611.3331589.

27 Saurabh Gupta. A Non-Consensus Based Decentralized Financial Transaction Processing Model with Support for Efficient Auditing. Master's thesis, Arizona State University, USA, 2016.

28 Youssef El Housni and Aurore Guillevic. Optimized and secure pairing-friendly elliptic curves suitable for one layer proof composition. Cryptology ePrint Archive, Paper 2020/351, 2020. URL: https://eprint.iacr.org/2020/351.

29 Aram Jivanyan. Lelantus: A new design for anonymous and confidential cryptocurrencies. Cryptology ePrint Archive, Paper 2019/373, 2019. URL: https://eprint.iacr.org/2019/373.

30 Aram Jivanyan and Aaron Feickert. Lelantus spark: Secure and flexible private transactions. Cryptology ePrint Archive, Paper 2021/1173, 2021. doi:10.1007/978-3-031-32415-4_28.

31 Petr Kuznetsov, Yvonne-Anne Pignolet, Pavel Ponomarev, and Andrei Tonkikh. Permissionless and asynchronous asset transfer. *Distributed Comput.*, 36(3):349–371, 2023. doi:10.1007/S00446-023-00449-X.

32 Dahlia Malkhi and Michael K. Reiter. A high-throughput secure reliable multicast protocol. *Journal of Computer Security*, 5(2):113–128, 1997. doi:10.3233/JCS-1997-5203.

33 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: http://bitcoin.org/bitcoin.pdf.

34 Shen Noether, Adam Mackenzie, and The Monero Research Lab. Ring Confidential Transactions. *Ledger*, 1:1–18, December 2016. doi:10.5195/ledger.2016.34.

35 David Pointcheval and Olivier Sanders. Short randomizable signatures. Cryptology ePrint Archive, Paper 2015/525, 2015. URL: https://eprint.iacr.org/2015/525.

36 Guillaume Quispe, Pierre Jouvelot, and Gérard Memmi. Nickpay, an auditable, privacy-preserving, nickname-based payment system. *CoRR*, abs/2503.19872, 2025. doi:10.48550/arXiv.2503.19872.

37 Amirreza Sarencheh, Aggelos Kiayias, and Markulf Kohlweiss. PEReDi: Privacy-enhanced, regulated and distributed central bank digital currencies. Cryptology ePrint Archive, Paper 2022/974, 2022. URL: https://eprint.iacr.org/2022/974.

38 Amirreza Sarencheh, Aggelos Kiayias, and Markulf Kohlweiss. Parscoin: A privacy-preserving, auditable, and regulation-friendly stablecoin. In *Cryptology and Network Security: 23rd International Conference, CANS 2024, Cambridge, UK, September 24–27, 2024, Proceedings, Part I*, pages 289–313, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-981-97-8013-6_13.

39 Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. *arXiv preprint*, 2018. arXiv:1802.07344.

40 Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *1997 IEEE Symposium on Security and Privacy, May 4-7, 1997, Oakland, CA, USA*, pages 44–54. IEEE Computer Society, 1997. doi:10.1109/SECPRI.1997.601314.

41 Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. UTT: decentralized ecash with accountable privacy. *IACR Cryptol. ePrint Arch.*, 2022. URL: https://eprint.iacr.org/2022/452.

42 Andrei Tonkikh, Pavel Ponomarev, Petr Kuznetsov, and Yvonne-Anne Pignolet. Cryptoconcurrency: (almost) consensusless asset transfer with shared accounts. In *CCS*, pages 1556–1570. ACM, 2023. doi:10.1145/3576915.3616587.

43 Visa. Visa annual report 2024, 2024. URL: https://s29.q4cdn.com/385744025/files/doc_downloads/2024/Visa-Fiscal-2024-Annual-Report.pdf.

44 Karl Wüst, Kari Kostiainen, Vedran Capkun, and Srdjan Capkun. PRCash: Fast, private and regulated transactions for digital currencies. Cryptology ePrint Archive, Paper 2018/412, 2018. URL: https://eprint.iacr.org/2018/412.

45 Karl Wüst, Kari Kostiainen, Noah Delius, and Srdjan Capkun. Platypus: A central bank digital currency with unlinkable transactions and privacy preserving regulation. Cryptology ePrint Archive, Paper 2021/1443, 2021. doi:10.1145/3548606.3560617.

Proxying Is Enough: Security of Proxying in TLS Oracles and AEAD Context Unforgeability

Zhongtang Luo
Purdue University, United States

Yanxue Jia
Purdue University, United States

Yaobin Shen
Xiamen University, China

Aniket Kate
Purdue University, United States
Supra Research, United States

Abstract

TLS allows a client to securely obtain data from a server, but does not allow the client to offer the data provenance to an external node. TLS oracle protocols are used to solve the problem. Specifically, the verifier node, as an external node, is convinced that the data is indeed coming from a pre-defined TLS server, while remaining unable to access the client's credentials (e.g., password). Previous TLS oracle protocols such as DECO (CCS 2020) enforced the communication pattern of server-client-verifier and utilized a novel three-party handshake process during TLS to ensure data integrity against potential tempering by the client. However, this approach introduces a significant performance penalty on the client and the verifier. Most recently, some works have proposed to reduce the overhead by putting the verifier (as a proxy) between the server and the client such that the correct TLS transcript is available to the verifier. Nevertheless, these works still rely on heavy two-party secure computations or zero-knowledge proofs. In this work, we push the proxy model to the extreme, where the verifier only needs to forward messages without performing any other heavy computational operations when only the credentials should be protected and the data retrieved from the server could be open to the verifier. Surprisingly, we prove that the thorough proxy model is enough to guarantee security in some common scenarios, allowing a saving of 60–90% in running time under common scenarios.

We first formalize the proxy-based Oracle protocol and functionality that allows the verifier to directly proxy client-server TLS communication, without entering a three-party handshake or interfering with the connection in any way. We then show that for common TLS-based higher-level protocols such as HTTPS, data integrity to the verifier proxy is ensured by the variable padding built into the HTTP protocol semantics. On the other hand, if a TLS-based protocol comes without variable padding, we demonstrate that data integrity cannot be guaranteed. In this context, we then study the case where the TLS response is pre-determined and cannot be tampered with during the connection. We propose the concept of context unforgeability and show that data integrity can also be guaranteed as long as the underlying Authenticated Encryption with Associated Data (AEAD) satisfies context unforgeability. We further show that ChaCha20-Poly1305 satisfies the concept while AES-GCM does not.

2012 ACM Subject Classification Security and privacy → Block and stream ciphers; Security and privacy → Security protocols

Keywords and phrases Oracle, TLS, AEAD, Key Commitment

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.4

Related Version *Full Version*: `https://ia.cr/2024/733`

Funding This work is supported partially by the National Science Foundation under grant CNS-1846316 and a gift from Supra Labs.
Yaobin Shen: Yaobin Shen is supported by the National Key Research and Development Program of China (2024YFB4504800), and the National Natural Science Foundation of China (62402404).

© Zhongtang Luo, Yanxue Jia, Yaobin Shen, and Aniket Kate;
licensed under Creative Commons License CC-BY 4.0

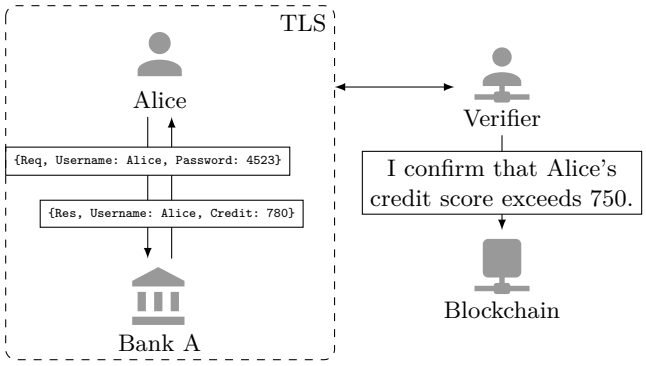


Figure 1 A use-case example. Alice queries Bank A about her credit score via TLS. The verifier/oracle attests to the blockchain about Alice's credit score, by intervening in the TLS communication.

Acknowledgements We would like to thank Madhavan Malolan and Kirill Kutsenok from the Reclaim Protocol for helpful discussions and suggestions.

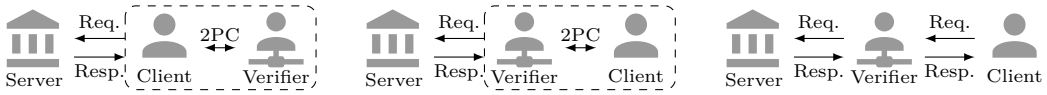
1 Introduction

Transport Layer Security (TLS) [24, 25] has been widely used to establish a secure communication channel between a client and a server, enabling secure data access (we refer to such data as TLS-protected data). However, TLS relies on symmetric encryption and both the client and the server can obtain the key. This enables the client to build a ciphertext for any data and falsely claim that it comes from the server. Thus, TLS does not allow the client to prove the *provenance* and *integrity* of the data to third parties, limiting the propagation of data. In particular, many decentralized applications on the blockchain need to access the TLS-protected data as third parties. Due to the TLS limitation, we cannot rely on each client to feed data into the blockchain. Therefore, a type of service called *Oracle* [5, 7] was proposed to feed TLS-protected data into blockchains while guaranteeing the provenance and integrity of data.

A straightforward way to implement the service is to allow a verifier to act as the client to access data from the server. Unfortunately, this approach compromises the client's security and privacy, as the verifier needs to obtain the client's credentials. Therefore, a desirable way is to allow the verifier to participate in the communication between the client and the server to prevent the client from modifying the data, but *without* obtaining additional private information.

In Figure 1, we give a use case to explain how the oracle works: There is an application on the blockchain that activates a rental contract if "Alice's credit score exceeds 750". Alice uses her password to request the credit score from bank A, through a TLS connection. The verifier participates in the TLS communication without learning request and response messages but can check if Alice's credit score in bank A exceeds 750. If so, the verifier submits the confirmation information to the blockchain, and then the rental contract is activated.

For some applications, only request should be hidden from the verifier, and the response could be open to the verifier. As in the above example, Alice may be willing to open `{Res, Username: Alice, Credit: 780}` to the verifier; however, she does not want to reveal the password in the request to the verifier, as the verifier can obtain more sensitive information using the password. On the other hand, there are also some applications where protecting the



(a) 2PC-based TLS Oracle [33, 31, 9, 6].

(b) Proxy-but-2PC TLS Oracle [33, 16].

(c) Proxy-based TLS Oracle (this work).

Figure 2 Design frameworks in the previous works use a 2PC protocol to prevent the client from forging a message. In contrast, in our work, the client and the server interact with each other as in TLS, but the messages between them are forwarded by the verifier.

response is also necessary, e.g., the response includes bank statements. An oracle protocol designed for the former scenario can be adapted to the latter by incorporating zero-knowledge proofs, allowing the client to prove that the data in response satisfies a specific predicate without revealing the data to the verifier.

As bringing clients' private data securely and selectively to blockchains is a problem of immense interest, a few academic [33, 32, 26, 16, 31, 9] and industrial [4, 22] efforts in this direction are already available. Earlier works typically involve modifying the TLS server-side code [26] or utilizing trusted hardware [32]. While the approaches remain an interesting theoretical probability, typical servers are reluctant to use modified TLS, and the usage of trusted hardware introduces additional trusted entities. Hence both approaches see limited use.

(Proxy-but-)2PC-based TLS Oracle. In a seminal work, Zhang et al. [33] proposed DECO, which overcomes this practicality barrier without making any changes to the TLS server-side code and using any trusted hardware. The core reason why the client in TLS can forge the TLS data is that the symmetric encryption and authentication key generated in the TLS session is shared by the client and the server (please see Section 2 for more details). DECO [33], essentially splits the role of the client in typical TLS into two parties, as shown in Figure 2a, so that the client cannot learn the entire symmetric encryption and authentication key. In particular, the client and the verifier collaboratively communicate with the server using secure two-party computation (2PC). Moreover, while the client and the verifier collaborate, it is only the client that communicates with the server as in TLS. We call this design as "2PC-based TLS oracle" protocol. Later, the subsequent works [31, 9, 6] also follow the model.

In addition, Zhang et al. [33] also discussed another design framework, where the verifier acts as a proxy to communicate with the sender but the verifier still needs to perform 2PC with the client, as shown in Figure 2b. We call this design as "proxy-but-2PC based TLS oracle" protocol. Xie et al. [31] proposed a solution in this model that utilizes a garble-then-prove scheme.

We can see that the previous works all rely on heavy 2PC between the client and the verifier. A natural question is *"Is there a secure TLS oracle solution that can totally avoid heavy 2PC between the client and the verifier?"*

Proxy-based TLS Oracle. To answer this question, in this work, we consider a pure proxy-based TLS oracle as shown in Figure 2c, where the verifier only needs to forward the messages generated by the client and the server, without additionally introducing any heavy computation, which directly implies *better performance*. In addition, compared with the (proxy-but-)2PC-based TLS oracles, the proxy-based oracle enables *better compatibility* across different TLS versions, as the verifier only needs to forward messages. Moreover, the proxy-based oracle not only does not modify server-side code but also achieves *client-side non-modification*.

Given that AEAD (Authenticated Encryption with Associated Data) is the core building block of TLS, we investigate the security of two AEAD schemes, AES-GCM and ChaCha20-Poly1305, which are overwhelmingly used in TLS 1.2/1.3 with an adoption rate of over 99% [29]. Based on the security of AEAD, we systematically conduct an investigation that outlines the boundary between security and insecurity of the proxy-based TLS oracle protocol. Surprisingly, we find that the proxy-based TLS oracle can be securely used in two prevalent situations, HTTPS (see Section 6) and prefixed relevant data (see Section 7).

1.1 Contributions

Formalization. We provide an oracle ideal functionality and formalize the notion of the above proxy-based TLS oracle protocol that allows the verifier to directly proxy client-server TLS communication. The definitions allow us to analyze its security in this paper but can also be applied to different versions of TLS and a comparison between oracle protocols. We consider them to be of independent interest.

Impossibility. Based on the above definitions, we also prove that if the underlying AEAD scheme satisfies key commitment, the proxy-based TLS oracle protocol can securely realize the oracle ideal functionality in Figure 11. On the other hand, in Section 5, we find that without the key commitment, an adversarial client can potentially forge a message to the verifier.

Therefore, we know that the key commitment is *sufficient* and *necessary* to guarantee the security of the proxy-based TLS oracle protocol. However, the AEAD schemes currently used in TLS do not satisfy the key commitment, according to previous research [10, 1], which means that the proxy-based TLS oracle protocol is not secure without specific constraints.

Possibility. Fortunately, we observe that the real-world application scenarios provide some practical constraints such that the proxy-based TLS oracle can be used securely.

HTTPS. In Section 6, we define a new notion called "variable padding" in Definition 6, and prove that when the plaintext contains a sufficiently long variable padding, AES-GCM and ChaCha20-Poly1305 satisfy the key commitment, which means that the proxy-based TLS oracle is secure in this case. Notably, we show that the TLS-based HTTPS protocol, which is used in over 85% of all web pages [23], contains a sufficiently long variable padding, and thus the proxy-based TLS oracle protocol can be securely used on it.

Prefixed relevant data. In Section 7, we focus on the constraint where the response plaintext is prefixed and not changeable by the client once the connection is established. For example, in practice, the client cannot arbitrarily modify his age, salary, social security number, etc. To analyze the security in this case, we propose a new cryptographic property, *context unforgeability*, and prove that AES-GCM is not secure yet, ChaCha20-Poly1305 is secure with respect to context unforgeability. In addition, we prove that as long as the TLS uses a context-unforgeable AEAD scheme, the proxy-based TLS oracle is secure.

New AEAD Security Property. Besides being used for analyzing the security of the proxy-based TLS oracle, the newly proposed context unforgeability precisely bridges the gap in the security analysis of AEAD, and thus may be of independent interest.

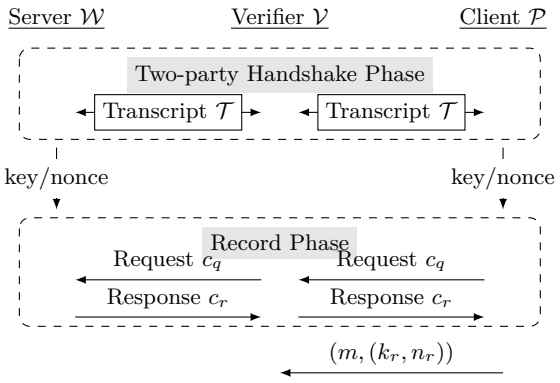


Figure 3 Our Proxy-based Oracle Protocol. The server and the client perform as in TLS, but the messages are forwarded by the verifier. Finally, the client sends a message m, a key/nonce pair (k_r, n_r) to the verifier, such that the verifier can verify that c_r could be decrypted to the message m using the key/nonce pair (k_r, n_r). When the client does not want to open the entire message m to the verifier, the client can use a zero-knowledge proof protocol to generate a proof π to the verifier to convince her that there is a message m that is decrypted from c_r and satisfies a predetermined requirement. The proof for data is orthogonal to our work. The detailed specification is defined in Π_{Proxy} (see Figure 7).

2 Technical Overview

In practice, a client $\mathcal{P}$ can retrieve a message m from a server $\mathcal{W}$ via TLS. An Oracle protocol aims to allow the client $\mathcal{P}$ to prove to a verifier $\mathcal{V}$ that the message m is indeed obtained from the server $\mathcal{W}$ and satisfies a predetermined requirement (e.g., age is greater than 18). Typically, there are two cases: one is that the message m can be open to the verifier, called "public mode", and the other is that the client does not want to leak the entire m to the verifier, called "private mode". The protocol for the public mode can be transformed to that for the private mode by using a zero-knowledge proof with the opening information in the public mode as witness. Therefore, for simplicity, here we focus on the public mode, and we also show the private mode in Section 4.

In this work, we assume that server $\mathcal{W}$ is always honest, while client $\mathcal{P}$ and verifier $\mathcal{V}$ can be malicious. The key idea of our work is to use verifier $\mathcal{V}$ as a proxy to prevent client $\mathcal{P}$ from tampering with the message m. Therefore, we also assume that verifier $\mathcal{V}$ reliably connects to server $\mathcal{W}$, i.e., verifier $\mathcal{V}$ can ensure that she indeed connects with server $\mathcal{W}$ and the communication messages cannot be tampered with by others (including client $\mathcal{P}$). The assumption has been accepted by other proxy-setting systems [28]. Based on the above assumptions, we design a proxy-based oracle protocol as illustrated in Figure 3 and obtain a series of theoretical results summarized in Figure 4. Next, we will detail how we obtain these results.

TLS relies on authenticated encryption with associated data (AEAD), a symmetric-key primitive, to achieve integrity and confidentiality of exchanged messages. As shown in Figure 3 (ignoring the verifier), TLS consists of a two-party handshake phase and a record phase. After the two-party handshake phase, the server $\mathcal{W}$ and the client $\mathcal{P}$ both obtain a key/nonce pair[1] of AEAD to encrypt and decrypt the server's messages. Then, in the record

[1] Here, we ignore the other key/nonce pair for encrypting and decrypting the client's messages, as this work mainly focuses on preventing the client from forging the server's messages.

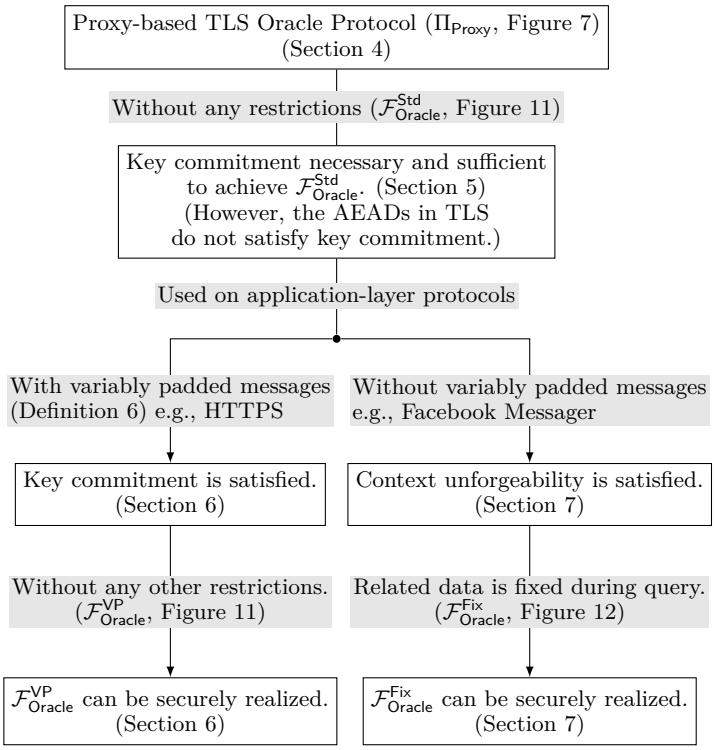


Figure 4 Main contributions. We first give a proxy-based TLS oracle protocol Π_{Proxy} and a functionality $\mathcal{F}^{\mathsf{Std}}_{\mathsf{Oracle}}$ without any restrictions. Then, we observe that key commitment is necessary and sufficient to achieve $\mathcal{F}^{\mathsf{Std}}_{\mathsf{Oracle}}$; however, the AEADs used in TLS are not key committing in general. Fortunately, real-world applications come with different restrictions on the plaintext response:

1. Plaintexts are variably padded (e.g., HTTPS), and we denote the functionality as $\mathcal{F}^{\mathsf{VP}}_{\mathsf{Oracle}}$.
2. Relevant data is immutable during the query (e.g., age), and we denote the functionality as $\mathcal{F}^{\mathsf{Fix}}_{\mathsf{Oracle}}$.

When applying the AEADs in TLS to variably padded messages when the padding is known by the verifier in advance, we show that key commitment is satisfied, and thus $\mathcal{F}^{\mathsf{VP}}_{\mathsf{Oracle}}$ can be achieved. In addition, we prove that regardless of whether the plaintexts are variably padded or not, ChaCha20-Poly1305 in TLS satisfy a newly proposed property, *context unforgeability*, which is sufficient to achieve $\mathcal{F}^{\mathsf{Fix}}_{\mathsf{Oracle}}$.

phase, the server $\mathcal{W}$ and the client $\mathcal{P}$ communicate with each other using the key/nonce pair generated by the handshake phase. We can see that the client $\mathcal{P}$ also holds the key/nonce pair of AEAD. Therefore, the client $\mathcal{P}$ can use the key/nonce pair to encrypt another distinct message m' to generate a ciphertext c'_r, and falsely claim that c'_r is retrieved from server $\mathcal{W}$. Note that here we discuss the original TLS without the verifier as a proxy.

To address the problem, previous works [33, 9, 31, 6, 16] prevent client $\mathcal{P}$ from obtaining the whole key by splitting the key into two parts, each of which is obtained by client $\mathcal{P}$ and verifier $\mathcal{V}$ respectively. Specifically, client $\mathcal{P}$ and verifier $\mathcal{V}$ collectively fulfill the role of the client in TLS through a secure two-party computation (2PC) protocol. However, compared with the original TLS protocol, introducing a 2PC protocol incurs significant extra costs, which raises a question: *Is it possible to avoid 2PC protocols?*

Preliminary Attempt. We observe that in the attack above, client $\mathcal{P}$ uses the AEAD key/nonce pair to generate a new ciphertext $c'_r \neq c_r$, where c_r is the original ciphertext from server $\mathcal{W}$. The essence of previous works [33, 9, 31, 6, 16] is to ensure that client $\mathcal{P}$ cannot

obtain the complete AEAD key before he commits the message. Our preliminary attempt is to ensure that verifier $\mathcal{V}$ can directly obtain the ciphertext c_r from server $\mathcal{W}$, such that verifier $\mathcal{V}$ can recognize any modifications on the ciphertext c_r. To this end, we treat verifier $\mathcal{V}$ as a proxy that forwards and records the messages between server $\mathcal{W}$ and client $\mathcal{P}$, as shown in Figure 3. In this way, the ciphertext c_r cannot be modified by client $\mathcal{P}$. Moreover, verifier $\mathcal{V}$ records the transcript $\mathcal{T}$ generated during the handshake phase. Therefore, client $\mathcal{P}$ can provide $(m, (k_r, n_r))$ to verifier $\mathcal{V}$ to prove the following three statements: he holds a key/nonce pair (k_r, n_r) that can decrypt c_r to a message m; the message m satisfies a preset requirement; the key/nonce pair (k_r, n_r) is derived from the transcript $\mathcal{T}$.

Obviously, once obtaining $(m, (k_r, n_r))$, the verifier can verify the first two statements directly[2]. However, the verifier cannot verify the third statement, since it involves private information (namely, the secret used in Diffie-Hellman Handshake) held by the client (please see Figure 6 for more details). Note that if the client provide the private information used in Diffie-Hellman handshake process to the verifier, the verifier can know all the keys and nonces, and thus the verifier can decrypt the request ciphertext (c_q in Figure 3) to learn the credential.

Using a zero-knowledge proof to prove the third statement is a straightforward way. However, the hash function used in TLS 1.2/1.3 to derive key/nonce is SHA256, which is not SNARK (Succinct Non-interactive Argument Knowledge)-friendly [13]. A concurrent and independent work by Ernstberger [12] have tried to reduce the cost for proving the statement. However, their results showed that the proof for this statement still occupies the cost 90% in the public mode and the cost 60% in the private mode (see Table 1). Therefore, to further reduce the overhead, we delve deeper into the potential of *eliminating the proof that the key/nonce pair is derived from the transcript $\mathcal{T}$ (i.e., the above statement 3).*

Eliminating the Proof of Key Origin. As shown by the previous research [10, 1], the AEAD schemes used by TLS do not satisfy the key commitment property (see Lemma 3); an adversary can efficiently construct a ciphertext c and two distinct key/nonce pairs (k_1, n_1) and (k_2, n_2), such that c can be decrypted to two distinct messages m_1 and m_2 by using (k_1, n_1) and (k_2, n_2) respectively. At first glance, in our proxy-based oracle protocol, client $\mathcal{P}$ cannot launch the above attack, since client $\mathcal{P}$ seems unable to construct the ciphertext sent by server $\mathcal{W}$. However, we observe this is not true, since client $\mathcal{P}$ obtains the key/nonce pair (k, n) as soon as completing the handshake phase. Then, client $\mathcal{P}$ can construct a ciphertext c such that it can be decrypted into m and m' under key/nonce pair (k, n) and another key/nonce pair (k', n') respectively. Moreover, m' meets the predetermined requirement, while m does not. Subsequently, the client can change the record in server $\mathcal{W}$ to m through external interactions, so that server $\mathcal{W}$ produces and sends the ciphertext c. Finally, client $\mathcal{P}$ can use (k', n') to convince verifier $\mathcal{V}$ that m' satisfying the preset requirement is the corresponding plaintext. For example, if the message is about bank balance, client $\mathcal{P}$ can adjust his balance to $m = 10$, and then prove that balance $m' = 1000$ by using (k', n') (see Section 5).

[2] Given that HTTP is a stateless protocol, to avoid using the same key the verifier obtained for future rounds of interaction, the client can terminate the current connection with the server and handshake again to restart one. For other protocols, we note that TLS 1.3 supports a key update mechanism (see Figure 6) that allows us to refresh the key/nonce pair for subsequent communications without needing another handshake.

With Variable Padding. According to the above analysis, we know that key commitment is necessary for a secure proxy-based oracle protocol without proving that the key is derived from a given transcript. Fortunately, we observe that if the plaintext is well-formatted as seen in HTTPS (or more formally, has a variable padding), the adversary cannot break the key commitment. Intuitively, even if the adversary can generate (c, k, n, m) and (c, k', n', m'), the probability that m and m' are both well-formatted is negligible (see Section 6).

In practice, HTTPS (using TLS on the application layer protocol HTTP) satisfies the above condition, since HTTP headers are variably padded [20]. Therefore, our proxy-based oracle protocol without *proving the key origin* and *2PC protocols* is secure with HTTPS. While HTTPS is one of the most popular protocols and thus the current results are already sufficient to address a wide range of application scenarios, we also discuss if our proxy-based oracle protocol can be securely used for other protocols whose messages are not variably padded.

Without Variable Padding. In the above attack, we assume that client $\mathcal{P}$ can arbitrarily modify the record in server $\mathcal{W}$. However, in some scenarios (e.g., ages, stock prices and insurance numbers), client $\mathcal{P}$ is not able to perform modifications. In these cases, the AEAD used in TLS only needs to satisfy a weaker security property: given a ciphertext c, a key/nonce pair (k, n) and the corresponding message m, the adversary cannot construct another key/nonce pair (k', n') such that c can be decrypted to another message m'. In Section 7, we formally define this security property as *context unforgeability (CFY-security)* and systematically investigate if the AEADs (including AES-GCM and ChaCha20-Poly1305) used in TLS satisfy this new security property. We prove that AES-GCM does not satisfy the context unforgeability, whereas ChaCha20-Poly1305 does.

Hierarchical Security of AEAD. Besides designing an efficient proxy-based oracle protocol, our newly proposed security property, context unforgeability, bridges the gap in the security analysis of AEAD, as shown in Figure 5. Interestingly, combined with the previous properties, context commitment[3] [10, 15, 18] and context undiscoverability [18], the hierarchical security of AEAD precisely corresponds to the hierarchical security of hash functions.

3 Preliminary

We briefly outline the relevant background knowledge on TLS, the definition and security properties of Authenticated Encryption with Associated Data (AEAD).

Notations. We use $|m|$ to denote the bit-length of a string m, and $m[i:j]$ to represent its substring starting at i (0-based) with length $(j-i)$, and $m||n$ to denote the concatenation of strings m and n. We represent the set size as $|S|$.

3.1 TLS

Transport Layer Security (TLS) is a family of communication protocols designed to provide end-to-end security over a computer network. Its most prominent use remains to be HTTPS, the web-browsing protocol that sees day-to-day use. TLS 1.3 is the latest protocol in the TLS family, defined in August 2018 [24].

[3] The works [10, 15] first proposed the key commitment problem. Then Bellare et al. [2] extended to committing nonce and associated data, and Menda et al. [18] summarized them as context commitment.

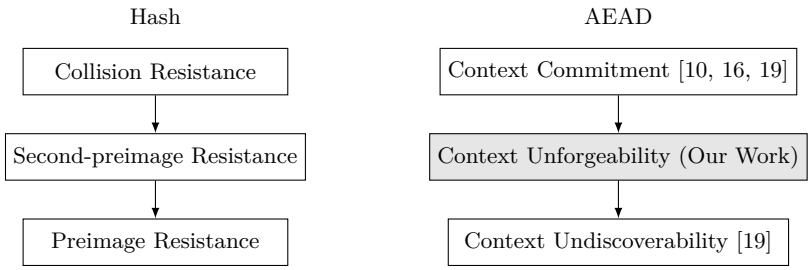


Figure 5 Hierarchical Security of AEAD. Both hash and AEAD can be abstracted as a map: $x \to y$; in AEAD, x includes key k, nonce n, and associated data A, and y refers to ciphertext c. The first level (collision resistance and context commitment) refers to the fact that an adversary cannot efficiently compute y and two different x and x' such that x and x' map to y. The second level (second-preimage resistance and context unforgeability) refers to the fact that given x and y where $x \to y$, an adversary cannot efficiently compute $x' \neq x$ such that x' also maps to y. The third level (preimage resistance and context undiscoverability) refers to that given y, an adversary cannot efficiently compute x such that x maps to y.

There are two main protocols in TLS. The handshake protocol negotiates a symmetric key to be used in the record protocol. The record protocol manages the transmission of messages, using an authenticated encryption with associated data (AEAD) cipher suite to ensure confidentiality and integrity. An overview of TLS is available in Figure 6.

Authenticated Encryption with Associated Data. Nonce-based authenticated encryption with associated data (AEAD) schemes [17] are employed in TLS to ensure data confidentiality and integrity. An AEAD scheme consists of the following four algorithms (AEAD.Setup, AEAD.Gen, AEAD.Enc, AEAD.Dec). We refer to Full Version's Appendix A for a full definition.

Cipher Suites in TLS. TLS supports a handful number of cipher suites [24], with AES-GCM and ChaCha20-Poly1305 being the most popular and enabled in OpenSSL by default [30]. Notably, all cipher suites in TLS adopt a block-based construction – the plaintext and the associated data are packed into a series of fixed-size blocks $(p_1, p_2, \ldots, p_n)$ and $(a_1, a_2, \ldots, a_m)$, which is then processed by the algorithm to produce the ciphertext.

AES-GCM. AES-GCM is the most widely used cipher suite in TLS and the only cipher suite that must be implemented by every application under the specification. In AES-GCM, all the computation is done over the field $GF(2^{128})$. The ciphertext is obtained by $c_i = p_i + E_k(n+i)$, where E is the AES block cipher.

AES-GCM also ensures authenticity by using an authentication tag $t = E_k(n) + \sum_{i=1}^{m+n+1} s_i E_k(0)^{m+n+2-i}$. where $s = (a, c, m \| n)$ is a concatenation of a, c and their length. For further information, we refer the reader to the specification for clarity [11].

ChaCha20-Poly1305. ChaCha20-Poly1305 is an alternative to AES-GCM. In ChaCha20-Poly1305, encryption is done similarly to AES-GCM: $c_i = p_i + H_k(n+i)$, where H is the ChaCha20 stream cipher.

[4] We note that there is a difference between the IV in the TLS protocol and the nonce in the AEAD scheme. In both AES-GCM and ChaCha20-Poly1305 of the RFC specification [17], IV is limited to 96 bits, while the nonce used in the encryption is derived from IV by padding and can be longer.

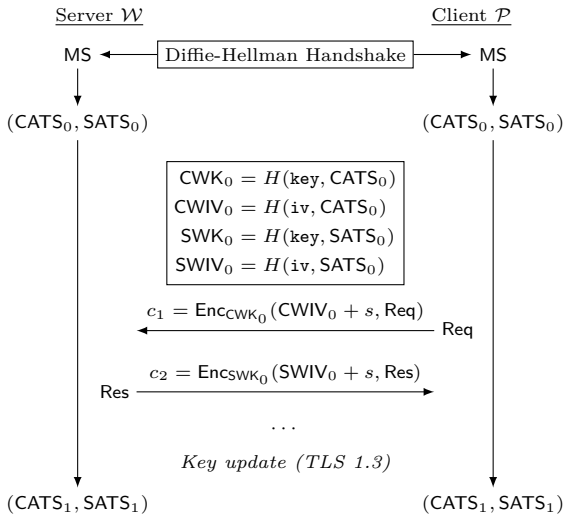


Figure 6 An overview of a TLS session between a client and a server. After establishing a master secret MS with a Diffie-Hellman handshake in the handshake protocol, the parties derive the 0-th client/server application secret $(\mathsf{CATS}_0, \mathsf{SATS}_0)$. They then derive the client write key CWK, the client write IV[4] CWIV, the server write key SWK, and the server write IV SWIV. The values are used in the communication, together with the sequence number s to ensure the uniqueness of each IV. Additionally, in TLS 1.3, each party can also trigger a key update event that refreshes the application secret.

However, the authentication tag computation in ChaCha20 is different from AES-GCM due to the usage of Poly1305. First, two 128-bit variables $(r, s) = H_k(n)[0:256]$ are sampled from the stream cipher. Then, the authentication tag $t = s + \sum_{i=1}^{m+n+1} s_i r^{m+n+2-i}$ where $s = (a, c, m \| n \| 0^8)$ is the concatenation of associated data, ciphertext and their length as in AES-GCM. The computation of t is done over $GF(2^{130} - 5)$, then truncated to 128 bits. For details, we refer the reader to the RFC specification [21].

Key Update. TLS 1.3, the latest TLS version, also supports an operation known as a key update. The operation allows any party to refresh the secret (i.e., CATS_i and/or SATS_i in Figure 6) used on either or both sides. The new secret is generated based on a hash of the old secret, and the new keys and IVs are derived from the new secrets based on the same rule as shown in Figure 6.

3.2 Context Attacks of AEAD

In the proxying oracle protocol, we need to discuss the possibility that an adversarial prover deceives the verifier by providing a symmetric key that does not correspond to the one in the handshake yet still decrypts the ciphertext. This corresponds to the concept of context discovery and commitment attacks, first summarized by Menda et al. [18]. Here we outline a couple of specific definitions that will be useful in our cases. We use † to denote a specification of the more general case discussed by Menda et al.

Context Discovery Attack. A context discovery (CDY) attack refers to the adversary's capability to come up with some part of the context (i.e. a key and/or a nonce) that decrypts the given ciphertext without error, although not necessarily to the original plaintext. Formally:

▶ **Definition 1** (Context Discovery). *Fix some AEAD parameter pp and a corresponding AEAD oracle* Π*. The game* $\mathsf{CDY}^\dagger$ *is defined as:*

1. *The challenger samples a random ciphertext c from some ciphertext space and its corresponding decryption context* (k, n, a)*.*
2. *The challenger sends* (c, a) *to some adversary* $\mathcal{A}$*.*
3. *The adversary wins if it outputs a valid context* (k', n', a) *that decrypt c successfully.*

The adversary's q-advantage $\Delta^{\mathcal{A}}_{\mathsf{CDY}^\dagger}$ *is defined as the probability it wins under q queries to the AEAD oracle* Π*.*

Context Commitment Attack. A context commitment (CMT) attack refers to the adversary's capability to come up with some context (e.g. ciphertext) and provide two interpretations of it. Formally, the game is defined as:

▶ **Definition 2** (Context Commitment). *Fix some AEAD parameter pp and a corresponding AEAD oracle* Π*. The game* $\mathsf{CMT}^\dagger$ *is defined as:*

1. *The challenger samples and sends* (k, n) *to some adversary* $\mathcal{A}$*.*
2. *The adversary wins if it outputs a ciphertext c and two valid contexts* (k, n, a) *and* (k', n', a) *with* $(k, n) \neq (k', n')$ *that decrypts c successfully.*

The adversary's q-advantage $\Delta^{\mathcal{A}}_{\mathsf{CMT}^\dagger}$ *is defined as the probability it wins under q queries to the AEAD oracle* Π*.*

Rationale for the Definition. As Menda et al. [18] have pointed out, there exist many variations of commitment attacks with different limitations on the key, nonce and associated data. Here we observe that in TLS the verifier cannot access the key and the nonce but can access the associated data. Therefore, the adversary (i.e., client $\mathcal{P}$) can only manipulate the key and the nonce. Based on the fact, we pick the above definition that matches our scenario. Nevertheless, we observe that our definition of CDY and CMT is a specification of the granular security framework proposed by Menda et al., and we defer the discussion of the general abstraction to Full Version's Appendix B.

Relationship between CMT and CDY. Menda et al. proved that CMT security implies CDY security, assuming that there is a non-negligible probability that a ciphertext can be decrypted under two different contexts (known as context compression). They also make an analogy that CDY is to CMT as a preimage attack is to a collision attack on a hash function. The required security for AEAD in proxying is a little more lenient than CDY but a little more strict than CMT. We will define an in-between game, the context forgery (CFY) attack, and discuss the relation between the three games in Section 7.2.

Key Commitment Attacks. The two major cipher suites in TLS, AES-GCM and ChaCha20-Poly1305, are not key committing under this definition. The concrete attack is well-studied and presented in multiple works [1, 18, 10].

▶ **Lemma 3** (Key Commitment Attacks). *For* $\mathsf{AEAD} \in \{E\text{-}GCM, H\text{-}Poly1305\}$*, where E is an ideal cipher modeling AES and H is a random function modeling ChaCha20, there exists an adversary that queries the oracle at most q times and wins* $\mathsf{CMT}^\dagger$ *with probability at least* $2^{-32}q$*.*

Protocol Π_{Proxy}

There are a server $\mathcal{W}$, a client $\mathcal{P}$ (i.e., prover), and a verifier $\mathcal{V}$, and server $\mathcal{W}$ behaves the same as in TLS. A predicate $\mathsf{P}(\cdot)$ is to decide if the response from $\mathcal{W}$ satisfies some conditions (e.g., age is greater than 18). A boolean value Mode_p indicates whether the execution is in a private mode.

Handshake phase:

- The client $\mathcal{P}$ and server $\mathcal{W}$ run TLS handshake protocol (see Section 3.1 for more details) via verifier $\mathcal{V}$ (i.e., the messages generated during handshake protocol execution are forwarded by $\mathcal{V}$) to obtain the pair of initial client and server application keys $(\mathsf{CATS}_0, \mathsf{SATS}_0)$;
- The client $\mathcal{P}$ and server $\mathcal{W}$ both compute $(\mathsf{CWK}_0, \mathsf{CWIV}_0) \leftarrow \mathsf{TLS.Derive}(\mathsf{client}, \mathsf{CATS}_0)$ and $(\mathsf{SWK}_0, \mathsf{SWIV}_0) \leftarrow \mathsf{TLS.Derive}(\mathsf{server}, \mathsf{SATS}_0)$;

Request phase:

- The client $\mathcal{P}$ computes $c \leftarrow \mathsf{AEAD.Enc}_{\mathsf{CWK}_i}(\mathsf{CWIV}_i, a, Q)$, where i is initialized as 0 and increases by 1 with each key update (described below), and a is the associated data in TLS 1.3;
- The verifier $\mathcal{V}$ forwards (c, a) to server $\mathcal{W}$;

Response phase:

- Get the query $Q = \mathsf{AEAD.Dec}_{\mathsf{CWK}_i}(\mathsf{CWIV}_i, a, c)$, and obtains the response R according to the current dataset;
- Generate $c' = \mathsf{AEAD.Enc}_{\mathsf{SWK}_i}(\mathsf{SWIV}_i, a, R)$ and sends (c', a') to verifier $\mathcal{V}$;
- After receiving the response ciphertext (c', a') from server $\mathcal{W}$, verifier $\mathcal{V}$ forwards (c', a') to client $\mathcal{P}$;
- The client $\mathcal{P}$ computes $m = \mathsf{AEAD.Dec}_{\mathsf{SWK}_i}(\mathsf{SWIV}_i, a', c')$, if $m \neq \perp$, the process continues, otherwise it aborts;

Prove phase:

- The client $\mathcal{P}$ proves that the response satisfies the conditions defined by the predicate $\mathsf{P}(\cdot)$ as follows:
  - $\mathsf{Mode}_p = 0$: m does not involve private information:
    - The client $\mathcal{P}$ sends m and $(\mathsf{SWK}_i, \mathsf{SWIV}_i)$ to verifier $\mathcal{V}$;
    - The verifier $\mathcal{V}$ checks if $m = \mathsf{AEAD.Dec}_{\mathsf{SWK}_i}(\mathsf{SWIV}_i, a', c')$ and $\mathsf{P}(\tilde{m}) = 1$ where $\tilde{m}$ is the pertinent information extracted from m, output 1 if both conditions hold true, and 0 otherwise.
    - The client $\mathcal{P}$ and server $\mathcal{W}$ update $(\mathsf{CATS}_i, \mathsf{SATS}_i)$ to $(\mathsf{CATS}_{i+1}, \mathsf{SATS}_{i+1})$ and further compute $(\mathsf{CWK}_{i+1}, \mathsf{CWIV}_{i+1})$ and $(\mathsf{SWK}_{i+1}, \mathsf{SWIV}_{i+1})$ using the key update mechanism (see Section 3.1 for more details).
  - $\mathsf{Mode}_p = 1$: m involves private information:
    - The client $\mathcal{P}$ generates a proof π' by invoking $\mathcal{F}_{\mathsf{NIZK}}$ (see Full Version's Appendix A.2) [14] to prove that he knows m, $\tilde{m}$ and $(\mathsf{SWK}_i, \mathsf{SWIV}_i)$ such that $m = \mathsf{AEAD.Dec}_{\mathsf{SWK}_i}(\mathsf{SWIV}_i, a', c')$ and $\tilde{m}$ is extracted from m, and sends π';
    - If π' is verified successfully, verifier $\mathcal{V}$ outputs 1, otherwise 0.

Figure 7 Proxy-based Oracle Protocol.

4 Proxy-based Oracle Protocol

Our proxy-based oracle protocol Π_{Proxy} (shown in Figure 7) is parameterized by a predicate $\mathsf{P}(\cdot)$. There are a server $\mathcal{W}$, a client $\mathcal{P}$ and a verifier $\mathcal{V}$. The client $\mathcal{P}$ aims to prove that $\mathsf{P}(R) = 1$ where R is from server $\mathcal{W}$, to verifier $\mathcal{V}$. To prevent client $\mathcal{P}$ from altering the data received from server $\mathcal{W}$, we use verifier $\mathcal{V}$ as a proxy to relay the messages between server $\mathcal{W}$ and client $\mathcal{P}$. Our protocol Π_{Proxy} consists of four phases: (1) handshake phase, (2) request phase, (3) response phase, and (4) prove phase. Next, we detail the four phases, and define the functionality $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Std}}$ in Figure 11 in Full Version's Appendix C.1.

In the handshake phase, client $\mathcal{P}$ and server $\mathcal{W}$ perform the TLS handshake protocol, and the communication messages are relayed by verifier $\mathcal{V}$. Then, the client and the server can obtain the pair of initial client/server application key $(\mathsf{CATS}_0, \mathsf{SATS}_0)$, and derive the client write key/IV pair $(\mathsf{CWK}_0, \mathsf{CWIV}_0)$ and the server write key/IV pair $(\mathsf{SWK}_0, \mathsf{SWIV}_0)$ (see Section 3.1 for more details). Later, $(\mathsf{CWK}_0, \mathsf{CWIV}_0)$ and $(\mathsf{SWK}_0, \mathsf{SWIV}_0)$ would be updated, so we use $(\mathsf{CWK}_i, \mathsf{CWIV}_i)$ and $(\mathsf{SWK}_i, \mathsf{SWIV}_i)$ to denote the currently used key/IV pairs.

In the request and response phases, client $\mathcal{P}$ and server $\mathcal{W}$ still follow the specification of TLS to generate the request ciphertext c under $(\mathsf{CWK}_i, \mathsf{CWIV}_i)$ and the response ciphertext c' under $(\mathsf{SWK}_i, \mathsf{SWIV}_i)$. The verifier $\mathcal{V}$ is responsible for relaying the two ciphertexts.

In the prove phase, client $\mathcal{P}$ can choose the public mode (i.e., $\mathsf{Mode}_p = 0$) or the private mode (i.e., $\mathsf{Mode}_p = 1$). In the public mode, client $\mathcal{P}$ opens the response m and the server write key/IV pair $(\mathsf{SWK}_i, \mathsf{SWIV}_i)$ to verifier $\mathcal{V}$. Then, verifier $\mathcal{V}$ checks if the response ciphertext c' can be decrypted to m using $(\mathsf{SWK}_i, \mathsf{SWIV}_i)$ and $\mathsf{P}(\tilde{m}) = 1$ where $\tilde{m}$ is the pertinent message extracted from m. If the verification is successful, verifier $\mathcal{V}$ outputs 1, otherwise, outputs 0. Since that $(\mathsf{SWK}_i, \mathsf{SWIV}_i)$ is opened and cannot be used for the subsequent communications, client $\mathcal{P}$ informs server $\mathcal{W}$ to update $(\mathsf{CWK}_i, \mathsf{CWIV}_i)$ and $(\mathsf{SWK}_i, \mathsf{SWIV}_i)$ to $(\mathsf{CWK}_{i+1}, \mathsf{CWIV}_{i+1})$ and $(\mathsf{SWK}_{i+1}, \mathsf{SWIV}_{i+1})$ by using the key update mechanism (see Section 3.1 for more details). In the private mode, the client $\mathcal{P}$ invokes $\mathcal{F}_{\mathsf{NIZK}}$ to generate a proof π' proving that he holds a server write key/IV pair $(\mathsf{SWK}_i, \mathsf{SWIV}_i)$ that can decrypt the response ciphertext c' to m and $\mathsf{P}(\tilde{m}) = 1$ where $\tilde{m}$ is the pertinent message extracted from m. If the proof π' is valid, the verifier $\mathcal{V}$ outputs 1, otherwise, outputs 0.

5 Vulnerability in Unrestricted Setting

While it is tempting to reason the protocol's security on general TLS connections, unfortunately, as we confirm the intuition of previous works [33, 16], the proxy-based oracle protocol is not secure if we only consider the oracle with unrestricted setting $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Std}}$. We present our findings below.

▶ **Theorem 4.** *When the underlying AEAD is vulnerable to key-committing attacks (see Lemma 3), there exists some server $\mathcal{W}$ and predicate $\mathsf{P}(m)$ such that the protocol Π_{Proxy} does not realize the functionality $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Std}}$ (see Figure 11 in Full Version's Appendix C.1).*

Proof. Recall that in key-committing attack (see Definition 2), the adversary (i.e., the client $\mathcal{P}$) can generate a ciphertext c and two contexts (k, n, a) and (k', n', a), where $(k, n) \neq (k', n')$, such that the ciphertext c can be successfully decrypted under the two contexts. At first glance, the key-committing attack is not applicable to our scenario, as it is the server that generates the response ciphertext c. However, since the client $\mathcal{P}$ can obtain (k, n) after the handshake phase and the associated data a can be known beforehand according to the specification of TLS [25, 24], the client $\mathcal{P}$ can manipulate the data stored by the server through a side channel to build a ciphertext c. For instance, the client's balance in some bank can be manipulated by withdrawing or saving funds. Specifically, we then construct an attacker $\mathcal{A}$ that does the following:

$\mathcal{A}$ starts the protocol. It starts handshaking with the server $\mathcal{W}$ through verifier $\mathcal{V}$ to obtain (k, n).

$\mathcal{A}$ builds a ciphertext c and another context $(k', n') \neq (k, n)$, such that $m = \mathsf{Dec}_k(n, a, c)$ and $m' = \mathsf{Dec}_{k'}(n', a, c)$ through traditional key commitment attacks (see Lemma 3).

$\mathcal{A}$ modifies the server-side data to m via side-channel methods, such that the verifier $\mathcal{V}$ receives and records the response ciphertext c.

$\mathcal{A}$ uses (k', n') to convince the verifier $\mathcal{V}$ that m' is the plaintext. ◀

```
HTTP/1.1 200 OK
Date: Sat, 30 Dec 2023 18:52:39 GMT
Server: Apache/2.4.52 (Ubuntu)
Last-Modified: Mon, 11 Dec 2023 01:10:39 GMT
ETag: "25c9-60c319a24a0d5-gzip"
Accept-Ranges: bytes
Vary: Accept-Encoding
Access-Control-Allow-Origin: *
Content-Length: 9673
Connection: close
Content-Type: text/html
```

Figure 8 An HTTP response header from an Apache server. The first two lines (status code and date) can be used as a generic variable padding for all HTTP responses.

A real-life attack scenario can also be found in the famous Facebook attack [10]. In the Facebook attack, the attacker constructs a specific ciphertext *after* the handshake, when it knows the symmetric key that will be used in the communication. It has the server store this ciphertext, which it will decrypt later using a different key other than the one in the handshake.

Nevertheless, not all hope is lost. We observe that the attack in the theorem exploited the fact that the AEAD scheme in TLS is not key-committing. In fact, as we will see in the theorem below, Π_{Proxy} realizes $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Std}}$ if and only if the underlying AEAD is key-committing.

▶ **Theorem 5.** *Given a secure AEAD with key commitment security (see Lemma 3), protocol* Π_{Proxy} *shown in Figure 7 can securely realize the oracle functionality* $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Std}}$ *(see Figure 11 in Full Version's Appendix C.1) in the* $\mathcal{F}_{\mathsf{NIZK}}$*-hybrid model, against a static active adversary who can corrupt client* $\mathcal{P}$ *or verifier* $\mathcal{V}$*.*

We refer the readers to Full Version's Appendix E for proof details.

6 Variable Padding and HTTPS Server

The previous section shows that key commitment is necessary and sufficient for a secure oracle protocol. However, with all that being said, key committing is simply *not* a property present in the current TLS protocol. Fortunately, we find that all HTTPS responses contain an HTTP header at the start of the plaintext, which can be used to prevent key-committing attacks. In this section, we define a notion called *variable padding* to capture the padding-like structure existing in HTTPS responses. Furthermore, we prove that when considering plaintexts with variable padding, both AES-GCM and ChaCha20-Poly1305 are secure against key-committing attacks. Next, we first give more details about variable padding.

6.1 Variable Padding

Figure 8 contains a typical response header from an Apache server. The sample header has 308 bytes of text – enough for 19 blocks of AES. Albertini et al. [1] have shown that AEAD schemes can be made key-committing with 4 blocks of fixed padding. However, we observe that an adversarial prover in our game has some wiggle room for the header, because it may be able to determine parameters such as HTTP status code and response date. Nevertheless, we find its capability highly constrained, since all of these parameters have limited ranges of value. For instance, the HTTP status code has only 63 variations. If we take a rather generous time window of one hour between the response of the server and the receival of the verifier, then the response date has no more than 3600 different variations. This motivates us to define the concept of variable padding: padding with limited variations controlled by the adversary.

▶ **Definition 6** (Variable padding). *Consider a set S that consists of only λ-bit strings. We say that a string m is variably padded by S, if the λ-bit prefix of m is in S.*

For instance, consider the sample header in Figure 8. We can easily verify that the first 54 bytes consist of only the status code and the date as the variable parameters. Therefore, we can assert that there exists a 54-byte string set S with $|S| = 63 \times 3600$, such that all response plaintext produced by this particular server is variably padded by S.

Application to Generic HTTPS Servers. While to achieve the tightest security bound, the specific padding and length should be analyzed on a per-server basis, we can nevertheless demonstrate a generic bound based on the HTTP status code and date header.

▶ **Corollary 7.** *Every valid HTTP response is variably padded by a 54-byte string set S with $|S| = 63t$ if it follows good practice and is received within t seconds.*

This corollary is verified by noticing that RFC 7231 requires any server with a reasonable clock to send the date header, and states "it is good practice to send header fields that contain control data first, such as... Date on responses". Notably, both Apache and Nginx, the two popular HTTP server implementations, have Server and Date as the first two response headers; meanwhile, the HTTP header with a date takes at least 54 bytes.

6.2 Key Commitment with Variable Padding

Next, we consider an adversarial prover's advantage with a variably padded plaintext, in AES-GCM and ChaCha20-Poly1305. Since the verifier has access to the ciphertext transcript, the attacker needs to come up with some ciphertext c and two pairs of key and nonce $((k, n), (k', n'))$ such that both $\mathsf{Dec}_k(n, a, c)$ and $\mathsf{Dec}_{k'}(n', a, c)$ give a plaintext variably padded by some S.

Analysis on AES-GCM. We first formalize our statement for AES-GCM.

▶ **Theorem 8** (GCM Key commitment with variable padding). *Assume E is an ideal block cipher with l-bit block size and $(\mathsf{Enc}, \mathsf{Dec})$ is a GCM scheme defined on E. The adversary's advantage of constructing a ciphertext c, an associated data a, and two key/nonce pairs $((k, n), (k', n'))$ such that both $\mathsf{Dec}_k(n, a, c)$ and $\mathsf{Dec}_{k'}(n', a, c)$ give a plaintext variably padded by some λ-bit string set S within q queries to E is at most $\epsilon = \frac{q^2|S|^2}{2^{\lambda-2v+1}} \left(\frac{2^l}{2^l-b}\right)^{b+1}$, where $b = \lfloor \frac{\lambda}{l} \rfloor$ and v is the bit-length of IV.*

Proof. We first fix a key/nonce pair (k, n) and two prefixes (s, s') and consider the bad event BAD where there exists another key/nonce pair (k', n') and a ciphertext c such that $\mathsf{Dec}_k(n, a, c)$ has prefix s and $\mathsf{Dec}_{k'}(n', a, c)$ has prefix s'. Let us upper-bound the probability $\Pr(\mathsf{BAD})$ as follows.

We assume that the λ-bit prefix spans across b blocks and λ_r extra bits, which means that $\lambda = l \cdot b + \lambda_r$. We know that for some ciphertext c, $\mathsf{Dec}_k(n, a, c)$ has prefix s and $\mathsf{Dec}_{k'}(n', a, c)$ has prefix s'. Therefore, according to the AES-GCM encryption process, the following equation set holds:

$$
\begin{aligned}
E_k(n+1) + s[0:l] &= c[0:l] = E_{k'}(n'+1) + s'[0:l], \\
E_k(n+2) + s[l:2l] &= c[l:2l] = E_{k'}(n'+1) + s'[l:2l], \\
&\vdots \\
E_k(n+b+1) + s[bl:\lambda] &= c[bl:\lambda] = E_{k'}(n'+b+1) + s'[bl:\lambda].
\end{aligned}
$$

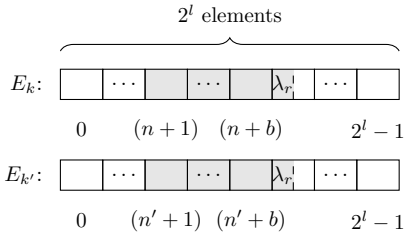


Figure 9 A representation of E_k and $E_{k'}$ as permutations. The shaded part in $E_{k'}$ is uniquely determined from the shaded part in E_k.

Recall that we consider fixed (k, n) and prefixes s and s'. Therefore, the equation set above tells us that b blocks and λ_r extra bits of $E_{k'}$ are also fixed. Next, we calculate how many $E_{k'}$ can satisfy the requirement.

Because E is an ideal cipher and block size is l bits, both E_k and $E_{k'}$ are permutations on 2^l elements. In Figure 9, we show the results of the permutation on the 2^l elements. We assume that from $(n+1)$-th block to $(n+b)$-th block are fixed and the first λ_r bits in the $(n+b+1)$-th block are also fixed. We can see that the remaining $(l-\lambda_r)$ bits in the $(n+b+1)$-th block can be randomly chosen from $\{0,1\}^{l-\lambda_r}$. Once the remaining $(l-\lambda_r)$ bits in the $(n+b+1)$-th block are chosen, the $(b+1)$ blocks are fixed. Then, the remaining (2^l-b-1) blocks have a total of $(2^l-b-1)!$ possible assignments. Therefore, the upper bound of the number of $E_{k'}$ satisfying the above requirement is given by $|E_{k'}| \leq 2^{l-\lambda_r}(2^l-b-1)!$.

Without considering the above requirement, $E_{k'}$ could be randomly chosen from $(2^l)!$ permutations. Hence the probability of BAD happening is at most $\Pr(\mathsf{BAD}) \leq \frac{2^{l-\lambda_r}(2^l-b-1)!}{(2^l)!} < \frac{2^{l-\lambda_r}}{(2^l-b)^{b+1}} = \frac{1}{2^\lambda}\left(\frac{2^l}{2^l-b}\right)^{b+1}$.

Now the total probability is bounded by the summation of all different keys the adversary tests (note that the adversary queries E at most q times) and all different nonces and prefixes, so $\Delta^{\mathcal{A}} < \underbrace{\binom{q}{2}}_{\text{keys}} \cdot \underbrace{2^{2v}}_{\text{nonces}} \cdot \underbrace{|S|^2}_{\text{prefixes}} \cdot \Pr(\mathsf{BAD}) < \frac{q^2|S|^2}{2^{\lambda-2v+1}}\left(\frac{2^l}{2^l-b}\right)^{b+1}$. ◀

Since in TLS, the nonce n for AES-GCM is uniquely determinely by a 96-bit IV, we have $v = 96$. Plugging in the generic HTTP padding with $\lambda = 8 \times 56$ bits, $l = 128$ bits, $|S| = 63t$ and a generous $t = 3600$ seconds gives us $\epsilon \leq 2^{-221}q^2$. Note that $\left(\frac{2^l}{2^l-b}\right)^{b+1} \approx 1$.

Therefore, the protocol is secure on HTTPS servers.

Analysis on ChaCha20-Poly1305. Similarly, below we demonstrate the security of ChaCha20-Poly1305.

▶ **Theorem 9** (Poly1305 Key commitment with variable padding). *Assume $H : \mathbb{Z}^{|k|} \times \mathbb{Z}^{|n|} \to \mathbb{Z}^l$ is an ideal random function representing ChaCha20 and* $(\mathsf{Enc}, \mathsf{Dec})$ *is a Poly1305 scheme defined on H. The adversary's advantage of constructing a ciphertext c, an associated data a, and two key/nonce pairs $((k,n),(k',n'))$ such that both $\mathsf{Dec}_k(n,a,c)$ and $\mathsf{Dec}_{k'}(n',a,c)$ give a plaintext variably padded by some λ-bit string set S within q queries to H is at most $\epsilon = \frac{q^2|S|^2}{2^{\lambda-2v+1}}$, where v is the bit-length of IV.*

Proof. The only difference in the analysis between AES-GCM and ChaCha20-Poly1305 is that AES is a random permutation and ChaCha20 is a random function. Therefore, following a similar analysis, we can bound the probability of getting a bad random function to $\Pr(\mathsf{BAD}) \leq \frac{2^{l-\lambda_r}(2^l)^{2^l-b-1}}{(2^l)^{2^l}} = \frac{2^{l-\lambda_r}}{(2^l)^{b+1}} = \frac{1}{2^\lambda}$. Therefore, the overall probability is bounded by $\Delta^{\mathcal{A}} < \binom{q}{2} \cdot 2^{2v} \cdot |S|^2 \cdot \Pr(\mathsf{BAD}) < \frac{q^2|S|^2}{2^{\lambda-2v+1}}$. ◀

Similarly, in TLS, the nonce n for ChaCha20-Poly1305 is uniquely determined by a 96-bit IV. Therefore, we have $v = 96$. Plugging in the generic HTTP padding with $\lambda = 8 \times 56$ bits, $l = 512$ bits, $|S| = 63t$ and $t = 3600$ seconds gives us $\epsilon \leq 2^{-221}q^2$.

An application of Theorem 8 and Theorem 9 gives us the following desired security property.

▶ **Theorem 10.** *Assume TLS uses either AES-GCM or ChaCha20-Poly1305. Fix some set variable padding S of polynomial size. Let $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{VP}}$ denote the same functionality as $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Std}}$, except that the server's response is guaranteed to be variably padded by S. Protocol Π_{Proxy} shown in Figure 7 can securely realize the oracle functionality $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{VP}}$ defined in Figure 11 in the $\mathcal{F}_{\mathsf{NIZK}}$-hybrid model, against a static active adversary who can corrupt client $\mathcal{P}$ or verifier $\mathcal{V}$.*

The proof follows a direct application of the above analysis on AES-GCM and ChaCha20-Poly1305 to Theorem 5.

7 Beyond HTTPS: without Variable Padding

We have shown that proxying is secure over HTTPS thanks to the variable padding. Nonetheless, exploring the potential usage of proxying over home-bake protocols is still an interesting topic from a theoretical perspective. We have seen that proxying is not secure given any home-bake protocol: the Facebook attack [10] is a practical counter-example. Therefore, we must restrict the adversarial prover's power to some extent. In this section, we explore the scenario where the prover must fix the server's response before the protocol. This is common in many daily scenarios. For instance, an adversarial prover should not be able to change his age depending on the handshake secret.

We formalize the intuition as functionality $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Fix}}$ in Figure 12 in Full Version's Appendix C.2. Unlike the functionality $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Std}}$, the functionality $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Fix}}$ obtains the data related to client $\mathcal{P}$, denoted as $\mathsf{Data}_{\mathcal{P}}$, from the environment $\mathcal{E}$, before the handshake phase. Then, in the response phase, the functionality $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Fix}}$ applies the query Q to $\mathsf{Data}_{\mathcal{P}}$ to obtain the response R. This reflects that $\mathsf{Data}_{\mathcal{P}}$ is fixed before the handshake phase.

7.1 Context Forgery Attack

What we are considering is akin to a forgery attack: our adversarial attacker must come up with an additional explanation of the ciphertext provided by the server. We notice that the context discovery (CDY) attack on AEAD (see Definition 1) considered by Menda et al. [18] is closely related to our problem, but there are notable differences.

Specifically, the adversarial $\mathcal{P}$ considered by us has more knowledge than what is described under the CDY model in Menda et al. [18], since he knows the original context (i.e., key/nonce pair) of the ciphertext. Moreover, he also has a different goal in mind: he cannot just find any context that decrypts the ciphertext – he has to find one that is different from the original context. Based on these differences, we propose a new model named context forgeability

(CFY) and relate this to the second-preimage attack on a hash function, similar to how CDY is to CMT as a preimage attack is to a collision attack. More formally, we can define the context forgery attack game as:

▶ **Definition 11** (Context Forgery). *Fix some AEAD parameter pp and a corresponding AEAD oracle* Π. *The game* $\mathsf{CDY}^\dagger$ *is defined as:*

1. *The challenger samples a random ciphertext* c *from some ciphertext space and its corresponding decryption context* (k, n, a).
2. *The challenger sends* (c, k, n, a) *to some adversary* $\mathcal{A}$.
3. *The adversary wins if it outputs a valid context* (k', n', a) *with* $(k, n) \neq (k', n')$ *that decrypts* c *successfully.*

The adversary's q*-advantage* $\Delta^{\mathcal{A}}_{\mathsf{CDY}^\dagger}$ *is defined as the probability it wins under* q *queries to the AEAD oracle* Π.

Analysis on AES-GCM. AES-GCM is known for various commitment weaknesses [10], and it is not difficult to intuit that it is not secure in this scenario. Since a CDY attack implies a CFY attack (see Corollary 16), we can use ideas that Menda et al. [18] proposed for the CDY attack to give an attack for CFY.

▶ **Theorem 12.** *There is an attack under* $\mathsf{CFY}^\dagger$ *that breaks E-GCM with probability at least* $\frac{q}{2^{33}}$, *assuming* E *is an 128-bit ideal block cipher.*

Proof. For simplicity, let us consider a message with one block of ciphertext and no associated data. If we obtain some key k, nonce n and the only ciphertext block c from the input, the tag t follows the definition of GCM where $t = E_k(n) + E_k(0)^2 c + E_k(0)$. Now suppose we sample some other key $k' \neq k$, then we have the following equation, given the fixed tag t and ciphertext block c that $t = E_{k'}(n') + E_{k'}(0)^2 c + E_{k'}(0)$.

It is obvious that given $(E_{k'}(0), c, t)$, $E_{k'}(n)$ can be solved using the above equation. Moreover, since $E_{k'}$ is reversible, we can solve for n'. In AES-GCM the nonce must end with $(0^{31} \| 1)$, so this particular n' has $\frac{1}{2^{32}}$ probability of satisfying the requirement. Observing that this try of k' uses two queries of E (that is, one is for $E_{k'}(0)$ and the other is for decrypting $E_{k'}(n')$), we can bound the success probability of q queries to $\frac{q}{2^{33}}$. ◀

The above attack can be generalized to any length of associated data and ciphertext, as discussed by Menda et al. [18]. Moreover, many works [18, 10] have shown that AES-GCM polyglot ciphertext can be decrypted to different *meaningful* plaintext under different keys.

Analysis on ChaCha20-Poly1305. To analyze game $\mathsf{CFY}^\dagger$ under ChaCha20-Poly1305, we first abstract its authentication tag computation mechanism as $t = \mathsf{poly}(r) + s$, where poly is a polynomial whose coefficient depends only on the concatenation of the associated data and the ciphertext, and $(r, s) = H(k, n)[0 : 256]$ is the secret generated by the ChaCha20 pseudorandom function. We then demonstrate its security in $\mathsf{CFY}^\dagger$.

▶ **Theorem 13.** *The adversary's advantage in* $\mathsf{CFY}^\dagger$ *when attacking H-Poly1305 is no more than* $\frac{q}{2^{128}}$, *where* q *is the number of queries the adversary made to* H *and* $H(k, n)$ *is a 512-bit random oracle.*

Proof. We start by fixing some key and nonce (k', n') the adversary has queried for the first time. Observe that since H is a random oracle, $\Pr(H(k', n') = h)$ is a uniform distribution regardless of any prior queries. Therefore, $(r', s') = H(k', n')[0 : 256]$ is also uniformly distributed over the whole range. Now observe that in order for (k', n') to satisfy the tag

requirement, we have $\mathsf{poly}(r') + s' = t = \mathsf{poly}(r) + s$. Rearrange and we have $s' - s = \mathsf{poly}(r) - \mathsf{poly}(r')$. Observe that for some fixed s, $s' - s$ is still uniformly distributed over $\mathbb{Z}/2^{128}\mathbb{Z}$. Denote $\Delta p(r') = \mathsf{poly}(r) - \mathsf{poly}(r')$. We can bound the probability of (k', n') satisfying the tag requirement $\Pr(\mathsf{BAD})$ by $\Pr(\mathsf{BAD}) = \sum_{i \in \mathbb{Z}/2^{128}\mathbb{Z}} \Pr(s' - s = i) \Pr(\Delta p(r') = i) = \frac{1}{2^{128}} \sum_i \Pr(\Delta p(r') = i) = \frac{1}{2^{128}}$. Since the attacker makes q queries, we bound the probability to $\frac{q}{2^{128}}$. ◀

Protocol Security Under CFY. We demonstrate that the proxying protocol Π_{Proxy} realizes $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Fix}}$ as long as the underlying AEAD is secure under $\mathsf{CFY}^\dagger$.

▶ **Theorem 14.** *Given a secure AEAD with context unforgeability security under* $\mathsf{CFY}^\dagger$*, the protocol* Π_{Proxy} *shown in Figure 7 can securely realize the functionality* $\mathcal{F}_{\mathsf{Oracle}}^{\mathsf{Fix}}$ *in Figure 12 in the* $\mathcal{F}_{\mathsf{NIZK}}$*-hybrid model, against a static active adversary who can corrupt client* $\mathcal{P}$ *or verifier* $\mathcal{V}$*.*

We refer the readers to Full Version's Appendix F for the full proof.

7.2 Relationship between CMT, CFY and CDY

An interesting observation is that a similar hierarchy exists concerning CDY, CFY and CMT just like the hash function. CMT-security implies CFY-security, and CFY-security implies CDY-security assuming context compression. We provide our insight in the following two corollaries and defer a formal proof on the generalized case under Menda et al.'s framework to Full Version's Appendix B.

▶ **Corollary 15.** *For some AEAD* Π *and any adversary* $\mathcal{A}$ *that wins the* $\mathsf{CFY}^\dagger$ *game with advantage* $\Delta^{\mathcal{A}}_{\mathsf{CFY}^\dagger}$*, there exists an adversary* $\mathcal{B}$ *that wins the* $\mathsf{CMT}^\dagger$ *game with advantage* $\Delta^{\mathcal{B}}_{\mathsf{CMT}^\dagger} = \Delta^{\mathcal{A}}_{\mathsf{CFY}^\dagger}$*.*

The relationship between CFY and CDY, on the other hand, is not so clear-cut. While a CDY attack on an AEAD scheme often implies a CFY attack, it is not universally true. In the case where a ciphertext cannot be decrypted under two different contexts (i.e., the following BadCtx event), a CDY attack may be easy but a CFY attack will be impossible. However, similar to the analysis of Menda et al. on CDY, we can bound the advantage of the adversary with the probability of BadCtx:

▶ **Corollary 16.** *For some AEAD* Π *and any adversary* $\mathcal{A}$ *that wins the* $\mathsf{CDY}^\dagger$ *game with advantage* $\Delta^{\mathcal{A}}_{\mathsf{CDY}^\dagger}$*, there exists an adversary* $\mathcal{B}$ *that wins the* $\mathsf{CFY}^\dagger$ *game with advantage* $\Delta^{\mathcal{B}}_{\mathsf{CFY}^\dagger}$ *such that* $\Delta^{\mathcal{B}}_{\mathsf{CFY}^\dagger} \geq \frac{1}{2}(1 - \Pr[\mathsf{BadCtx}])\Delta^{\mathcal{A}}_{\mathsf{CDY}^\dagger}$*, where* BadCtx *is the event that for a randomly sampled* (K, N, M, A)*, the resulting ciphertext* C *can only be decrypted when the associated data is* A*.*

8 Evaluation

Due to the fact that the related works' implementations are close-sourced, including DECO [33], the work by Xie et al. [31], Janus [16], ORIGO[5] [12], we conduct an empirical survey to outline the cost that can be saved by our technique in Table 1, using numbers reported in these papers. We split the result from these papers into the following three parts, and our result in this paper can eliminate the cost in key-related assurance:

[5] The end-to-end codebase in ORIGO (`https://github.com/opex-research/tls-oracle-demo`) is open-sourced, but the zero-knowledge component (`https://github.com/opex-research/tls-zkp`) is not.

Table 1 Time comparison between various related works, using AES-GCM as the cipher suite. Shaded parts represent the cost we can save with our result. For reference, ORIGO [12] measured the TLS overhead to be 1.26 s under 1 Gbps WAN network. DECO tested query and record layer assurance separately, but did not report response size.

Work	Scenario	Record Type	Resp. Size (B)	Key-Related (s) Off.	Key-Related (s) On.	Record (s) Off.	Record (s) On.
ORIGO [12] (Groth16)	Circuit[1]	Open[2] (32 B)	1429	22.47	1.08	10.55	0.61
ORIGO [12] (Plonk)	Circuit	Open (32 B)	1429	27.51	14.84	13.99	7.67
Janus [16]	LAN	Open (256 B)	2048	3.94	4.79	1.13	2.08
Xie et al. [31]	LAN	–	1024	–	0.61[4]	–	
DECO [33] (512 B Query)	LAN	–	–	2.28	0.42	–	
DECO [33] (512 B Query)	WAN (100 Mbps)	–	–	22.81	5.20	–	
DECO [33] (1 KB Query)	LAN	–	–	2.50	0.47	–	
DECO [33] (1 KB Query)	WAN (100 Mbps)	–	–	24.59	7.42	–	
DECO [33]	Circuit	Binary Options[3]	–	–		–	12.97
DECO [33]	Circuit	Age Proof	–	–		–	3.67
DECO [33]	Circuit	Price Discrimination	–	–		–	12.68

1. Local computation of proof circuit.
2. Selective opening of a substring of the response.
3. DECO designed some scenarios where the prover proves a specific statement.
4. Xie et al.'s work only reported the online computation cost of 2-party computations.

1. TLS proxy overhead: cost associated with TLS proxying and network cost.
2. Key-related assurance: cost associated with ensuring key commitment. Some prior work [33, 31, 16] utilizes 2PC protocols to generate query since the key is split between the client and the oracle. Here, we consider the processes of three-party handshake and relevant 2PC to be in this category, since our result does not need to split the key, and eliminates these costs.
3. Record layer assurance: cost on proving statements in the record layer.

Based on empirical data, in the simple case where everything that needs to be hidden resides in the request and the record layer assurance does not need to be deployed, we can achieve around a 90% saving in running time by eliminating key-related assurance. On the other hand, if we consider typical record layer assurance protocols deployed by these works, we can achieve around a 60% saving in performance, depending on the complexity of the record layer circuit.

9 Related Work

AEAD Commitment Security. It is known that a lot of AEAD schemes do not have the most robust commitment security. One prominent example of an exploit is the attack on Facebook Messenger by Dodis et al. [10]. Commitment security on AEAD has received a lot of research since around that time [15, 8]. Menda et al. provided a generalization and analysis on the topic [18], which we leverage in this paper. Albertini et al. analyzed several possible ways to fix popular AEAD schemes [1]. Our work builds and extends on the framework by Menda et al. [18] and gives it a practical scenario. With this toolset, we are able to reason the security of proxying and give concrete security bounds with proof.

TLS Proxy & Oracle Protocols. Using TLS under multi-party scenarios has been investigated a number of times under different scenarios [19, 3, 14, 27]. TLS oracle protocols have also recently been studied by a variety of academic papers. Earlier results often include modifying the TLS server to some extent and using trusted hardware which are considered not universal [32, 26]. DECO by Zhang et al. [33] in 2020 is one of the first papers that combines a TLS oracle with zero-knowledge proof to preserve user privacy over a general TLS server. Janus by Lauinger et al. [16] demonstrates an efficient two-party computation that optimizes the performance of zero-knowledge proof by the client. A work in a similar direction by Xie et al. [31] uses the garble-then-prove technique instead. DIDO by Chan et al. [9] proposes further optimization based on TLS 1.3. ORIGO [12], an independent and concurrent work of ours, observed that the ZK proof for key derivation can be optimized.

Meanwhile, there is also significant industry effort on this topic. The recent Reclaim Protocol [22] provides an implementation that is based purely on proxying without the three-party handshake but provides no security proof of the integrity of the data. Thus our work is also a theoretical discussion of the industry effort that validates its usage under common scenarios but also points out its limitations.

10 Conclusion and Discussion

In this paper, we formalize the notion of a proxy TLS oracle protocol that does not enter any communication between the client and the server like previous works. We first reason for its limitation on arbitrary TLS protocols, confirming the intuition of previous works. We then prove that the proxy protocol is secure under an application layer protocol with a variable padding, such as HTTPS. We further explore the scenario where the application layer does not provide variable padding. We show that if the adversary cannot tamper with the response after establishing the connection, context unforgeability (CFY) of the underlying AEAD is sufficient to demonstrate security. We analyze the cipher suites in TLS, and find that AES-GCM does not satisfy the property, but ChaCha20-Poly1305 does.

Multi-Round Support. We observe that the NIZK proof of the protocol does not reveal the key/nonce pair used in the communication. Therefore, our protocol naturally supports the client and the server communicating in multiple rounds after a handshake connection, as in the original TLS.

In practice, when the data do not involve privacy, the client can choose to directly reveal the key/nonce pair and data to the verifier, rather than using the NIZK proof. In this case, since HTTP is a stateless protocol, the client can terminate the connection and handshake again to start a new one for future rounds. Moreover, we note that TLS 1.3 supports a key update mechanism (see Figure 6) that allows us to refresh the key/nonce pair for subsequent communications without needing another handshake.

Side-Channel Connection. Our protocol does not stop an adversarial client from connecting to the server simultaneously without proxying through the verifier. In fact, when the web server is not restricted in its response, there is an attack using a side channel connection (see Theorem 4), and we prove that the HTTPS response format can be used to defend against the attack. Moreover, in our scenario, the purpose of the client is to have the data obtained from the server validated by the verifier so that the data can be submitted to the blockchain. Therefore, the client has no motivation to bypass the verifier when he tries to acquire the data.

BGP Attacks and Proxy-Server Connection Hijacking. In the main body of the paper, we assumed that the verifier has a trusted connection to the TLS server. Indeed, the problem is not meaningful if the server is not trusted or if there is no trusted way to access the server, since there would not exist any root of trust under this scenario.

Here, we note that if the client can adaptively corrupt the connection between the verifier and the TLS server, it can first establish the connection with the real server and obtain the HTTPS certificate, and then hijack the connection to transmit the message with the master secret it knows. This attack works even if the verifier pins the server certificate in advance, but requires the client to adaptively corrupt the connection.

References

1 Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. How to abuse and fix authenticated encryption without key commitment. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 3291–3308. USENIX Association, August 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/albertini.

2 Mihir Bellare and Viet Tung Hoang. Efficient schemes for committing authenticated encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 845–875. Springer, Heidelberg, May / June 2022. doi:10.1007/978-3-031-07085-3_29.

3 Karthikeyan Bhargavan, Ioana Boureanu, Antoine Delignat-Lavaud, Pierre-Alain Fouque, and Cristina Onete. A formal treatment of accountable proxying over TLS. In *2018 IEEE Symposium on Security and Privacy*, pages 799–816. IEEE Computer Society Press, May 2018. doi:10.1109/SP.2018.00021.

4 Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, et al. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks, 2021. URL: https://chain.link/.

5 Vitalik Buterin. Ethereum white paper: A next-generation smart contract and decentralized application platform. https://finpedia.vn/wp-content/uploads/2022/02/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf. Accessed: March 28, 2024.

6 Sofía Celi, Alex Davidson, Hamed Haddadi, Gonçalo Pestana, and Joe Rowell. Distefano: Decentralized infrastructure for sharing trusted encrypted facts and nothing more. Cryptology ePrint Archive, Paper 2023/1063, 2023. URL: https://eprint.iacr.org/2023/1063.

7 Chainlink Foundation. What is the blockchain oracle problem? https://blog.chain.link/what-is-the-blockchain-oracle-problem/. Accessed: March 28, 2024.

8 John Chan and Phillip Rogaway. On committing authenticated-encryption. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022, Part II*, volume 13555 of *LNCS*, pages 275–294. Springer, Heidelberg, September 2022. doi:10.1007/978-3-031-17146-8_14.

9 Kwan Yin Chan, Handong Cui, and Tsz Hon Yuen. DIDO: data provenance from restricted TLS 1.3 websites. In *Information Security Practice and Experience*, pages 154–169. Springer Nature, 2023. doi:10.1007/978-981-99-7032-2_10.

10 Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 155–186. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96884-1_6.

11 Morris Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. Technical Report NIST Special Publication (SP) 800-38D, National Institute of Standards and Technology, Gaithersburg, MD, 2007. doi:10.6028/NIST.SP.800-38D.

12 Jens Ernstberger, Jan Lauinger, Yinnan Wu, Arthur Gervais, and Sebastian Steinhorst. ORIGO: Proving provenance of sensitive data with constant communication. Cryptology ePrint Archive, Paper 2024/447, 2024. URL: https://eprint.iacr.org/2024/447.

13 Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi.

14 Paul Grubbs, Arasu Arun, Ye Zhang, Joseph Bonneau, and Michael Walfish. Zero-knowledge middleboxes. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 4255–4272. USENIX Association, August 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/grubbs.

15 Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 66–97. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63697-9_3.

16 Jan Lauinger, Jens Ernstberger, Andreas Finkenzeller, and Sebastian Steinhorst. Janus: Fast privacy-preserving data provenance for TLS 1.3. Cryptology ePrint Archive, Report 2023/1377, 2023. URL: https://eprint.iacr.org/2023/1377.

17 David McGrew. An Interface and Algorithms for Authenticated Encryption. RFC 5116, January 2008. doi:10.17487/RFC5116.

18 Sanketh Menda, Julia Len, Paul Grubbs, and Thomas Ristenpart. Context discovery and commitment attacks - how to break CCM, EAX, SIV, and more. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part IV*, volume 14007 of *LNCS*, pages 379–407. Springer, Heidelberg, April 2023. doi:10.1007/978-3-031-30634-1_13.

19 David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R. López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS. In *ACM SIGCOMM 2015*, volume 45, pages 199–212. ACM Press, August 2015. doi:10.1145/2829988.2787482.

20 Henrik Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999. doi:10.17487/RFC2616.

21 Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539, May 2015. doi:10.17487/RFC7539.

22 Reclaim Protocol. Reclaim protocol: Claiming and managing self-sovereign credentials, 2023. URL: https://www.reclaimprotocol.org/.

23 Q-Success. Usage statistics of default protocol https for websites. https://w3techs.com/technologies/details/ce-httpsdefault. Accessed: April 10, 2024.

24 Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. doi:10.17487/RFC8446.

25 Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008. doi:10.17487/RFC5246.

26 Hubert Ritzdorf, Karl Wüst, Arthur Gervais, Guillaume Felley, and Srdjan Capkun. TLS-N: Non-repudiation over TLS enablign ubiquitous content signing. In *NDSS 2018*. The Internet Society, February 2018. URL: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_09-4_Ritzdorf_paper.pdf.

27 Sijun Tan, Weikeng Chen, Ryan Deng, and Raluca Ada Popa. MPCAuth: Multi-factor authentication for distributed-trust systems. In *2023 IEEE Symposium on Security and Privacy*, pages 829–847. IEEE Computer Society Press, May 2023. doi:10.1109/SP46215.2023.10179481.

28 Liang Wang, Gilad Asharov, Rafael Pass, Thomas Ristenpart, and abhi shelat. Blind certificate authorities. In *2019 IEEE Symposium on Security and Privacy*, pages 1015–1032. IEEE Computer Society Press, May 2019. doi:10.1109/SP.2019.00007.

29 David Warburton. The 2021 TLS telemetry report. https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report. Accessed: April 10, 2024.

30 The OpenSSL Wiki. Tls1.3, 2023. URL: https://wiki.openssl.org/index.php/TLS1.3.

31 Xiang Xie, Kang Yang, Xiao Wang, and Yu Yu. Lightweight authentication of web data via garble-then-prove. Cryptology ePrint Archive, Report 2023/964, 2023. URL: https://eprint.iacr.org/2023/964.

32 Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 270–282. ACM Press, October 2016. doi:10.1145/2976749.2978326.

33 Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. DECO: Liberating web data using decentralized oracles for TLS. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1919–1938. ACM Press, November 2020. doi:10.1145/3372297.3417239.

Blockchain Governance via Sharp Anonymous Multisignatures

Wonseok Choi
DGIST, Daegu, South Korea

Xiangyu Liu[1]
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Vassilis Zikas
Georgia Tech, Atlanta, GA, USA

Abstract

Electronic voting has occupied a large part of the cryptographic protocols literature. The recent reality of blockchains – in particular, their need for online governance mechanisms – has brought new parameters and requirements to the problem. We identify the key requirements of a blockchain governance mechanism, namely correctness (including eliminative double votes), voter anonymity, and traceability, and investigate mechanisms that can achieve them with minimal interaction and under assumptions that fit the blockchain setting.

First, we define a signature-like primitive, which we term *sharp anonymous multisignatures* (in short, ♯AMS) that tightly meets the needs of blockchain governance. In a nutshell, ♯AMSs allow any set of parties to generate a signature, e.g., on a proposal to be voted upon, which, if posted on the blockchain, hides the identities of the signers/voters but reveals their number. This can be seen as a (strict) generalization of threshold ring signatures (TRS).

We next turn to constructing such ♯AMSs and using them in various governance scenarios – e.g., single vote vs. multiple votes per voter. In this direction, although the definition of TRS does not imply ♯AMS, one can compile some existing TRS constructions into ♯AMS. This raises the question: What is the TRS structure that allows such a compilation? To answer the above, we devise templates for TRSs. Our templates encapsulate and abstract the structure that allows for the above compilation – most of the TRS schemes that can be compiled into ♯AMS are, in fact, instantiations of our template. This abstraction makes our template generic for instantiating TRSs and ♯AMSs from different cryptographic assumptions (e.g., DDH, LWE, etc.). One of our templates is based on chameleon hashes, and we explore a framework of lossy chameleon hashes to understand their nature fully.

Finally, we turn to how ♯AMS schemes can be used in our applications. We provide fast (in some cases non-interactive) ♯AMS-based blockchain governance mechanisms for a wide spectrum of assumptions on the honesty (semi-honest vs malicious) and availability of voters and proposers.

2012 ACM Subject Classification Security and privacy → Digital signatures

Keywords and phrases Blockchain, E-voting, Threshold Ring Signatures, Threshold Cryptography

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.5

Related Version *Full Version*: `https://eprint.iacr.org/2023/1881`

Funding Wonseok Choi and Xiangyu Liu were supported in part by Sunday Group, Inc.
Vassilis Zikas: Vassilis Zikas was supported in part by NSF Award No. 2448339, AnalytiXIN, and Sunday Group, Inc.

Acknowledgements This work was done when authors were at Purdue University and Georgia Tech.

[1] Corresponding author.

© Wonseok Choi, Xiangyu Liu, and Vassilis Zikas;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Since the emergence of Bitcoin [53] in 2009, the world of cryptocurrencies and blockchain platforms has witnessed a surge in popularity. One of the distinguishing features of these blockchain platforms is their decentralized nature, wherein decision-making authority is distributed among various actors within the ecosystem.

There are two basic governance mechanisms: off-chain governance, as seen in Bitcoin and Ethereum, and on-chain governance, exemplified by projects like Algorand [17], Tezos [32], and EOS. While off-chain governance allows core contributors to work more seamlessly, it contradicts the philosophy of decentralization. Conversely, on-chain governance faces technical challenges, and this area of research is still relatively new and in its early stages.

Regardless of the governance mechanism used, blockchain platforms face a common vulnerability: community divisions, often resulting in hard forks. A hard fork arises when stakeholders disagree on a critical change, leading to some sticking with the current chain while others adopt the new one. Alternatively, competing updates may further fragment the community. These divisions can erode cohesion, devalue the platform, and jeopardize security. Security concerns are paramount, as a reduced number of resources supporting a fork can make it susceptible to attacks like 51% attacks.

On-Chain Governance and Voting Systems for Improvement Proposals

On-chain governance formalizes decision-making by embedding governance rules directly into the blockchain protocol, offering a transparent and verifiable process for implementing upgrades. At the heart of this system lies the voting mechanism, which determines how stakeholders collectively approve or reject proposals that shape the network's evolution. The integrity and security of this voting process are therefore critical: any vulnerability can undermine trust, distort consensus, and jeopardize the decentralized nature of the ecosystem.

Generally, there are three periods for on-chain governance: the posting, voting, and announcement periods. The developers submit their improvement proposals to the blockchain during the posting period. Then, in the voting period, eligible voters participate in the voting protocol to vote for their preferred proposals. Finally, the voting result is announced in the announcement period, and the most voted proposal is elected.

There are several foundational requirements for a robust voting system.

Correctness. The accuracy of the voting result is perhaps the most important property. A key goal here is to prevent double-voting, wherein a voter casts more than one vote on the same or different proposals. This act of multiple-voting contradicts the standard single-vote setting, wherein each voter is restricted to voting only once, irrespective of the proposal chosen. We emphasize that the permissibility of multiple-voting is contingent upon the specific application. In this context, double-voting includes instances where a malicious voter attempts to cast more than one vote for a single proposal, and such duplications are promptly nullified.

(Unconditional) Anonymity of Voters. A crucial factor to consider in blockchain systems is their immutability. Once signatures are uploaded, they remain permanently etched in the system. This implies that over time, the authorship of a linkable or traceable ring signature [50, 27] could potentially be unveiled due to inadequacies in the underlying computational assumptions.[2] Therefore, for optimal applicability in blockchain governance, the assurance of unconditional anonymity (i.e., the anonymity does not rely on any hardness assumptions) is one of the imperative features of a voting system in blockchain.

[2] Even assuming unconditionally anonymous linkable ring signatures [48, 18], the resulted protocol still suffers from a quadratic signature size.

Traceability. If a malicious voter attempts to vote twice, we need to have an efficient mechanism to trace its identity. Note that this traceability property is a relaxation/fallback of the above strong correctness to allow for more practical constructions. Indeed, traceability is irrelevant if double-voting is infeasible.

Receipt-freeness. It should be impossible to generate a proof that a voter indeed cast an intended vote for others to see. This property has been widely studied [2, 3, 10, 23, 24, 35, 37, 39, 40, 45, 46, 47, 52, 54, 59].

Cryptographic Mechanisms for Blockchain Voting/Governance Systems

In pursuing our objective to design a voting system that ensures traceability and unconditional anonymity, we encounter inherent challenges when leveraging traditional cryptographic tools such as signature schemes or Multi-Party Computations (MPC).

Linkable Ring Signatures (LRS). Linkable ring signatures (LRS) [50] are a prevalent tool in constructing e-voting systems. In LRS, a user can sign a message on behalf of a group/ring while maintaining anonymity. Moreover, if a user signs twice for the same message and on behalf of the same group, these signatures are "linked", making it evident that they originate from a single actual signer. This inherent linkability property of LRS plays a vital role in preventing double-voting. However, it's important to note that the link algorithm in LRS does not disclose the specific identity of the signer when two signatures are linked. Consequently, LRS does not fulfill the traceability property, implying that a malicious voter engaging in double-voting may go unpunished. This lack of traceability can undermine the system's balance and fairness, necessitating additional measures to ensure accountability and uphold the integrity of the voting process.

Traceable Ring Signature. An alternative approach is to employ traceable ring signatures [27], where the link algorithm establishes the link between signatures and discloses the signer's identity. However, unconditional anonymity is compromised in traceable ring signatures, as proved in [18]. Striking a balance between traceability and unconditional anonymity in ring signatures appears challenging, presenting a fundamental trade-off within this cryptographic context.

Multi-Party Computation. Multi-party computation (MPC) [64, 31, 9, 15] appears to be the ultimate solution to the above voting problem. MPC allows n parties to compute any given function on their inputs securely so that no malicious party (or coalition) can learn the inputs of other parties (privacy), and no party can affect the output any more than choosing their own input. MPC can directly be used to realize our voting functionality by having each voter submit their votes and output the appropriate tally. MPC-privacy (which can be information-theoretic [9, 15, 57]) will ensure unconditional anonymity; MPC-correctness (for the appropriate function) can ensure our above voting correctness property. Traceability is a more elusive goal, but it can also be achieved by so-called identifiable MPC [38] (which ensures that upon aborting the identity of a cheater is revealed).[3]

Unfortunately, despite its very general functionality, MPC is also not the right solution to our problem: For starters, information-theoretic MPC needs an honest majority of the parties [31], an assumption which is unrealistic in our setting.[4] Even if one is willing to

[3] In our blockchain governance application, we need a property which is stronger than identifiability, namely *public verifiability*, which informally ensures that an abort provides a cheating certificate that can be verified even by a non-MPC party later on, e.g., [4].

[4] Although there are solutions which replace the honest majority of parties assumption with an assumption on the resource distribution, e.g., honest majority of hashing-power or stake [28], they come at a high cost in terms of blockchain utilization – multiple on-chain rounds – which renders them mainly of theoretical interest.

resort to security with (identifiable) abort and no fairness,[5] we still need to implement Byzantine broadcast – which requires in the worst case a polylogarithmic in the party-set size – or use again several on-chain rounds. Even given broadcast, turning an MPC protocol identifiable above into one with guaranteed output delivery (which is needed in our application) would require restarting the computation whenever it aborts (and potentially removing the identified cheater); this would again yield a larger number of rounds which is undesirable.

From Signature Schemes to an Interactive Structure

The above discussion highlights the inherent challenge of satisfying all requirements simultaneously. However, despite this seemingly conflicting nature, there is a way to address it. In this work, we provide a positive answer by proposing a structured approach embedded in the signature generation process. We carefully define an interactive structure – a simple protocol – within the signature generation. This structured approach is a key innovation that enables us to design a signature scheme that simultaneously conceals individual identities while revealing the number of signers involved. This dual property is a critical advancement for secure and reliable e-voting systems. We call this new signature protocol Sharp Anonymous Multisignatures (♯AMS). The term "sharp" is used in analogy with complexity theory, as in $\sharp P$, indicating that the signatures output the number of valid signers rather than a mere validity bit. Meanwhile, "anonymous" denotes that the signers' identities remain hidden. Equipped with these properties, ♯AMS is a perfect fit for e-voting systems, especially in the context of blockchain governance. We discuss the details of our contributions below.

1.1 Our Contributions

♯ Anonymous MultiSignatures

Our first and major contribution is proposing and formalizing a new concept of signing *protocol*, dubbed ♯AMS that tightly meets the needs of blockchain governance. The protocol allows any set of parties to collaborate jointly and outputs unconditionally anonymous signatures. To compare with threshold ring signatures (TRS), ♯AMS does not need the threshold and always generates a valid signature regardless of the number of parties, and the verification algorithm reveals the number of parties. Regarding the number of parties as a threshold, which may vary every signing, this can be seen as a strict generalization of TRS.

Generic Compiler from TRS with A Flexible Threshold

Despite the above separation of TRS and ♯AMS, it turns out that several instantiations of TRS actually possess a *flexible threshold* property, i.e., these TRSs can change the threshold depending on the actual number of signers.

To characterize the class of TRSs that admit such a lifting to ♯AMSs, we provide a generic template for TRSs that (1) abstracts many such "liftable" schemes, and (2) admits a generic compiler to transform to ♯AMS. Several existing TRS constructions can be seen as instantiations of our template, which implies that those TRS schemes are more versatile than previously known.

[5] This is already a discount in security that should be avoided.

♯AMS Constructions from Chameleon Hashes

Using the above, we provide concrete ♯AMS schemes from (lossy) chameleon hashes in a black-box model – which covers the post-quantum case. In a nutshell, we propose the following types of constructions:

C1. A basic construction with three communication rounds. This construction achieves unconditional anonymity.

C2. A fault-tolerant variant of C1 – for an arbitrary number of corruptions – without any overhead. This construction achieves unconditional anonymity and public verifiability.

Voting Systems from ♯AMS Constructions

Since anyone can verify the number of signers from a ♯AMS signature, we can immediately turn the constructions into voting systems. Furthermore, we develop a *conditioned key generation paradigm* to enable the single-vote setting.

V1. A basic system implemented by C2. This allows multiple voting. The system can tolerate malicious voters (those who claimed to vote but later quit). Including the posting period and the announcement period, voting can be completed within two on-chain blocks.

V2. A round-optimal system implemented by C1 in the multiple-vote setting by leveraging one-time key generation. This system satisfies the "vote-and-go" property.

V3. A variant of V2 for the single-vote setting, based on the conditioned key generation paradigm. V3 requires one additional on-chain round when there are malicious users. However, if the maximum number of proposals is known in advance, the voting process still requires only two on-chain blocks, as in V1 and V2.

Notably, our voting systems possess several desirable properties, which are discussed in depth in the full version [19]. We briefly introduce these properties here:

On-Chain. The vote will be uploaded on the blockchain and cannot be altered further. All our systems achieve this property.

Vote-and-go. Voters can leave immediately after casting their vote (by sending their cryptographic information on the vote) without any interaction. V2 and V3 achieve this property.

Vote-Count Concealment. Voters remain unaware of the current vote count (and therefore of the votes of others) until the results are announced. V2 and V3 achieve this property.

▶ Remark 1 (On receipt-freeness). We would like to mention that our protocols do not satisfy receipt-freeness properties due to their simple structure. Any voting scheme that requires voters to choose their own randomness, like ours, is vulnerable to a well-known generic receipt-freeness attack [37]. In such an attack, a malicious voter can generate randomness using a one-way function and later provide the input to the one-way function to prove the source of the randomness. To resolve this problem, one would need to rely on a much more complex system (e.g., [37] assumes homomorphic encryption, an honest assumption on the authorities, and many other assumptions). While this is an intriguing and interesting open problem, we are focusing on proposing a new signature protocol, ♯AMS, and thus leave this problem for future research.

A Framework of Lossy Chameleon Hashes

As a by-product, we explore a framework of lossy chameleon hashes, including its relationship with existing cryptographic primitives (e.g., lossy identification and lossy encryption) and more concrete constructions from various assumptions. See Fig. 1 for details.

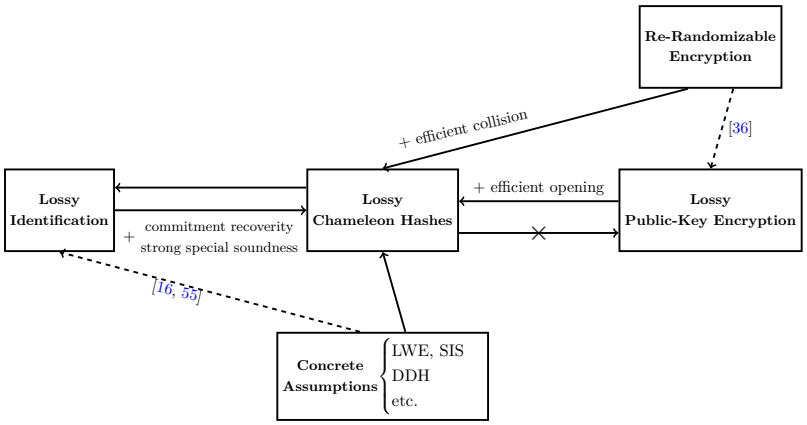


Figure 1 A framework for lossy chameleon hashes (solid arrows: shown in this work; dashed line: shown in previous works [16, 55, 36]).

1.2 Related Works

We refer to the full version [19] for more related works on multisignatures, threshold signatures, and ring signatures.

Threshold Ring Signatures. ♯AMS in this paper are highly related to threshold ring signatures (TRS) [11]. An (n, t)-TRS scheme allows t or more members to generate a ring signature together, and the actual signers remain anonymous. The verification algorithm in TRS will output either 0 or 1, indicating the validity w.r.t. the threshold t. Differently, by defining ♯AMS, we emphasize the property that the verification outputs exactly the credibility of the signature, i.e., how many users have participated in the generation. This holds even against malicious users who behave wantonly when signing. For example, a malicious user may quit at the middle or contribute senseless results in the signing process. Therefore, ♯AMS is strictly stronger than TRS.

There are many threshold ring signature schemes [49, 14, 63, 20] whose threshold t is changeable every signing. By adding t into the message to be signed, a TRS scheme with a flexible threshold can be tuned to a ♯AMS scheme.

Graded Signatures. Kiayias, Osmanoglu, and Tang [43] proposed the concept of graded signatures that allow a combiner/user to combine a set of different signatures w.r.t. the same message to a "consolidated" signature. Meanwhile, the consolidated signature leaks the actual number of signatures used to consolidate the graded signature and nothing else, which is the same as a ♯AMS scheme. However, graded signatures [43] have two phases in signing, which means that every participant has to sign by themselves and then pass their signature share to someone for consolidation. Due to such a definition, graded signatures cannot achieve unconditional anonymity. Therefore, ♯AMS provides more robust security.

Meanwhile, a trusted authority is required in graded signatures for generating global key pairs. The concrete instantiation in [43] relies on building blocks like structure-preserving signatures and Groth-Sahai proofs [33] and consequently cannot achieve post-quantum security.

Blockchain Governance and E-voting. Beck et al. [5], Pelt et al. [60], and Kiayias and Lazos [42] discussed the core properties for blockchain governance in multiple disciplines. Khan et al. [41], Venugopalan and Homoliak et al. [61], and Gersbach et al. [30] also focused on blockchain governance, especially based on decision-making processes and voting in terms of game-theoretical analyses.

Many papers in various fields have studied e-voting systems that combine blockchain and cryptographic primitives to realize distributed and decentralized voting systems. However, the cryptography community, which is the one that can technically contribute the most, has less interest in the topic. A few papers introduce e-voting systems derived from advanced cryptographic primitives such as linkable ring signatures. In terms of ring signatures, Lyu et al. [51] and Russo et al. [58] present systems that use blockchain, smart contracts, and ring signatures. Most of the other blockchains' e-voting systems rely not only on simple cryptographic primitives but also heavily on complex functionalities, e.g., smart contracts and zero-knowledge proofs. Note that our proposed solutions can simplify their approaches much since ♯AMS provides not only privacy and authenticity but also signatures with the corresponding number of signers in the primitive level.

1.3 Technical Overview

This subsection briefly overviews the techniques and concepts used in this paper.

Formalization of ♯AMS

We start from formalizing ♯Anonymous Multisignature (♯AMS) and its security definitions. Suppose a group of t users, $\{U_i\}_{i\in G}$, wants to sign a message msg together, and the resulting signature σ reveals only the number of participants t and nothing else. We refer to t as the credibility of the signature.

The first issue is the anonymity property, which means that the identities of the t actual signers are hidden among the total of n users ($n \geq t$). Besides, as a (group) signature scheme, ♯AMS should ensure unforgeability; namely, any adversary controlling fewer than t users cannot forge a valid signature on a new message that shows a credibility of t.

As we discussed above, if every signer U_i contributes their share using their own secret key and the signature is simply a concatenation of different shares (as in linkable/traceable ring signatures), it seems impossible to achieve both unconditional anonymity and traceability [18].[6] Therefore, we focus on interactive signing processes and introduce a moderator P in the signing protocol. To generate a ♯AMS signature, each signer communicates only with P and not with other signers. It is P's responsibility to count the number of participants and finally produce a ♯AMS signature. In this case, even a signer cannot de-anonymize a ♯AMS signature (i.e., they remain unaware of the other participants). P can be a member of G or not, and is assumed to be honest. We believe this is a simple yet reasonable assumption; see more discussion in Section 3. Even if a malicious P leaks the quorum of signers later, their own identity would be revealed at the same time, and consequently, P would face punishment from the system.

[6] In the application of voting systems, if double voting is feasible, then traceability is necessary.

Generic Compiler from Threshold Ring Signatures

A ♯AMS scheme directly implies a threshold ring signature (TRS) scheme since the verification algorithm Ver in ♯AMS returns the real number of participants in the signing process, while Ver in TRS returns only a single bit. Therefore, the definition of ♯AMS is stronger than that of TRS.

Nevertheless, we surprisingly notice that many TRS schemes (e.g., [49, 14, 13, 62, 34]) possess a *flexible threshold* property. Namely, the threshold t does not need to be fixed when generating the public/secret key pairs; it can be changed in every signing session. Motivated by this, we design a generic compiler that tunes any TRS scheme with a flexible threshold into a ♯AMS scheme:

- A participant P among the signers is selected randomly as the moderator in ♯AMS.
- P knows the quorum of signers, hence the number t. To generate a ♯AMS signature on message msg, it starts the TRS signing protocol on the message $(\mathsf{msg}||t)$ and outputs the TRS signature $\tilde{\sigma}$ and t as the final ♯AMS signature.
- In the verification of ♯AMS, if $\tilde{\sigma}$ is valid in TRS, then the threshold t is returned.

Thanks to this compiler and the rich literature on TRS, we immediately obtain many ♯AMS schemes from well-studied TRS schemes, based on the DL assumption [14], the RSA assumption [49], the SIS assumption [13], the code assumption [22], and others.

C1: Construction from (Lossy) Chameleon Hashes

A chameleon hash (CH) function [44] is a special hash function indexed by a hash key hk, which is associated with a trapdoor td. It takes as input two parts: a message m[7] and randomness r. On the one hand, given only the hash key, it is hard to find a collision. On the other hand, with the help of the trapdoor, finding collisions becomes easy.

Chameleon hashes can be converted into signature schemes via the well-known Fiat-Shamir paradigm [26], where hk and td serve as the public key and the signing key, respectively. To sign a message msg, the signer first randomly samples dummy values $\bar{m}$ and $\bar{r}$, and then uses its trapdoor to find a randomness r for m so that it collides with $(\bar{m}, \bar{r})$ on h, where $m = H(\mathsf{msg}, h)$ with $H(\cdot)$ a random oracle and h the chameleon hash value of $(\bar{m}, \bar{r})$.

Now, we extend this idea to the ♯AMS setting. Suppose there are n users, and each user U_i has its own hash key hk_i and trapdoor td_i. We borrow the idea of the t-out-of-n zero-knowledge proof from [21] by Cramer, Damgård, and Schoenmakers. That is, given n random values m_i, the group of signers can find t random values r_i that create collisions. For the message msg, we design a ♯AMS signature in the form of $\sigma = (t, m_1, \ldots, m_n, r_1, \ldots, r_n)$, where $m_1, \ldots, m_n$ are required to satisfy the following linear equations.

$$\begin{cases} a_{1,1}m_1 + a_{1,2}m_2 + ... + a_{1,n}m_n = u_1, \\ a_{2,1}m_1 + a_{2,2}m_2 + ... + a_{2,n}m_n = u_2, \\ ... \\ a_{t,1}m_1 + a_{t,2}m_2 + ... + a_{t,n}m_n = u_t. \end{cases} \tag{1}$$

Here, the coefficients $(a_{i,j})$ are public parameters, and $(u_1, \ldots, u_t)$ are the outputs of hash function $H(h_1, \ldots, h_n, \mathsf{msg}, t)$ (modeled as a random oracle), with $(h_1, \ldots, h_n)$ being the corresponding chameleon hash (CH) values. To satisfy Eq. (1), at least t trapdoors are necessary to determine the corresponding randomness r_i, thereby proving that at least t users have participated in the signing process.

[7] Note that the message m in chameleon hashes is different from the message msg to be signed in signature schemes. Here, we slightly abuse the terminology to keep consistent with previous works.

Instead of using the above linear equations, we can also set a polynomial g of degree at least $n - t$ and require that

$$g(0) = u, \quad g(i) = m_i \text{ for all } i \in [n], \text{ where } u = H(h_1, \ldots, h_n, \mathsf{msg}, t). \tag{2}$$

The security of the CH-based signature scheme, as well as our ♯AMS scheme, is based on the collision-resistance property of the underlying chameleon hash scheme. However, the reduction in the security proof is not black-box since it relies on the forking lemma [7] to find a collision. Inspired by the ideas of lossy trapdoor functions [56] and lossy encryption [6, 36], we define lossy chameleon hashes (LCH) in this work to enable a black-box reduction in the security proof. An LCH scheme operates in two modes: the collision mode, which behaves as a standard CH, and the lossy mode, where a lossy hash key is used instead of a normal hash key, and the adversary (even if computationally unbounded) cannot find randomness for a random message that hashes to a previously chosen hash value.

We present an efficient instantiation of LCH from the learning with errors (LWE) assumption and the short integer solution (SIS) assumption. We also explore the relationship between LCH and other primitives, including lossy identification schemes [1], re-randomizable encryption, and lossy encryption [6, 36]; see Fig. 1 and more details in the full version [19].

C2: Fault-Tolerant Variant

For better application in blockchain governance, we consider a variant of the above construction in the faulty signer setting. A faulty signer may quit in the middle of the protocol or return malicious results to the moderator. To address this, we develop a fault-tolerant variant of our ♯AMS scheme. This is possible because our (L)CH-based ♯AMS scheme has "backward compatibility": namely, even if a voter quits in the middle, by exposing the identity of the faulty voter, the moderator can still output a signature with credibility $(t - 1)$. More precisely, let F be the set of faulty signers. In the fault-tolerant ♯AMS scheme, we tailor the signature into the form

$$\sigma' := \left(t, F, \{m_i, r_i\}_{i \in [n] \setminus F}, \{h_i\}_{i \in F}\right).$$

If $\{m_i\}_{i \in [n]}$ satisfy the linear equation (1) or (2), then the verification algorithm will output $(t - |F|)$ instead of t. Consequently, the voters in F will lose their anonymity as a cost of their malfeasance.

V1: Blockchain Governance via ♯AMS Schemes

♯AMS directly implies a voting protocol since a ♯AMS signature reveals the number of signers. The protocol consists of the following four periods using scheme C2:

1. The posting period, in which each developer publishes their improvement proposal on the blockchain.
2. The declaration period, in which each voter (a signer in ♯AMS) signals their willingness to support a proposal by sending a hash value of a dummy message and randomness to the developer.
3. The signing period, in which the developer performs the signature algorithm based on the number of supporters. Specifically, developers compute all m_i values according to the received h_i and Eq. (1) or (2). They then return the m_i values to the supporters, who use their trapdoors to find collisions r_i and send them back to the developer. This helps the developer complete the generation of the ♯AMS signature.

4. The announcement period, in which each developer uploads their ♯AMS signature to the blockchain. The voting result is then included in the block and published across the network.

Our protocol enables multiple improvement proposals to compete for votes, and the proposal with the most votes is elected.

To generate a ♯AMS signature, the developer needs three rounds of interaction with its supporters during the signing period. This opens the door to faulty attacks by malicious voters: for instance, a voter may send h_i and claim to participate, but then abort or send incorrect randomness r' after receiving m from the developer. Thanks to our fault-tolerant ♯AMS scheme, such attacks do not compromise the entire voting system, since a (fault-tolerant) ♯AMS signature always reveals the true number of (honest) participants.

V2: Round Optimization

Notice that in the protocol above, voters must wait for the message m_i from the developer after declaring their support. To achieve the "vote-and-go" property, we further optimize the protocol to a single round. Our idea is to use ♯AMS in a one-time paradigm. That is, voters U_i, regardless of whether they intend to vote for a proposal, generate a new hash key and trapdoor pair $(hk_i^{(j)}, td_i^{(j)})$ for some proposal by developer P_j. Then, supporters of P_j secretly send their trapdoors to P_j using a standard public-key encryption scheme. With knowledge of all secret keys, P_j can generate a ♯AMS signature without further interaction with the supporters. To prevent a malicious developer from generating one-time hash keys on their own, we additionally require every hash key to be accompanied by a signature proving the authority of its owner.

V3: Single-Vote Setting via the Conditioned Key Generation Paradigm

We further extend the above protocol to the single-vote setting, where each user in the voting system can vote for at most one proposal. Since there are multiple developers P_j, each with their own proposal, every voter U_i now needs to generate multiple key pairs $\{(hk_i^{(j)}, td_i^{(j)})\}$, where the superscript (j) indicates the key pair is for the j-th proposal.

The protocol as previously described does not work in the single-vote setting because a malicious voter could vote on different proposals using different trapdoors. We employ the same method used in constructing ♯AMS to prevent multiple votes. Recall that to vote on a proposal $IP^{(j)}$ (i.e., participate in the signing process for that proposal), user U_i must possess the trapdoor $td_i^{(j)}$ corresponding to the hash key $hk_i^{(j)}$ specifically designed for $IP^{(j)}$. Suppose there are p proposals $(IP^{(j_1)}, ..., IP^{(j_p)})$, then U_i must generate p hash keys $(hk_i^{(j_1)}, ..., hk_i^{(j_p)})$. We apply a so-called *conditioned key generation paradigm*, which sets a restriction among the p hash keys so that the voter can know at most one trapdoor.

Let $\mathcal{HK}$ be the space of hash keys for (lossy) chameleon hashes, and let $\mathbf{B} = (b_{i,j}) \in \mathcal{HK}^{(p-1)\times p}$ be a matrix of full rank. Define $\mathbf{hk}_i^\top = (hk_i^{(j_1)}, ..., hk_i^{(j_p)})^\top$. We require $\mathbf{B} \cdot \mathbf{hk}_i = \hat{\mathbf{hk}}_i$, where $\hat{\mathbf{hk}}_i \in \mathcal{HK}^{p-1}$ is the output of some hash function $H(IP^{(j_1)}, ..., IP^{(j_p)}, i)$.

If $H(\cdot)$ behaves as a random oracle, then to satisfy the above equation, every user can know at most one trapdoor among the total p hash keys. Consequently, the single-voting property is achieved.

1.4 Roadmap

This paper is organized as follows. In Section 2, we introduce the definitions of (lossy) chameleon hashes. The definition and security properties of ♯AMS are formally described in Section 3. We provide a generic transformation for most threshold ring signature (TRS) schemes in Section 4. In Section 5, we propose an efficient construction of ♯AMS from (lossy) chameleon hashes ((L)CH). Section 6 presents three voting systems and their applications in blockchain governance. We refer to the full version [19] for more details, including basic notations, cryptographic primitives, more security notions of ♯AMS and formal proofs.

2 Preliminaries

▶ **Definition 2** (Chameleon Hashes [44]). *A chameleon hash (CH) scheme consists of the following three algorithms. Namely* $\mathsf{CH} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{TdColl})$.

- $(hk, td) \leftarrow \mathsf{Gen}(1^\lambda)$*: The key generation algorithm takes as input the security parameter* 1^λ*, and outputs a hash key* hk *and a trapdoor* td*. W.l.o.g., we assume* hk *implicitly determines the message space* $\mathcal{M}$*, the randomness space* $\mathcal{R}$*, and the hash space* $\mathcal{H}$*.*
- $h \leftarrow \mathsf{Hash}(hk, m, r)$*: The hash algorithm takes as input* hk*, a message* $m \in \mathcal{M}$ *and a randomness* $r \in \mathcal{R}$*, and outputs the hash value* $h := \mathsf{Hash}(hk, m, r)$*.*
- $r' \leftarrow \mathsf{TdColl}(td, m, r, m')$*: The trapdoor collision algorithm takes as input the trapdoor* td*, a message-randomness pair* (m, r) *and another message* m'*, and outputs* r' *such that* $\mathsf{Hash}(hk, m, r) = \mathsf{Hash}(hk, m', r')$*.*

We introduce lossy chameleon hashes (LCH) as follows.

▶ **Definition 3** (Lossy Chameleon Hashes). *A lossy chameleon hash (LCH) scheme consists of four algorithms,* $\mathsf{LCH} = (\mathsf{Gen}, \mathsf{LGen}, \mathsf{Hash}, \mathsf{TdColl})$*, where* $\mathsf{Gen}, \mathsf{Hash}, \mathsf{TdColl}$ *are defined as in Def. 2, and* LGen *is defined as follows.*

- $hk \leftarrow \mathsf{LGen}(1^\lambda)$*: The lossy key generation algorithm takes as input the security parameter* 1^λ*, and outputs a lossy hash key* hk*.*

We explore a detailed framework of LCH in the full version [19].

3 Sharp Anonymous Multisignatures

In this section we formally define sharp anonymous multisignatures and their security notions. Let n be the total number of signers, $G \subseteq [n]$ be a group of signers, and $t = |G|$.

▶ **Definition 4** (Sharp Anonymous Multisignatures (♯AMS)). *A sharp anonymous multisignature (♯AMS) scheme* $\sharp\mathsf{AMS} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *consists of the following three algorithms/protocols:*

- $(vk, sk_1, ..., sk_n) \leftarrow \mathsf{Gen}(1^\lambda, n)$*. The key generation algorithm* Gen *takes as input the security parameter* λ *and the number of signers* n*, and outputs a verification key* vk*, and signing keys* $(sk_1, ..., sk_n)$ *for different signers. W.l.o.g., we assume that* vk *is implicitly contained in every* sk_i*.*
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, P, G \subseteq [n], \{sk_i\}_{i \in G})$*. The signing protocol takes place between a moderator* P *and a group of signers* $G \subseteq [n]$*, where* P *takes as input* vk *and the message* msg*, and each signer* $U_i \in G$ *takes* msg *and its own secret key* sk_i *as input. The moderator* P *can be a member of* $[n]$ *or not. Finally,* P *outputs a signature* σ*.*
  If we focus solely on the algorithmic properties of the signing process, we will ignore the moderator P *by denoting it as* $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, G \subseteq [n], \{sk_i\}_{i \in G})$*.*

- $t \leftarrow \mathsf{Ver}(vk, \mathsf{msg}, \sigma)$. *The verification algorithm* Ver *takes as input the verification key* vk*, a message* msg *and a signature* σ*, and outputs a number* $t \in [n] \cup \{0\}$*, indicating the number of signers for this signature.*[8]
  We say a signature σ *(w.r.t. a message* msg*) is* t*-valid (resp., invalid), if* $\mathsf{Ver}(vk, \mathsf{msg}, \sigma) = t$ *(resp.,* $\mathsf{Ver}(vk, \mathsf{msg}, \sigma) = 0$*).*

Correctness. *For any* $(vk, sk_1, ..., sk_n) \leftarrow \mathsf{Gen}(1^\lambda, n)$*, any message* msg*, any group* $G \subseteq [n]$ *of honest signers, any honest moderator* P*, and* $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, P, G, \{sk_i\}_{i \in G})$*, it holds that* $\mathsf{Ver}(vk, \mathsf{msg}, \sigma) = |G|$.

In the above definition, we assume all users' keys are generated via a centralized algorithm Gen, which is more generalized and covers the case where there is a trusted setup. In this work, we focus on the non-trusted setup case, and each user samples its own key pair and then publishes the public key. Moreover, we focus on cryptographic property of ♯AMS and do not consider the details of the signing protocol, e.g., adaptive corruptions during the running, protocol rounds, and just provide a proof-of-concept use of ♯AMS.

We require the unforgeability and anonymity for the security of ♯AMS.

- **Unforgeability**. The adversary that controls less than t participants cannot forge a signature that is t-valid.
- **Anonymity**. From a signature the adversary learns nothing about the quorum of the signers G that contributed to the signature, except the size $|G|$.

We formalize the security definitions via the following security experiments.

▶ **Definition 5** (Unforgeability of ♯AMS)**.** *Consider the following unforgeability experiment* $\mathsf{Exp}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *between the challenger* $\mathcal{C}$ *and the adversary* $\mathcal{A}$*.*

1. $\mathcal{A}$ *sets the maximum number of signers* n*.*
2. $\mathcal{C}$ *generates* $(vk, sk_1, ..., sk_n) \leftarrow \mathsf{Gen}(1^\lambda, n)$ *and passes* vk *to* $\mathcal{A}$*.*
3. $\mathcal{A}$ *has access to two oracles* $\mathcal{O}(\cdot, \cdot, \cdot)$ *and* $\mathcal{O}_{corr}(\cdot)$*. Here the signing oracle* $\mathcal{O}(\mathsf{msg}, P, G)$ *returns* $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, P, G, \{sk_i\}_{i \in G})$ *(with* P *the moderator) and adds* (msg, σ) *into the set* $\mathcal{S}$*. The corruption oracle* $\mathcal{O}_{corr}(i)$ *returns* sk_i*.*
4. *Finally* $\mathcal{A}$ *outputs* $(\mathsf{msg}^*, \sigma^*)$*.*

Let $t^* \leftarrow \mathsf{Ver}(vk, \mathsf{msg}^*, \sigma^*)$. $\mathsf{Exp}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *outputs* 1 *if*
(1) $t^* > t'$*, where* t' *is the total number of queries to* $\mathcal{O}_{corr}(\cdot)$*; and*
(2) $\mathcal{A}$ *never asks* $\mathcal{O}(\mathsf{msg}^*, P, G)$ *such that* $|G| = t^*$*.*

Define by $\mathsf{Adv}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *the probability that* $\mathsf{Exp}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *outputs 1. We say that* ♯AMS *is unforgeable, if for all PPT adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *is negligible in* λ*.*

▶ Remark 6 (On the Formalization of Unforgeability). One might wonder why we require "$\mathcal{A}$ never asks $\mathcal{O}(\mathsf{msg}^*, P, G)$ s.t. $|G| = t^*$" at the end of the experiment. Intuitively, a more "reasonable" definition should allow $\mathcal{A}$ to win, if it asks $\mathcal{O}(\mathsf{msg}^*, P, G)$ with $|G| = t^*$, and later forges a t^*-valid signature σ^* from a different set $G' \neq G$. However, since the identities are hidden from the signature, the challenger cannot detect whether σ^* comes from a group G' that is different from G (i.e., whether $\mathcal{A}$ wins in a non-trivial way). Therefore, to prevent trivial attacks, in Def. 5, $\mathcal{A}$ is prohibited to query $\mathcal{O}(\mathsf{msg}^*, P, G)$ with $|G| = t^*$.

[8] Note that vk contains information of $[n]$ so we omit $[n]$ from the input of Ver, which will be explicitly denoted for TRS schemes because of consistency from previous works.

▶ **Definition 7** (Strong Unforgeability). *Consider the strong unforgeability experiment* $\mathsf{Exp}^{s\text{-}unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$*, which is defined as* $\mathsf{Exp}^{unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *in Def. 5, except that condition* (2) *is replaced with*

(2') $(\mathsf{msg}^*, \sigma^*) \notin \mathcal{S}$.

Define by $\mathsf{Adv}^{s\text{-}unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *the probability that* $\mathsf{Exp}^{s\text{-}unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *outputs 1. We say that* ♯AMS *is strongly unforgeable, if for all PPT adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{s\text{-}unforg}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *is negligible in* λ.

In the full version [19] we also define the weak unforgeability and (strong/weak) unforgeability under static corruptions.

Now we formally define the strong anonymity of ♯AMS.

▶ **Definition 8** ((Unconditionally) Strong Anonymity of ♯AMS). *Consider the following strong anonymity experiment* $\mathsf{Exp}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *between* $\mathcal{C}$ *and* $\mathcal{A}$.

1. $\mathcal{A}$ *sets the maximum number of signers* n.
2. $\mathcal{C}$ *generates* $(vk, sk_1, ..., sk_n) \leftarrow \mathsf{Gen}(1^\lambda, n)$ *and passes* vk *to* $\mathcal{A}$*. Meanwhile,* $\mathcal{C}$ *randomly samples a bit* $b \xleftarrow{\$} \{0,1\}$.
3. $\mathcal{A}$ *has access to two oracles* $\mathcal{O}(\cdot,\cdot,\cdot,\cdot,\cdot)$ *and* $\mathcal{O}_{corr}(\cdot)$*, where the signing oracle* $\mathcal{O}(\mathsf{msg}, P_0, G_0, P_1, G_1)$ *returns* $\sigma \leftarrow \mathsf{Sign}(\mathsf{msg}, P_b, G_b, \{sk_i\}_{i\in G_b})$ *if* $|G_0| = |G_1|$ *and* $\perp$ *otherwise, and the corruption oracle* $\mathcal{O}_{corr}(i)$ *returns* sk_i.
4. *Finally* $\mathcal{A}$ *outputs* b'.

$\mathsf{Exp}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *outputs 1 if* $b' = b$*. Let* $\mathsf{Adv}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \Rightarrow 1] - 1/2|$*. We say that* ♯AMS *has strong anonymity (resp., unconditional and strong anonymity) if for all PPT (resp., computationally unbounded) adversary* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda)$ *is negligible in* λ.

The aforementioned definition requires that anonymity holds even if all secret keys are leaked (that is why we formalize it as "strong" anonymity). In the above definition, if we restrict that $\mathcal{A}$ cannot ask $\mathcal{O}_{corr}(i)$ with $i \in (G_0 - G_1) \cup (G_1 - G_0)$ for all (G_0, G_1) involved in $\mathcal{O}(\cdot,\cdot,\cdot,\cdot,\cdot)$, then we define the anonymity where it might be easy to decide whether a signer i has participated in the generation of a signature with the knowledge of sk_i. This is somewhat similar to the so-called *culpability* property in [50]. In other words, one signer is able to claim the authorship of some (ring) signature by revealing its secret key (and some other private information, if necessary) to the public.

4 Generic Compiler of ♯AMS from TRS with A Flexible Threshold

In this section, we provide a generic transformation for threshold ring signature (TRS) schemes with flexible threshold, which implies that such TRS schemes are more versatile than their original definitions. We refer to the full version [19] for the definition and security of TRS, and a detailed discussion for the relationship between TRS and ♯AMS.

Generic compiler from TRS with flexible threshold to ♯AMS. Here, we show the most generalized version of compilers without considering optimization. We will take a deeper look at specific cases in the next section. Let $\mathsf{TRS} = (\mathsf{Gen}, \mathsf{TSign}, \mathsf{Ver})$ be a TRS scheme with a flexible threshold. We design ♯AMS scheme ♯AMS as follows.

- $\mathsf{Gen}(1^\lambda, n)$. For $i = 1, ..., n$, invoke $(pk_i, sk_i) \leftarrow \mathsf{TRS.Gen}(1^\lambda)$. Return $vk := (pk_1, ..., pk_n)$ and $\{sk_i\}_{i\in[n]}$.

- $\mathsf{Sign}(\mathsf{msg}, P, G \subseteq [n], \{sk_i\}_{i\in G})$. Every signer U_i where $i \in G$ first sends a HELLO message to the moderator P so that P will know the number of signers $t := |G|$. Then, the t signers run the threshold signing protocol $\mathsf{TRS.TSign}(\mathsf{msg}||t, [n], vk, G, \{sk_i\}_{i\in G})$ with P works as a moderator. Let $\tilde{\sigma}$ be the output of $\mathsf{TRS.TSign}$. Finally, P outputs $\sigma := (t, \tilde{\sigma})$ as a ♯AMS signature.[9]
- $\mathsf{Ver}(vk, \mathsf{msg}, \sigma)$. Parse $\sigma = (t, \tilde{\sigma})$. If $\mathsf{TRS.Ver}(t, [n], vk, \mathsf{msg}, \tilde{\sigma}) = 1$ then output t. Otherwise, output 0.

The security of ♯AMS constructed above inherits from the security of the underlying TRS scheme, and we omit the detailed proof here.

5 Constructions of ♯AMS from Lossy Chameleon Hashes

In this section we propose efficient constructions of ♯AMS from lossy chameleon hashes (LCH) in a black-box way.

5.1 Formalization of Constraint Functions

First we introduce the definition of constraint functions, an important tool in constructing the t-out-of-n proofs [21].

▶ **Definition 9** (Constraint Functions). *Let n, t be positive integers, $n \geq t \geq 1$, and $\mathcal{M}, \mathcal{U}$ be two finite sets. We call $\mathbf{F}_\theta : (\mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathcal{M}^n \times \mathcal{U}) \to \{0, 1\}$ a series of constraint functions indexed by θ, if it has the following properties.*

- *$\mathbf{F}_\theta(n, t, m_1, ..., m_n, u)$ is efficiently evaluated.*
- *There exists an efficient and deterministic algorithm $f(\cdot)$, such that $f(n, t, m_1, ..., m_n)$ outputs the unique u (if it exists) satisfying $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = 1$.*
- *For any $G \subseteq [n]$ and $|G| = t$, there exists two efficient sample algorithms s_{fwd} and s_{back} that both output $(m_1, ..., m_n, u)$, and the two distributions are identical.*
  - *Forward sample algorithm $s_{fwd}(n, t, G)$ first samples $\{m_i\}_{i\in[n]}$, and then computes u according to $\mathbf{F}_\theta$.*
  - *Backward sample algorithm $s_{back}(n, t, G)$ first randomly chooses $\{m_i\}_{i\in[n]\setminus G}$ and u, and then computes $\{m_i\}_{i\in G}$ according to $\mathbf{F}_\theta$.*
- *Interdependency.*
  - *If $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = \mathbf{F}_\theta(n, t, m'_1, ..., m'_n, u) = 1$, then either $(m_1, ..., m_n) = (m'_1, ..., m')$, or there are at least t different $i \in [n]$ such that $m_i \neq m'_i$.*
  - *For randomly sampled u and u', if $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = \mathbf{F}_\theta(n, t, m'_1, ..., m'_n, u') = 1$, then with overwhelming probability there are at least t different $i \in [n]$ such that $m_i \neq m'_i$.*
- *Randomness. Conditioned on $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = 1$, if u distributes uniformly, then*
  - *either there exist at least t different $i \in [n]$ such that m_i distribute uniformly, or*
  - *for any $i \in [n]$, m_i distributes uniformly.*

We refer to two instantiations of $\mathbf{F}_\theta$ from linear equations and polynomial interpolation in the full version [19].

[9] If $\tilde{\sigma}$ itself already contains a threshold t then P just outputs $\sigma := \tilde{\sigma}$.

5.2 C1: Interactive ♯AMS

Now, we describe our generic construction of ♯AMS from (lossy) chameleon hashes ((L)CH) as follows. We note that the underlying building blocks (LCH or CH) affect only the way of security proofs but not the construction itself.

Construction. Let $\mathsf{LCH} = (\mathsf{Gen}, \mathsf{LGen}, \mathsf{Hash}, \mathsf{TdColl})$ be a lossy chameleon hash scheme with message space $\mathcal{M}$ a field. Let $\mathbf{F}_\theta : (\mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathcal{M}^n \times \mathcal{U}) \to \{0,1\}$ be a constraint function indexed by θ. $H(\cdot) : \{0,1\}^* \to \mathcal{U}$ is a hash function that is modeled as a random oracle.

- $(vk, sk_1, ..., sk_n) \leftarrow \mathsf{Gen}(1^\lambda, n)$. For $i \in [n]$, invoke $(hk_i, td_i) \leftarrow \mathsf{LCH.Gen}(1^\lambda)$. Return $vk := (hk_1, ..., hk_n)$ and $\{sk_i\}_{i\in[n]} := \{td_i\}_{i\in[n]}$.
- $\mathsf{Sign}(\mathsf{msg}, P, G, \{sk_i\}_{i\in G})$.
  1. For every signer $i \in G$, it samples $\bar{m}_i \xleftarrow{\$} \mathcal{M}, \bar{r}_i \xleftarrow{\$} \mathcal{R}$, and sends $h_i \leftarrow \mathsf{Hash}(hk_i, \bar{m}_i, \bar{r}_i)$ to the moderator P.
  2. The moderator counts $t := |G|$ from the received messages.
  3. For $i \in [n] \setminus G$, P samples $m_i \xleftarrow{\$} \mathcal{M}, r_i \xleftarrow{\$} \mathcal{R}$ and computes $h_i \leftarrow \mathsf{Hash}(hk_i, m_i, r_i)$.
  4. P invokes $u \leftarrow H(vk, h_1, ..., h_n, \mathsf{msg}||t)$.[10] Then it computes $\{m_i\}_{i\in G}$ according to the backward sample algorithm $s_{back}(\cdot)$ of $\mathbf{F}_\theta$ such that

     $$\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = 1.$$

     For $i \in G$, P sends m_i to signer i.
  5. For $i \in G$, signer i computes $r_i \leftarrow \mathsf{LCH.TdColl}(td_i, \bar{m}_i, \bar{r}_i, m_i)$ and sends r_i to P.
  6. Finally P outputs the signature $\sigma := (t, \{m_i\}_{i\in[n]}, \{r_i\}_{i\in[n]})$.
- $\mathsf{Ver}(vk, \mathsf{msg}, \sigma)$. Parse $\sigma = (t, \{m_i\}_{i\in[n]}, \{r_i\}_{i\in[n]})$. Compute $h_i \leftarrow \mathsf{Hash}(hk_i, m_i, r_i)$ for all $i \in [n]$. Let $u \leftarrow H(vk, h_1, ..., h_n, \mathsf{msg}||t)$. Return t if $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = 1$, and 0 otherwise.

Generality of the Construction. Our construction exhibits strong generality. In Section 4, we have shown a generic compiler from TRS with a flexible threshold to ♯AMS. In fact, many existing TRS constructions can be categorized within the above framework due to the equivalence between chameleon hashes and Sigma protocols (identification schemes) [8] and the result (of (L)CH) in this work. For example, by instantiating with $\mathbf{F}_\mathbf{A}$ (see the full version [19]) and the DL-based CH [44], we get the TRS scheme in [14], and by instantiating with $\mathbf{F}_p$ and the DL-based CH, we get the TRS scheme in [49]. Thanks to this generic construction, we immediately get more schemes of ♯AMS (also TRS) from lattices [29, 12], isogenies [25], etc.

▶ **Theorem 10.** *If* LCH *is strongly secure (i.e., it has κ-uniformity, γ-random trapdoor collision, strong collision resistance, indistinguishability, and ϵ-lossiness) and unique, and $\mathbf{F}_\theta$ is a constraint function, then* ♯AMS *constructed above has strong unforgeability and unconditionally strong anonymity under static corruptions. More precisely, for any PPT adversary $\mathcal{A}$, there exist PPT algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$, such that* $\max(Time(\mathcal{B}_1), Time(\mathcal{B}_2)) \approx Time(A)$, *and*

[10] By including both msg and the threshold t into the hash input, the non-malleability is achieved in our signature scheme.

$$\mathsf{Adv}^{s\text{-}unforg\text{-}sta\text{-}corr}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq n(\mathsf{Adv}^{s\text{-}cr}_{\mathsf{LCH},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{ind}_{\mathsf{LCH},\mathcal{B}_2}(\lambda)) + n \cdot \epsilon + \frac{1}{|\mathcal{M}|} + \frac{Q_{sign} + Q_H}{2^{n\kappa}} + Q_{sign} n \cdot \gamma,$$

$$\mathsf{Adv}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq \frac{Q_{sign} n}{2} \cdot \gamma,$$

where Q_{sign} and Q_H are the numbers of signing queries (in the strong unforgeability experiment or the strong anonymity experiment) and hash queries, respectively.

We refer to the full version [19] for the proof.

Similarly, we get the following theorem for (regular) unforgeability.

▶ **Theorem 11.** *If* LCH *is secure (i.e., it has κ-uniformity, γ-random trapdoor collision, collision resistance, indistinguishability, and ϵ-lossiness) and $\mathbf{F}_\theta$ is a constraint function, then* ♯AMS *constructed above has unforgeability and unconditional strong anonymity under static corruptions. More precisely, for any PPT adversary $\mathcal{A}$, there exist PPT algorithm $\mathcal{B}$ such that $Time(\mathcal{B}) \approx Time(A)$, and*

$$\mathsf{Adv}^{unforg\text{-}sta\text{-}corr}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq n\mathsf{Adv}^{ind}_{\mathsf{LCH},\mathcal{B}}(\lambda) + n \cdot \epsilon + \frac{1}{|\mathcal{M}|} + \frac{Q_{sign} + Q_H}{2^{n\kappa}} + Q_{sign} n \cdot \gamma,$$

where Q_{sign} and Q_H are the numbers of signing queries and hash queries, respectively.

Security under Adaptive Corruptions and the Tightness of Unforgeability

We prove the security of unforgeability in the static corruption model. The proof can also be extended to the adaptive model (Def. 5 and 7), but it suffers from a large loss factor 2^n. We present the security bound under adaptive corruptions in the full version [19]. At a high level, to make use of the lossiness property of LCH, the challenger has to make sure that all hash keys of non-corrupted users are generated in the lossy mode. However, there is no corresponding trapdoor for a lossy hash key, which means that to deal with $\mathcal{A}$'s adaptive corruptions, the challenger has to decide the way of key generation very carefully so that it can offer the trapdoor of a user when a corruption happens on it. We also give another proof based on normal chameleon hashes in the full version [19].

5.3 C2: The Fault-Tolerating Variant

In this subsection, we consider faulty signers, i.e., signers who quit in the middle of signing or return faulty results to the moderator P. We propose a fault-tolerant variant of the (L)CH-based ♯AMS construction. In this variant, even if there exist faulty signers who behave maliciously, the moderator P can still output a ♯AMS that is t-valid, where t is the number of honest signers in the generation of that signature. Besides, the identities of those faulty signers are exposed, and anonymity holds only for the honest signers.

Faulty Attacks. We first introduce possible faulty attacks and then introduce a variant of (L)CH-based ♯AMS for tolerating faults while generating signatures. Note that our constructions need communication with a developer within a fixed period to ensure the number of total signers. This leads to the following faulty attacks. In the declaration period, the faulty node U_k faithfully follows the first two steps of the interaction. Then after receiving m_k from P, it crashes or sends a value instead of r_k.

Since P cannot compute r_k without td_k, it cannot produce a t-valid ♯AMS signature. Furthermore, since $\{m_j\}_{j\in G}$ are determined by $H(vk, h_1, ..., h_n, \mathsf{msg}||t)$ from $\{h_j\}_{j\in[n]}$ and $t = |G|$, P cannot generate a signature of $G \setminus \{k\}$ by simply discarding U_k.

Fault-Tolerant ♯AMS. We follow the same notion as the previous section. The Gen algorithm and the Sign protocol are also essentially the same as the previous one, but only the following treatment is included:

- In the signing protocol, if a signer U_i does not response for P's message m_i, or the signer U_i returns a wrong r_i such that $\mathsf{Hash}(hk_i, m_i, r_i) \neq h_i$, then P include i into the set F. Here h_i is the first message U_i sens to P (if U_i does not send h_i, then P would not include U_i into the signer group G).
  Finally, P outputs the fault-tolerated signature

$$\sigma := \left(t, F, \{m_i, r_i\}_{i\in[n]\setminus F}, \{m_i, h_i\}_{i\in F}\right).$$

- $t \leftarrow \mathsf{Ver}(vk, \mathsf{msg}, \sigma)$. Let $\sigma = \left(t, F, \{m_i, r_i\}_{i\in[n]\setminus F}, \{h_i\}_{i\in F}\right)$ and $h_i \leftarrow \mathsf{Hash}(hk_i, m_i, r_i)$ for $i \in [n] \setminus F$. Let $u \leftarrow H(vk, h_1, ..., h_n, \mathsf{msg}||t)$. Return $(t - |F|)$ if $\mathbf{F}_\theta(n, t, m_1, ..., m_n, u) = 1$ and 0 otherwise.

We show that the fault-tolerant ♯AMS scheme has weak unforgeability (against malicious signers) and unconditional strong anonymity (for honest signers).

▶ **Theorem 12.** *If* LCH *is secure (i.e., it has κ-uniformity, γ-random trapdoor collision, strong collision resistance, indistinguishability, and ϵ-lossiness) and $\mathbf{F}_\theta$ is a constraint function, then the fault-tolerant ♯AMS scheme above has weak unforgeability and strong anonymity under static corruptions. More precisely, for any PPT adversary $\mathcal{A}$, there exist PPT algorithms $\mathcal{B}_1$ and $\mathcal{B}_2$, such that* $\max(Time(\mathcal{B}_1), Time(\mathcal{B}_2)) \approx Time(A)$,

$$\mathsf{Adv}^{w\text{-}unforg\text{-}sta\text{-}corr}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq n(\mathsf{Adv}^{s\text{-}cr}_{\mathsf{LCH},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{ind}_{\mathsf{LCH},\mathcal{B}_2}(\lambda)) + n\cdot\epsilon + \frac{1}{|\mathcal{M}|} + \frac{Q_{sign}+Q_H}{2^{n\kappa}} + Q_{sign}n\cdot\gamma,$$

$$\mathsf{Adv}^{s\text{-}anony}_{\sharp\mathsf{AMS},\mathcal{A}}(\lambda) \leq \frac{Q_{sign}n}{2}\cdot\gamma$$

where Q_{sign} and Q_H are the numbers of signing queries (in the strong unforgeability experiment or the strong anonymity experiment) and hash queries, respectively.

6 Applications: Blockchain Governance and the Beyond

♯AMS can be massively applied in the scenario of blockchain and privacy-preserving, where authenticity and privacy are required simultaneously. The first and the most significant application of ♯AMS is blockchain governance, especially about ranking improvement proposals, which is one of the most uprising topics in the blockchain era. Our main goal is to implement a ranking – in other words, a voting – system on formal on-chain blockchain governance. (E-)voting systems need a signature scheme to prevent double-voting or any other possible exploitations related to confidentiality.

In this section, we give a generic treatment to achieve blockchain governance by developing a voting system via ♯AMS. Note that our applications are simplified to give a showcase of using our ♯AMS. Applying our scheme to more complex voting designs could be an interesting direction. However, our simple solutions have already achieved many attractive properties, e.g., lightweight and publicly verifiable – thus, there is no need for tallying authorities.

Note that in our schemes, anonymity does not hold for the moderator, meaning that the moderator (whether honest or malicious) knows the identities of all signers. However, unforgeability still holds even against a malicious moderator. Recall that in blockchain governance, a developer P who intends to propose an improvement proposal will start the ♯AMS protocol as the moderator. P's goal is to gather as many supporters as possible, and

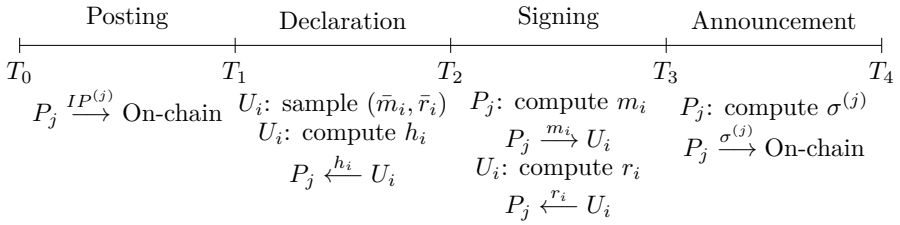


Figure 2 Timeline of V1. Here U_i votes to $IP^{(j)}$.

then generate a ♯AMS signature to demonstrate the credibility of the proposal and thereby win the RFP. A malicious moderator P can do nothing except reduce t – which trivially harms P's utility. For this reason, we assume the moderator is honest in the following analysis.

6.1 V1: Blockchain Governance via ♯AMS

As introduced in Section 3, the verification function of ♯AMS returns the number of the signers who participated in the signature generation, which can turn into a voting protocol. In our concrete scheme of ♯AMS, each participant generates their own keys and publishes them, which is a straightforward pre-processing step in voting via blockchain. The voting protocol enables multiple improvement proposals to compete for votes, and the one with the most votes is elected. Our on-chain governance voting protocol processes as follows:

1. A *voting session* implies the whole process of this protocol. For each RFP, the participants run a voting session to evaluate proposals. Each session has four time periods: *posting period*, *declaration period*, *signing period*, and *announcement period*. We denote $[n]$ as the index set of voters while U_i implies an i-th voter for $i \in [n]$. Similarly, for $p < n$, $[p]$ is the index set of developers who propose an improvement proposal while P_j stands for j-th developer and $IP^{(j)}$ stands for the proposal made by P_j. Note that $P_j = U_j$ for $j \in [p]$, which means P_j is also eligible to vote and will lead the quorum of $IP^{(j)}$. P_j serves a dual role as both the initiator of $IP^{(j)}$ and the representative of voters of that proposal, ensuring the concealment of their identities during signature generation.
2. In the posting period, for all $j \in [p]$, P_j submits $IP^{(j)}$. This can be done via the blockchain network using a smart contract.
3. In the declaration period, for each $i \in [n]$, U_i expresses their will to vote on $IP^{(j)}$ by generating a random string pair $(\bar{m}_i, \bar{r}_i)$ using their own randomness, which should be kept in secret, and sending $h_i = \mathsf{Hash}(hk_i, \bar{m}_i, \bar{r}_i)$ to P_j.
4. In the signing period, P_j sends m_i to U_i and U_i sends r_i to P_j, where m_i and r_i are defined in the description of ♯AMS in Section 5.
5. In the announcement period, P_j generates a signature $\sigma^{(j)}$ from (m_i, r_i) pairs for each voter and uploads it to the blockchain system. In the end, the most voted proposal is elected.

See Fig. 2 for the pictorial explanation of our voting system.

6.2 V2: Round Optimization

Recall that in the voting scheme above, after publishing the improvement proposal, each participant has to execute three rounds of interaction before uploading the ♯AMS signature on the blockchain.

- Round 1 (in the declaration period): P_j receives h_i as a voting-claim from its supporter U_i.
- Round 2 (in the signing period): P_j computes m_i according to the signing algorithm of ♯AMS and sends m_i to U_i.
- Round 3 (in the signing period): U_i returns r_i such that $\mathsf{Hash}(hk_i, m_i, r_i) = h_i$.

After Round 3, P_j can finish signing and upload the signature σ on the blockchain. Now we come up with the following question:

Can we reduce the round complexity? Namely, can we optimize the protocol such that the developer P_j can generate the ♯AMS signature immediately after receiving the voting claims from its supporters?

The natural idea is to let the supporter, say U_i, send its secret key (the trapdoor of chameleon hash schemes) directly to the developer P_j. With the knowledge of all secret keys, P_j can now generate a ♯AMS signature without interaction with its supporters. However, this will totally expose users' secret keys to the developers in a single vote event, which is not the case users want.

Our idea is to use ♯AMS in a one-time paradigm, see Fig. 3. That is, for each proposal $IP^{(j)}$ proposed by the developer P_j, the supporter U_i generates a new hash key and trapdoor pair $(hk_i^{(j)}, td_i^{(j)})$, and then sends the key pair to P_j. For a user U_k who does not want to support $IP^{(j)}$, it also generates a key pair $(hk_k^{(j)}, td_k^{(j)})$, but then sends only the hash key to P_j. This hash key is used for P_j to add U_k into the anonymous group to hide the identities of the real voters. Moreover, we have the following two modifications.

1. To make sure a hash key $hk_i^{(j)}$ is generated from voter U_i but not developer P_j (otherwise it can always make an n-valid ♯AMS signature), U_i will sign on $hk_i^{(j)}$ via a standard signature scheme to show its authority.
2. The message from U_i to P_j is encrypted using P_j's public key, so that no eavesdropper except P_j will know the corresponding trapdoor $td_i^{(j)}$. Meanwhile, the message, either $hk_i^{(j)}$ or $(hk_i^{(j)}, td_i^{(j)})$, is padded to the same length, preventing the side-leakage of anonymity.

Formally, the round-optimal protocol is described as follows. Let $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ be a (regular) signature scheme with unforgeability, and PKE be a public key encryption scheme with CPA security. At the beginning, we assume that every user U_i for $i \in [n]$ has its own key pairs $(\widetilde{vk}_i, \widetilde{sk}_i)$ of Sig and (pk_i, sk_i) of PKE.

1. In the posting period, developers $P_1, ..., P_p$ submit improvement proposals $IP^{(1)}, ..., IP^{(p)}$ to the blockchain network using a smart contract.
2. In the declaration & signing period, for all $i \in [n]$ and $j \in [p]$,
   a. U_i invokes $(hk_i^{(j)}, td_i^{(j)}) \leftarrow \mathsf{LCH.Gen}(1^\lambda)$ and $cert_i^{(j)} \leftarrow \mathsf{Sign}(\widetilde{sk}_i, hk_i^{(j)})$;
   b. If U_i wants to vote $IP^{(j)}$, U_i computes $ct_{i\to j} \leftarrow \mathsf{Enc}(pk_j, hk_i^{(j)} || cert_i^{(j)} || td_i^{(j)})$;
   c. If U_i does not want to vote $IP^{(j)}$, U_i computes $ct_{i\to j} \leftarrow \mathsf{Enc}(pk_j, hk_i^{(j)} || cert_i^{(j)} || \mathbf{0})$, where $\mathbf{0}$ is a zero-string of length $|td_i^{(j)}|$;
   d. U_i sends $ct_{i\to j}$ to P_j.
3. In the announcement period, for each $j \in [p]$, after decrypting $ct_{i\to j}$ for all $i \in [n]$, the developer P_j obtains a series of hash keys $\{hk_i^{(j)}\}_{i\in[n]}$ as well as their corresponding certificates $\{cert_i^{(j)}\}_{i\in[n]}$. Meanwhile, P_j also gets a group of trapdoors $\{td_i^{(j)}\}$ from users U_i who are willing to support $IP^{(j)}$. P_j can check the validity of certificates using long-term verification keys. If the verification fails with respect to user U_i, then P_j discards U_i from the anonymous group. Finally, P_j generates a ♯AMS signature $\sigma^{(j)}$ for its proposal $IP^{(j)}$ and uploads the ♯AMS signature, all hash keys, and certificates concerning the proposal $IP^{(j)}$ to the blockchain system.

Note that if a user U_i behaves in a Byzantine manner, for example, by failing to send $ct_{i\to j}$ to P_j, this misbehavior will be easily detected after the announcement period ends. In such cases, all developers can simply re-run the announcement period with the participant set updated to $[n] \setminus i$. This approach naturally generalizes to tolerate any number of faulty users, with the only cost being an additional announcement period for each round of fault recovery.

6.3 V3: Single-Vote Setting via the Conditioned Key Generation Paradigm

A voting system in the single-vote setting is desirable in many applications. For example, if two proposals conflict, an unruly user voting on both will undoubtedly disrupt the normal voting process and outcomes. However, in the protocols described above, users can vote on several proposals. More precisely, suppose there are two distinct proposals, $IP^{(j)}$ by P_j and $IP^{(k)}$ by P_k with $j \neq k \in [p]$. User U_i can generate two key pairs $(hk_i^{(j)}, td_i^{(j)})$ and $(hk_i^{(k)}, td_i^{(k)})$, then vote on both proposals by sending the corresponding key pairs to P_j and P_k, respectively. Due to the anonymity property of ♯AMS, the verifier cannot determine which users have voted multiple times.

To prevent double voting, we introduce a new paradigm, dubbed the *conditioned key generation paradigm*. Recall that to vote on a proposal $IP^{(j)}$ (i.e., to participate in the signing process for the proposal), user U_i must have the trapdoor $td_i^{(j)}$ corresponding to the hash key $hk_i^{(j)}$ specifically designed for $IP^{(j)}$. User U_i needs to generate p hash keys $(hk_i^{(1)}, ..., hk_i^{(p)})$ initially. Our idea is to enforce a restriction such that among all p hash keys, the user can know at most one trapdoor.

The conditioned key generation paradigm is applying t-out-of-n proof strategy during the subkey generation. Assume the hash key space $\mathcal{HK}$ is a field. Let $\mathbf{B} = (b_{i,j}) \in \mathcal{HK}^{(p-1)\times p}$ be a public matrix of full rank. Let $\mathbf{hk}_i^\top = (hk_i^{(1)}, ..., hk_i^{(p)})^\top$. Now we require

$$\mathbf{B} \cdot \mathbf{hk}_i = \hat{\mathbf{hk}}_i,$$

where $\hat{\mathbf{hk}}_i \in \mathcal{HK}^{p-1}$ is the output of some hash function $H(IP^{(j_1)}, ..., IP^{(j_p)}, i)$.

If the hash key generated via $\mathsf{Gen}(1^\lambda)$ is computationally indistinguishable from a uniform hash key in $\mathcal{HK}$, and $H(\cdot)$ is a random oracle, then to satisfy the aforementioned equation, each user has at most one trapdoor of the p hash keys. Single-voting property is achieved as a result. See Fig. 3 for the pictorial explanation of our round-optimal voting system in multiple and single-vote settings.

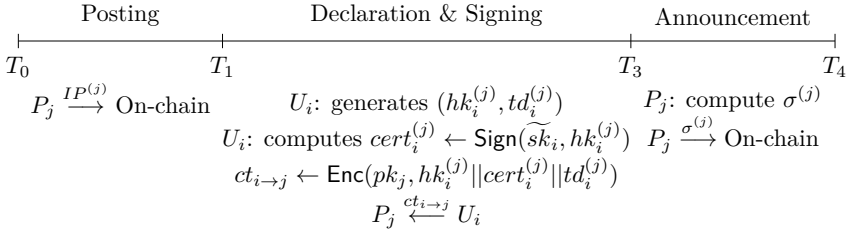


Figure 3 Timeline of V2 and V3. Here U_i votes to $IP^{(j)}$. In V2, U_i also needs to compute and send $ct_{i\to k} = \mathsf{Enc}(pk_k, hk_i^{(k)}||cert_i^{(k)}||td_i^{(k)})$ or $ct_{i\to k} = \mathsf{Enc}(pk_k, hk_i^{(k)}||cert_i^{(k)}||\mathbf{0})$ to P_k for all $k \in [p]$. In V3, there should be a unique $j \in [p]$ for each $i \in [n]$ such that U_i uniquely upvotes to $IP^{(j)}$. For all $k \in [p]$ such that $k \neq j$, U_i should generate $hk_i^{(k)}$ by following the rule described in the context.

References

1 Michel Abdalla, Pierre-Alain Fouque, Vadim Lyubashevsky, and Mehdi Tibouchi. Tightly-secure signatures from lossy identification schemes. In *EUROCRYPT 2012*, volume 7237, pages 572–590. Springer, 2012. doi:10.1007/978-3-642-29011-4_34.

2 Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In *CRYPTO 2015*, pages 763–780. Springer, 2015. doi:10.1007/978-3-662-48000-7_37.

3 Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *CSF 2008*, pages 195–209. IEEE Computer Society, 2008. doi:10.1109/CSF.2008.26.

4 Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In *CRYPTO 2020*, volume 12171, pages 562–592. Springer, 2020. doi:10.1007/978-3-030-56880-1_20.

5 Roman Beck, Christoph Müller-Bloch, and John Leslie King. Governance in the blockchain economy: A framework and research agenda. *J. Assoc. Inf. Syst.*, 19(10):1, 2018. URL: https://aisel.aisnet.org/jais/vol19/iss10/1.

6 Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In *EUROCRYPT 2009*, volume 5479, pages 1–35. Springer, 2009. doi:10.1007/978-3-642-01001-9_1.

7 Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS 2006*, pages 390–399. ACM, 2006. doi:10.1145/1180405.1180453.

8 Mihir Bellare and Todor Ristov. Hash functions from sigma protocols and improvements to VSH. In *ASIACRYPT 2008*, volume 5350, pages 125–142. Springer, 2008. doi:10.1007/978-3-540-89255-7_9.

9 Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC 1988*, pages 1–10. ACM, 1988. doi:10.1145/62212.62213.

10 Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *STOC 1994, Montréal, Québec, Canada*, pages 544–553. ACM, 1994. doi:10.1145/195058.195407.

11 Emmanuel Bresson, Jacques Stern, and Michael Szydlo. Threshold ring signatures and applications to ad-hoc groups. In *CRYPTO 2002*, volume 2442, pages 465–480. Springer, 2002. doi:10.1007/3-540-45708-9_30.

12 David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT 2010*, volume 6110, pages 523–552. Springer, 2010. doi:10.1007/978-3-642-13190-5_27.

13 Pierre-Louis Cayrel, Richard Lindner, Markus Rückert, and Rosemberg Silva. A lattice-based threshold ring signature scheme. In *LATINCRYPT 2010*, volume 6212, pages 255–272. Springer, 2010. doi:10.1007/978-3-642-14712-8_16.

14 Tony K. Chan, Karyin Fung, Joseph K. Liu, and Victor K. Wei. Blind spontaneous anonymous group signatures for ad hoc groups. In *ESAS 2004*, volume 3313, pages 82–94. Springer, 2004. doi:10.1007/978-3-540-30496-8_8.

15 David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC 1988*, pages 11–19. ACM, 1988. doi:10.1145/62212.62214.

16 David Chaum, Jan-Hendrik Evertse, and Jeroen van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *EUROCRYPT 1987*, volume 304, pages 127–141. Springer, 1987. doi:10.1007/3-540-39118-5_13.

17 Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. doi:10.1016/j.tcs.2019.02.001.

18 Wonseok Choi, Xiangyu Liu, Lirong Xia, and Vassilis Zikas. K-linkable ring signatures and applications in generalized voting. *IACR Cryptol. ePrint Arch.*, page 243, 2025. URL: https://eprint.iacr.org/2025/243.

19 Wonseok Choi, Xiangyu Liu, and Vassilis Zikas. Blockchain governance via sharp anonymous multisignatures. *Cryptology ePrint Archive*, 2023.

20 Sherman S. M. Chow, Lucas Chi Kwong Hui, and Siu-Ming Yiu. Identity based threshold ring signature. In *ICISC 2004*, volume 3506, pages 218–232. Springer, 2004. doi:10.1007/11496618_17.

21 Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO 1994*, volume 839, pages 174–187. Springer, 1994. doi:10.1007/3-540-48658-5_19.

22 Léonard Dallot and Damien Vergnaud. Provably secure code-based threshold ring signatures. In *Cryptography and Coding 2009*, volume 5921, pages 222–235. Springer, 2009. doi:10.1007/978-3-642-10868-6_13.

23 Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *CSFW-19 2006*, pages 28–42. IEEE Computer Society, 2006. doi:10.1109/CSFW.2006.8.

24 Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols: A taster. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, volume 6000, pages 289–309. Springer, 2010. doi:10.1007/978-3-642-12980-3_18.

25 Ali El Kaafarani, Shuichi Katsumata, and Federico Pintore. Lossy csi-fish: Efficient signature scheme with tight reduction to decisional CSIDH-512. In *PKC 2020*, volume 12111, pages 157–186. Springer, 2020. doi:10.1007/978-3-030-45388-6_6.

26 Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO 1986*, volume 263, pages 186–194. Springer, 1986. doi:10.1007/3-540-47721-7_12.

27 Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In *PKC 2007*, volume 4450, pages 181–200. Springer, 2007. doi:10.1007/978-3-540-71677-8_13.

28 Juan A. Garay, Aggelos Kiayias, Rafail M. Ostrovsky, Giorgos Panagiotakos, and Vassilis Zikas. Resource-restricted cryptography: Revisiting MPC bounds in the proof-of-work era. In *EUROCRYPT 2020*, volume 12106, pages 129–158. Springer, 2020. doi:10.1007/978-3-030-45724-2_5.

29 Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC 2008*, pages 197–206. ACM, 2008. doi:10.1145/1374376.1374407.

30 Hans Gersbach, Akaki Mamageishvili, and Manvir Schneider. Vote delegation and misbehavior. In *SAGT 2021*, volume 12885, page 411. Springer, 2021. URL: https://link.springer.com/content/pdf/bbm%3A978-3-030-85947-3%2F1.pdf.

31 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC 1987*, pages 218–229. ACM, 1987. doi:10.1145/28395.28420.

32 LM Goodman. Tezos – A self-amending crypto-ledger white paper. URL: https://www.tezos.com/static/papers/whitepaper.pdf, 4:1432–1465, 2014.

33 Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT 2008*, pages 415–432. Springer, 2008. doi:10.1007/978-3-540-78967-3_24.

34 Abida Haque and Alessandra Scafuro. Threshold ring signatures: New definitions and post-quantum security. In *PKC 2020*, volume 12111, pages 423–452. Springer, 2020. doi:10.1007/978-3-030-45388-6_15.

35 James Heather and Steve A. Schneider. A formal framework for modelling coercion resistance and receipt freeness. In *FM 2012*, volume 7436 of *Lecture Notes in Computer Science*, pages 217–231. Springer, 2012. doi:10.1007/978-3-642-32759-9_19.

36 Brett Hemenway, Benoît Libert, Rafail Ostrovsky, and Damien Vergnaud. Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. In *ASIACRYPT 2011*, volume 7073, pages 70–88. Springer, 2011. doi:10.1007/978-3-642-25385-0_4.

37 Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 539–556. Springer, 2000. doi:10.1007/3-540-45539-6_38.

38 Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In *CRYPTO 2014*, volume 8617, pages 369–386. Springer, 2014. doi:10.1007/978-3-662-44381-1_21.

39 Hugo L. Jonker and Erik P. de Vink. Formalising receipt-freeness. In *ISC 2006*, volume 4176 of *Lecture Notes in Computer Science*, pages 476–488. Springer, 2006. doi:10.1007/11836810_34.

40 Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, volume 6000, pages 37–63. Springer, 2010. doi:10.1007/978-3-642-12980-3_2.

41 Nida Khan, Tabrez Ahmad, Anass Patel, and Radu State. Blockchain governance: An overview and prediction of optimal strategies using nash equilibrium. *CoRR*, abs/2003.09241, 2020. arXiv:2003.09241.

42 Aggelos Kiayias and Philip Lazos. Sok: Blockchain governance. In *AFT 2022*, pages 61–73. ACM, 2022. doi:10.1145/3558535.3559794.

43 Aggelos Kiayias, Murat Osmanoglu, and Qiang Tang. Graded signatures. In *ISC 2015*, volume 9290, pages 61–80. Springer, 2015. doi:10.1007/978-3-319-23318-5_4.

44 Hugo Krawczyk and Tal Rabin. Chameleon hashing and signatures. *IACR Cryptol. ePrint Arch.*, page 10, 1998. URL: http://eprint.iacr.org/1998/010.

45 Ralf Küsters and Tomasz Truderung. An epistemic approach to coercion-resistance for electronic voting protocols. In *(SP 2009)*, pages 251–266. IEEE Computer Society, 2009. doi:10.1109/SP.2009.13.

46 Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, privacy, and coercion-resistance: New insights from a case study. In *SP 2011*, pages 538–553. IEEE Computer Society, 2011. doi:10.1109/SP.2011.21.

47 Ralf Küsters, Tomasz Truderung, and Andreas Vogt. A game-based definition of coercion resistance and its applications. *Journal of Computer Security*, 20(6):709–764, 2012. doi:10.3233/JCS-2012-0444.

48 Joseph K Liu, Man Ho Au, Willy Susilo, and Jianying Zhou. Linkable ring signature with unconditional anonymity. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):157–165, 2013. doi:10.1109/TKDE.2013.17.

49 Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. A separable threshold ring signature scheme. In *ICISC 2003*, volume 2971, pages 12–26. Springer, 2003. doi:10.1007/978-3-540-24691-6_2.

50 Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In *ACISP 2004*, volume 3108, pages 325–335. Springer, 2004. doi:10.1007/978-3-540-27800-9_28.

51 Jiazhuo Lyu, Zoe Lin Jiang, Xuan Wang, Zhenhao Nong, Man Ho Au, and Junbin Fang. A secure decentralized trustless e-voting system based on smart contract. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 570–577. IEEE, 2019. doi:10.1109/TRUSTCOM/BIGDATASE.2019.00082.

52 Markus Michels and Patrick Horster. Some remarks on a receipt-free and universally verifiable mix-type voting scheme. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 125–132. Springer, 1996. doi:10.1007/BFB0034841.

53 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, May 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

54 Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Security Protocols, 5th International Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 25–35. Springer, 1997. doi:10.1007/BFB0028157.

55 Jiaxin Pan and Benedikt Wagner. Lattice-based signatures with tight adaptive corruptions and more. In *PKC 2022*, volume 13178, pages 347–378. Springer, 2022. doi:10.1007/978-3-030-97131-1_12.

56 Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *STOC 2008*, pages 187–196. ACM, 2008. doi:10.1145/1374376.1374406.

57 Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC 1989*, pages 73–85. ACM, 1989. doi:10.1145/73007.73014.

58 Antonio Russo, Antonio Fernández Anta, María Isabel González Vasco, and Simon Pietro Romano. Chirotonia: A scalable and secure e-voting framework based on blockchains and linkable ring signatures. In *2021 IEEE International Conference on Blockchain, Blockchain 2021*, pages 417–424. IEEE, 2021. doi:10.1109/BLOCKCHAIN53845.2021.00065.

59 Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth. In *EUROCRYPT 1995*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer, 1995. doi:10.1007/3-540-49264-X_32.

60 Rowan van Pelt, Slinger Jansen, Djuri Baars, and Sietse Overbeek. Defining blockchain governance: A framework for analysis and comparison. *Inf. Syst. Manag.*, 38(1):21–41, 2021. doi:10.1080/10580530.2020.1720046.

61 Sarad Venugopalan and Ivan Homoliak. Always on voting: A framework for repetitive voting on the blockchain. *CoRR*, abs/2107.10571, 2021. arXiv:2107.10571.

62 Baodian Wei, Yusong Du, Huang Zhang, Fangguo Zhang, Haibo Tian, and Chong-zhi Gao. Identity based threshold ring signature from lattices. In *NSS 2014*, volume 8792, pages 233–245. Springer, 2014. doi:10.1007/978-3-319-11698-3_18.

63 Duncan S. Wong, Karyin Fung, Joseph K. Liu, and Victor K. Wei. On the rs-code construction of ring signature schemes and a threshold setting of RST. In *ICICS 2003*, volume 2836, pages 34–46. Springer, 2003. doi:10.1007/978-3-540-39927-8_4.

64 Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS 1982*, pages 160–164. IEEE Computer Society, 1982. doi:10.1109/SFCS.1982.38.

Modeling Loss-Versus-Rebalancing in Automated Market Makers via Continuous-Installment Options

Srisht Fateh Singh
University of Toronto, Canada

Reina Ke Xin Li
University of Toronto, Canada

Samuel Gaskin
Independent Researcher, Toronto, Canada

Yuntao Wu
University of Toronto, Canada

Jeffrey Klinck
University of Toronto, Canada

Panagiotis Michalopoulos
University of Toronto, Canada

Zissis Poulos
York University, Toronto, Canada

Andreas Veneris
University of Toronto, Canada

Abstract

This paper mathematically models a constant-function automated market maker (CFAMM) position as a portfolio of exotic options, known as perpetual American continuous-installment (CI) options. This model replicates an AMM position's delta at each point in time over an infinite time horizon, thus taking into account the perpetual nature and optionality to withdraw of liquidity provision. This framework yields two key theoretical results: (a) It proves that the AMM's adverse-selection cost, loss-versus-rebalancing (LVR), is analytically identical to the continuous funding fees (the time value decay or theta) earned by the at-the-money CI option embedded in the replicating portfolio. (b) A special case of this model derives an AMM liquidity position's delta profile and boundaries that suffer approximately constant LVR, up to a bounded residual error, over an arbitrarily long forward window. Finally, the paper describes how the constant volatility parameter required by the perpetual option can be calibrated from the term structure of implied volatilities and estimates the errors for both implied volatility calibration and LVR residual error. Thus, this work provides a practical framework enabling liquidity providers to choose an AMM liquidity profile and price boundaries for an arbitrarily long, forward-looking time window where they can expect an approximately constant, price-independent LVR. The results establish a rigorous option-theoretic interpretation of AMMs and their LVR, and provide actionable guidance for liquidity providers in estimating future adverse-selection costs and optimizing position parameters.

2012 ACM Subject Classification Applied computing → Economics

Keywords and phrases blockchain, decentralized finance, automated market makers, mathematical finance, perpetual options, continuous installments

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.6

Related Version *Full Version*: `https://arxiv.org/abs/2508.02971` [17]

Acknowledgements The authors would like to thank the reviewers for their helpful comments.

1 Introduction

The success of blockchains supporting smart contract such as Ethereum [4], Solana [18], etc., has led to the rise of *Decentralized Finance* (DeFi) which offers alternatives to traditional financial services by removing central trusted intermediaries and replacing them with public, verifiable, and immutable computer programs. One of the pivotal components of the DeFi infrastructure stack is automated market makers, or AMMs, allowing the exchange of one token for another at prices decided by an underlying algorithm. In recent years, AMMs have seen a rapid adoption reflected in financial metrics such as total value locked (above

© Srisht Fateh Singh, Reina Ke Xin Li, Samuel Gaskin, Yuntao Wu, Jeffrey Klinck, Panagiotis Michalopoulos, Zissis Poulos, and Andreas Veneris;
licensed under Creative Commons License CC-BY 4.0

$21B) and yearly transaction volumes (above $2T), as well as by their composability [8, 13]. Today, tens of thousands of tokens are listed and hundreds of applications are built on top of them [10].

The key participants in an AMM are traders and agents known as *liquidity providers*, or LPs. Traders exchange one token for another, where the token pair generally consists of a risky asset with volatile value, and a stable asset or numéraire.[1] On the other hand, LPs serve as counterparties to traders (sellers to buyers and buyers to sellers) by depositing both tokens upfront to the exchange. As a result of each trade, LPs receive the less favourable of the two tokens. To hedge against the adverse selection faced by AMMs, LPs can continuously rebalance an off-chain replicating portfolio by accumulating the risky token as its price rises and selling it when the price falls. However, this hedge is not perfect when the AMM is not the primary venue for price discovery because the pool's quoted price tends to lag behind the prevailing price on a primary venue such as a centralized exchange.

Under such a setting, even proactive LPs who rebalance in response to price changes are exposed to a systematic cost. Since AMM quotes lag those on primary markets, arbitrageurs can act faster than LPs and restore the pool price to the external market level. This generates a small but persistent transfer of value from LPs to arbitrageurs, and when aggregated over multiple price updates, this cost becomes significant. This cost is referred to as *loss-versus-rebalancing* (LVR) [16]. The rate at which LVR accumulates depends on the steepness of the AMM curve and the volatility of the underlying token; both amplify the arbitrage gap and thus accelerate LVR.

Despite such costs, LPs are incentivized to participate through the earning of trading fees that are proportional to the value of each trade and paid by the trader. Therefore, LPs considering whether to provide liquidity on an AMM pool must calculate their expected *payoff* by estimating and comparing their position's LVR with the anticipated trading fees in some predetermined forward time window. Recent work [16] have analyzed and quantified expressions for instantaneous LVR and retrospectively tested with historic market data. However, there is limited work that provides estimation methods for future LVR.

This work estimates the LVR for an LP that decides to provide liquidity for an arbitrarily long period and can exit at any point. It does so by mathematically modeling liquidity provision on a general class of AMMs, known as constant function AMMs (CFAMMs), as selling a continuum of perpetual American put options across continuous strikes. Perpetual American options are financial derivatives that give their holder the right to buy (known as a call) or sell (known as a put) an underlying asset at an agreed-upon price (strike) with no expiration. Unlike traditional vanilla options, this work uses exotic options in its model, known as continuous installment (CI) options, in which the holder must pay a stream of constant installment rate, referred to as *funding fees*, to keep their position alive. This funding fee is analogous to the time value decay of traditional fixed-term options. The paper uses perpetual American CI options because, unlike fixed-term vanilla options, the pricing function of these options does not change over time (assuming other market parameters are constant). Moreover, despite their exotic nature, CI options have been well-studied in the past, and this work builds on results from the existing literature [5]. This approach yields two key theoretical results.

[1] Although some token pairs consist of two stable assets, this work focuses primarily on pairs of one risky and one stable asset.

Funding Fees = LVR. In the limit where the installment rate tends to infinity (analogous to extremely short-dated fixed-term options), a CI put option has the classic hockey-stick payoff function: it pays the difference between the strike and the spot price when the underlying's spot price is below the strike price, and zero otherwise. Therefore, in this limit, a continuous distribution of CI puts *exists* that delta-replicates an arbitrary LP position's payoff *at each point in time*. Moreover, the installment rate earned on this distribution of puts (which, at a given point in time, is earned by the option whose strike equals the spot price at that time) reproduces the expression for the instantaneous LVR. Therefore, theoretically, an LP can hold such an options portfolio to stay delta-neutral – the cost due to time value decay of holding this portfolio, *i.e.*, funding fees, is the LVR.

Since the above model is theoretical, as a continuum of options cannot be reproduced in practice, the paper subsequently quantifies the approximation error when a discrete portfolio of options with discrete strike prices is used to replicate the payoff of a liquidity position.

Constant Future LVR. The second result analyzes the converse scenario where a liquidity position's payoff replicates the valuation of a single perpetual American CI option. This produces a unique liquidity profile with *almost constant* instantaneous LVR rate over a forward time window. Moreover, this rate is approximately equal to the funding fee of the replicated CI option. As a result, this yields guidance to LPs, under the model assumptions, on:

- Choosing the price boundaries and shape of liquidity provision that incurs *predictable, flat, price-path independent* future LVR.
- Estimating the forward adverse-selection cost for a planned holding period.
- Understanding the relationship between the optimal holding period and the width of the liquidity position.
- Selecting an appropriate pool based on its expected future trading fee income.

The paper is organised as follows. Section 2 explains notations and the necessary background, Section 3 discusses prior literature and related works, Section 4 motivates the option-based interpretation, Section 5 provides the options decomposition, Section 6 proves the funding-fee–LVR identities, Section 7 measures the approximation error on delta when the continuous strip is replaced by finitely many strikes, and Section 8 presents the volatility calibration and design rules for LPs. Finally, Section 9 concludes with directions for future research.

2 Background

In this section, we provide the necessary notation, terminology and background concepts used in the remainder of the paper.

2.1 Notation

We consider two tokens: *token* 0 representing a risky asset (*e.g.* BTC, ETH) and *token* 1 representing a stable/safe asset (*e.g.* USDC). Let S_t denote the spot price of *token* 0 in units of *token* 1 at time t that follows a geometric Brownian motion (GBM) on a filtered probability space $\left(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\geq 0}, \mathbb{Q}\right)$ satisfying the standard assumptions for GBMs (where $\mathbb{Q}$ is a risk-neutral probability measure), so that

$$\frac{\mathrm{d}S_t}{S_t} = r\ \mathrm{d}t + \sigma\ \mathrm{d}B_t^{\mathbb{Q}}, \tag{1}$$

with constant annual risk-free rate r and volatility $\sigma > 0$. $\{B_t^{\mathbb{Q}}\}_{t\geq 0}$ is a Wiener process. As usual, we assume that AMMs constitute secondary markets and the price S_t is governed by primary markets such as centralized exchanges. In the subsequent sections, we omit the subscript t and use S and S_t interchangeably for convenience.

2.2 Constant-Function Automated Market Makers (CFAMMs)

A *constant-function automated market maker* (CFAMM) maintains token reserves (x, y), deposited by LPs, such that each trade transforms the reserves to (x', y') and the reserves before and after the trade satisfy an invariant function $F(x, y) = F(x', y') = k$. In a *constant-product* AMM (CPAMM), such as Uniswap v2 [1], the invariant takes the form of $F(x, y) = \sqrt{xy}$. As a result, the marginal exchange price, assuming no arbitrage, takes the form: $S = -\frac{\partial y}{\partial x} = \frac{y}{x}$. This is shown in Figure 1a. Therefore, the expression for token reserves are $x = \frac{k}{\sqrt{S}}$, and $y = k\sqrt{S}$.

A CPAMM with concentrated liquidity (as in Uniswap v3 [2]) uses an invariant parameter k within a price band $[a, b]$. The token reserves of LPs in this band consist of only *token 0* when the price $S \geq b$, only *token 1* when $S \leq a$, and for prices $S \in (a, b)$, the reserves are:

$$x = k\frac{\sqrt{b} - \sqrt{S}}{\sqrt{Sb}}, \qquad y = k\left(\sqrt{S} - \sqrt{a}\right). \tag{2}$$

Therefore, the reserve value (denominated in token 1) is

$$V(S) = k\left(\sqrt{S} - \sqrt{a}\right) + kS\,\frac{\sqrt{b} - \sqrt{S}}{\sqrt{Sb}}, \qquad S \in [a, b]. \tag{3}$$

The liquidity position's sensitivity to price, known as it's *delta*, is denoted by $X(S)$, and the sensitivity of delta to price, known as *gamma*, is denoted by $\Gamma(S)$. In practice, delta is a measure of exposure to small changes in the price of the risky asset, whereas gamma is a measure of exposure to large movements of the risky asset's price. These are given by the first and second derivatives of the value function $V(S)$ with respect to the underlying's price, respectively, and are expressed as follows:

$$X(S) := V'(S), \qquad \Gamma(S) := V''(S) = X'(S) \leq 0 \quad (S \in (a, b)). \tag{4}$$

Figure 1b illustrates the behavior of delta and gamma across the liquidity band. As shown, the magnitudes of both delta and gamma decrease monotonically with price, as higher prices correspond to a greater allocation to the numeraire asset.

2.2.1 Loss-Versus-Rebalancing (LVR)

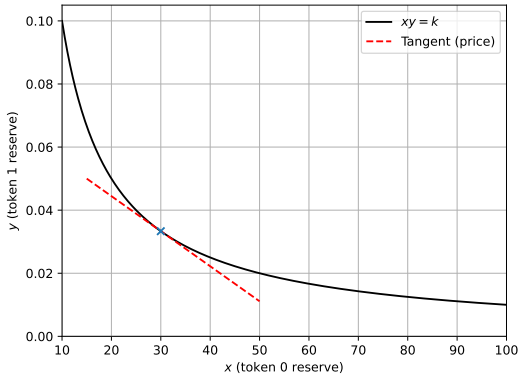
A continuously rebalanced, self-financing delta-hedge portfolio that holds $X(S_t)$ units of token 0 has value W_t with $\mathrm{d}W_t = X(S_t)\,\mathrm{d}S_t$. The difference

$$\mathrm{LVR}_t := V(S_t) - W_t \tag{5}$$

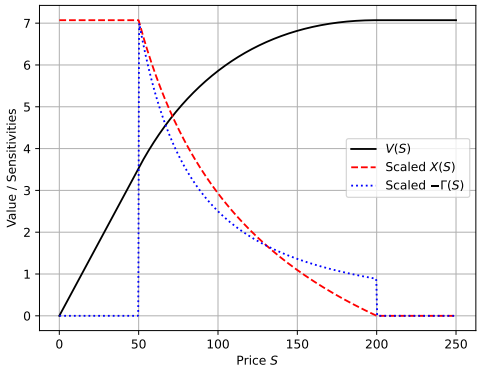
quantifies the AMM's adverse-selection loss relative to a hedged trader and is referred to as loss-versus-rebalancing or LVR. The instantaneous LVR, dLVR_t, grows quadratically with the spot price and volatility, and linearly with the gamma of the liquidity position [16].

$$\mathrm{dLVR}_t = \tfrac{1}{2}\,\sigma^2 S_t^2 \Gamma(S_t)\,\mathrm{d}t = \tfrac{1}{2}\,\sigma^2 S_t^2 \big[X'(S_t)\big]\,\mathrm{d}t. \tag{6}$$

We will later demonstrate the equivalence between the right-hand side of Eq. (6) and the funding fee of a perpetual American CI option.



(a) CPAMM Invariant with $k = 1$ and Marginal Price (Tangent at $x = 30$).



(b) $V(S)$, $100X(S)$, and $-5000\Gamma(S)$ for concentrated liquidity position in price band $[50, 200]$, and $k = 1$.

Figure 1 Profiling CPAMM invariant and concentrated liquidity position, its delta and gamma.

2.3 Perpetual American Continuous-Installment Options

A *perpetual American continuous-installment put option* has no expiration date and requires the holder to pay a continuous stream of *constant funding fee* $q > rK$ per year to keep the contract alive. At any point, the holder may choose to stop paying the fee, at which point they can either exercise the option or drop the position. The option can be exercised at any time for a payoff of $\max(K - S, 0)$ [5]. In the analysis below, we assume the underlying asset (token 0) pays zero dividends.

2.3.1 Notation and Ordinary Differential Equation Formulation

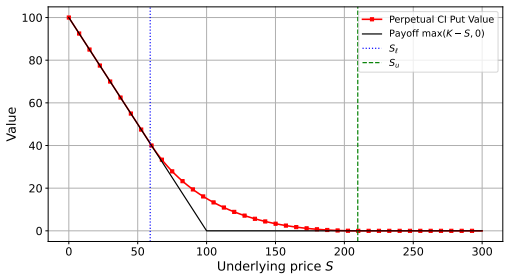
Let $P_q(S; K)$ denote the discounted put option value at spot price S, strike K, and funding fee q. Let S_ℓ denote the lower boundary, below which the option value equals its payoff, and let S_u denote the upper boundary, above which the option value is zero, as illustrated in Figure 2a. Under the risk–neutral dynamics, $P_q(S; K)$ satisfies the inhomogeneous Black-Scholes ordinary differential equation in the *continuation region* $S_\ell < S < S_u$:

$$\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 P_q}{\partial S^2} + rS\frac{\partial P_q}{\partial S} - r\,P_q \;=\; q, \qquad S \in (S_\ell, S_u). \tag{7}$$

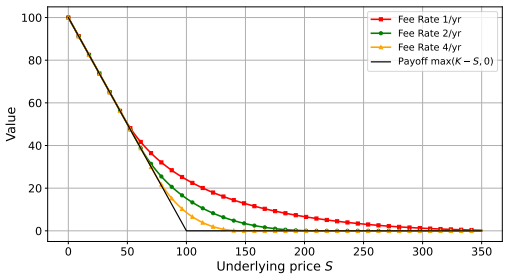
The left and right boundaries are determined endogenously from the value–matching and delta-matching conditions

$$\begin{aligned} P_q(S_\ell; K) &= K - S_\ell, & \frac{\partial P_q}{\partial S}(S_\ell; K) &= -1, \\ P_q(S_u; K) &= 0, & \frac{\partial P_q}{\partial S}(S_u; K) &= 0. \end{aligned} \tag{8}$$

Solving (7) with the four boundary conditions in (8) yields the closed-form expressions below.



(a) Discounted value of a perpetual CI put option with lower and upper boundaries.



(b) Discounted value of a perpetual CI put option with fee rates 1, 2, 4 per annum.

Figure 2 Profiling of perpetual CI put option with $K = 100.0$, $\sigma = 0.25$, $q = 2$ (in the first figure) and $r = 0.01$ (all parameters are annualized).

2.3.2 Closed-form solution

The expression for the option price $P_q(S; K)$ takes the following closed form as derived in [5].

$$P_q(S; K) = \alpha_p S + \beta_p S^{\gamma_p} + \frac{q}{r} \tag{9}$$

The delta of the put option, $X_q(S; K) = \frac{\partial}{\partial S} P_q(S; K)$, thus takes the form:

$$X_q(S; K) = \alpha_p + \beta_p \gamma_p S^{\gamma_p - 1}. \tag{10}$$

Moreover, the upper and lower boundaries, S_u and S_ℓ respectively, have the following closed form:

$$S_\ell = \frac{q}{r + \sigma^2/2} \left[g - g^{1/\gamma_p} \right], \tag{11}$$

$$S_u = \frac{q}{r + \sigma^2/2} \left[g^{1-1/\gamma_p} - 1 \right]. \tag{12}$$

Here, α_p, β_p, γ_p, and g are expressions that depend on parameters r, σ, K, and q and their expressions are provided in Appendix (A).

The Black–Scholes partial differential equation in (7) contains no derivative with respect to time. Consequently, the value of a perpetual CI option is *time-invariant*. This makes it unique from vanilla finite expiry American or European options, whose pricing has a time-varying component. The analog of expiration in CI options is the funding rate, where high funding rates make it "behave similar" to short expiration option and vice versa for small fee rates. This is depicted in Figure 2b, where increasing the fee rate reduces the price of the CI option closer to its payoff. The lower and upper boundaries, S_ℓ and S_u, characterize the holder's optimal policy. When the spot price first falls below S_ℓ, it is optimal to *exercise* the option; when it first exceeds S_u, it is optimal to *drop* the option – *i.e.*, to exit the position with zero payoff. This is because, in both cases, the expected benefit of continued funding is outweighed by its cost. In either scenario, the holder stops paying the funding fee immediately upon exit. For this reason, S_ℓ and S_u are also called *optimal exercise* and *dropping boundaries*, respectively. Conversely, the option seller receives the continuous funding fee only while the spot price remains in the continuation region $S_\ell < S < S_u$. We will exploit this fact in the sections that follow.

3 Related Work

Early work on studying LP positions in CFAMMs focus on mitigating risks associated with impermanent loss – the loss experienced by a liquidity provider compared to simply holding the asset, so that an LP can earn trading fees without exposure to this loss. Deng et al. [9] and Fukasawa et al. [12] study static replication strategies for impermanent loss in finite-time CFAMM positions using European options and variance swaps, respectively. Lipton et al. [14] further studies model-based dynamic replication. However, it can be argued that a comparison against a buy-and-hold strategy is insufficient as it does not account for adverse selection, where informed traders extract value from passive LPs over time.

Another approach to quantifying LP loss is the concept of LVR, which captures the loss incurred by a liquidity provider compared to continuously rebalancing their portfolio at market prices, due to adverse selection. LVR was formalized by Millionis et al. [16]. Their work offers closed-form expressions for instantaneous LVR in a CFAMM and provides empirical validation using historical market data. However, they focus on instantaneous and historical LVR and do not address forward-looking or long-term LVR estimation. Meanwhile, our approach provides a framework that captures forward-looking LVR by delta-replication with options portfolios.

Maire and Wunsch [15] make a case for the hedging of the LP position value instead of the impermanent loss. They study the problem of market-neutral liquidity provision by constructing a static replication portfolio that matches the AMM position's dollar value over time, effectively achieving a constant value position for the finite lifetime of the position. The replicating portfolio's margin requirement is then itself hedged by shorting a perpetual or dated futures contract to offset changes in the margin value, enabling a market-neutral LP strategy that generates interest from LP trading fees and futures funding fees. Meanwhile, Clark [6, 7] investigates the replicating portfolio of the payoff of a constant product AMM position and shows how an LP position's terminal value can be fully statically replicated using a portfolio of European options. The approach focuses on fixed, finite time horizons and seeks to hedge only the terminal value of the liquidity position. On the other hand, our approach neither relies on dynamic hedging, nor assumes finite time horizons. Instead, we model the LP's position over an indefinite time horizon using a portfolio of perpetual American CI put options, providing a theoretical framework that statically captures the path-dependency of forward-looking LVR.

4 Work Motivation

The value profile of a concentrated liquidity position in a finite price band, as shown in Figure 1b, closely resembles that of a portfolio consisting of cash (a constant payoff) and short put options (the negative of the payoff shown in Figure 2a): flat on one wing, linear on the other, and smoothly curved in between. Figure 3a makes this visual similarity precise by comparing the *value* of a CPAMM position with $k = 1$ and band $(80, 125)$ to a portfolio comprising cash and a one-month European put, with strike at the geometric mean of the price boundaries. The put is valued using the standard Black–Scholes model [3].

Despite the superficial similarity, key differences emerge. As shown in Figure 3b, the two profiles diverge meaningfully. More critically, the European option's value is inherently time-variant: even a single day's passage erodes its time value (*theta*), while the CPAMM's value remains time-stationary. This contrast is illustrated in Figure 3c, where the solid line represents the option value as a function of the time-to-maturity. In addition, the CPAMM offers a flexible, perpetual holding period, whereas fixed-term options require periodic rolling – selling expiring contracts and buying new ones – to replicate a liquidity position.

These challenges raise a natural question: *Can one construct a static portfolio that tracks a perpetual AMM band* without *daily rebalancing?* The answer is affirmative, provided we replace European options with a class of *American continuous-installment options*. This is because the ongoing constant funding fee, $q\,\mathrm{d}t$, charged by an active CI option offsets the time decay found in European options. In the limit of infinite maturity (the perpetual CI variant), the mark-to-market value of a CI put or call becomes *time-invariant*, producing a flat line as shown by the dashed line in Figure 3c.

This observation enables a decomposition of a CFAMM band into a *perpetual strip* of CI puts that both matches the pool's delta and offers stationarity. Beyond its conceptual appeal, this decomposition yields two practical insights:

1. The instantaneous *funding fee* of the active CI option equals the LP's *loss-versus-rebalancing* cost. Since a CI option is perfectly hedgeable with a Black-Scholes type rebalancing porfolio [5] and both the CPAMM and the CI option strip have the same delta, they share the same rebalancing portfolio, and hence the LVR is precisely the difference between the two instruments, *i.e.* instantaneous funding fee.
2. If the CFAMM's delta and price boundaries are calibrated to match those of a single CI put, then the LVR over a future time window becomes nearly constant and equal to the funding fee q. Notably, this construction relies on implied volatility, rather than instantaneous volatility, meaning it can be formulated using observed market data.

The following section formalizes the CI decomposition, proving the equivalence between funding fees and LVR.

5 Modeling a CFAMM Position with CI Options

5.1 Overview

In this section, we construct a portfolio whose delta (change in option price w.r.t. change in spot price) is the same as the delta of a CFAMM $X(S)$. This portfolio consists of a distribution of perpetual American CI put options in the limit $q \to \infty$ across a continuum of strike prices. As the funding rate tends to infinity, the exercise and dropping boundaries of each option collapse to the strike price, and the option's valuation converges to $\max(K-S, 0)$. Consequently, the option's delta converges to a step function. This property enables the construction of a portfolio that replicates the delta of an arbitrary (but smooth) CFAMM payoff function.

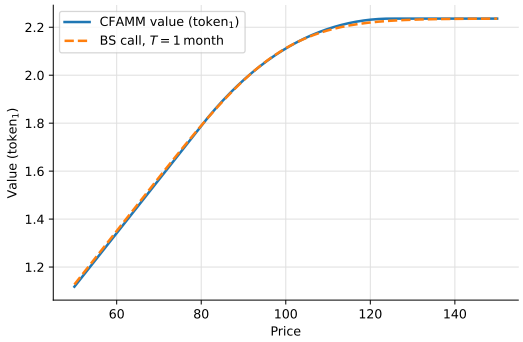
5.2 Portfolio Construction

We begin with the following lemma:

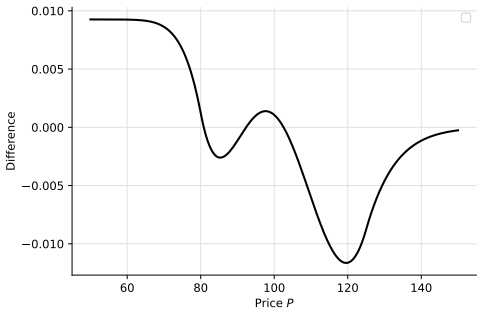
▶ **Lemma 1.** *In the limit $q \to \infty$, both the lower and upper boundary of a CI put converge to the strike K, and option's delta $X_q(S;K)$ becomes a step function $-\mathbf{1}_{\{S<K\}}$.*

This lemma encapsulates the relationship between the funding rate and the option's "effective" time to expiration. As the funding rate increases, the option behaves increasingly like a zero-time-to-expiry option, with its delta approaching a discontinuous step as $q \to \infty$.

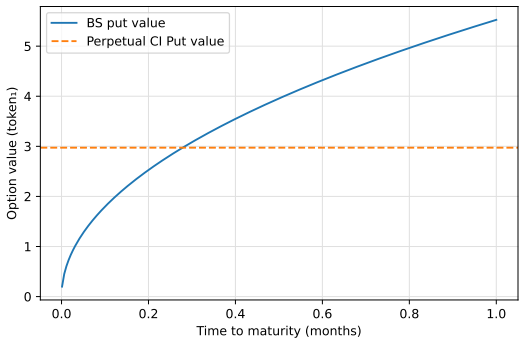
Next, we specify the weight distribution of the options portfolio that delta-replicates a CFAMM value function $V(S)$.



(a) Value of a CPAMM band $[80, 125]$ versus a short one-month Black–Scholes put plus cash.



(b) Difference between CPAMM value and short put plus cash.



(c) Time value decay of a European put versus the constant value of a perpetual CI call, whose funding fee offsets theta exactly.

Figure 3 Comparison between CPAMM liquidity value and a portfolio of short put (K=100, r=5%, σ=50%) plus cash.

▶ **Theorem 2.** *Let $V : \mathbb{R}_{>0} \to \mathbb{R}$ be a twice continuously differentiable function. Assume that $V' \in L^1(\mathbb{R}_{>0})$, V' has bounded variation on $\mathbb{R}_{>0}$ and that $\lim_{S\to\infty} V'(S) = 0$. Define the weight $w(K) := V''(K)$ and, for each $q < \infty$,*

$$\Pi_q(S) := \int_0^\infty w(K)\, P_q(S; K)\, dK,$$

and

$$\Pi(S) := \lim_{q\to\infty} \Pi_q(S).$$

Then the portfolio $\Pi(S) - V(S)$ is delta-neutral.

Thus, a perpetual CFAMM position that can be closed by its owner at any time can be perfectly modeled using a distribution of CI puts with very large funding rates[2]. In practice, a continuous distribution is infeasible, and funding rates are finite. Therefore, one may replicate the delta profile using a discrete set of puts with different strikes. However, discretization and

[2] $\Pi_q(S_t)$ has the same payoff as described above at all times. Therefore, the holder must continuously reissue options that are exercised or dropped.

finite funding rates introduce non-negligible delta-replication error, dependent on inter-strike spacing and funding rate. We quantify this approximation error numerically for a constant product AMM position in Section 7.

An additional corollary of the above result is the relationship between the instantaneous LVR of a liquidity position and the funding fees of options with strikes around the spot price (also referred to as activated strikes) in the replicating portfolio. This relationship is analyzed in the following section.

6 Establishing LVR as Funding Fees

The portfolio Π delta-replicates a given CFAMM liquidity position. However, unlike the AMM position, the short options portfolio pays a continuous funding fee to the seller – arising from the active CI puts with strike around the spot price. We show that these funding fees, absent in the AMM, are precisely equal to the LP's LVR.

To compute this running funding fee, we first establish the following lemma:

▶ **Lemma 3.** *As $q \to \infty$, the product $q\,(S_u(q;K) - S_\ell(q;K))$ converges to a finite limit:*

$$\lim_{q\to\infty} q \cdot \big(S_u(q;K) - S_\ell(q;K)\big) = \frac{\sigma^2 K^2}{2}.$$

Next, we express Π as the limit of a discrete sum of options. In the following lemma, we construct such a discretization and show that, as $q \to \infty$, this portfolio converges pointwise to that of Π. We also compute the limiting funding fee contribution from the option whose holding region contains the spot price.

▶ **Lemma 4.** *Fix $q > 0$ and an interval $[a, b] \subset \mathbb{R}_{>0}$. Define a sequence of strikes $K_1 < K_2 < \cdots < K_{N(q)}$ recursively by:*

$$\begin{cases} S_\ell(q;K_1) = a, \\ S_\ell(q;K_{i+1}) = S_u(q;K_i), \quad i = 1,\dots,N(q)-1, \\ S_u(q;K_{N(q)}) = b. \end{cases}$$

For each i, define the weight

$$w_i := \int_{S_\ell(q;K_i)}^{S_u(q;K_i)} w(K)\,\mathrm{d}K = X\big(S_u(q;K_i)\big) - X\big(S_\ell(q;K_i)\big),$$

so that

$$\sum_{i=1}^{N(q)} w_i = \int_a^b w(K)\,\mathrm{d}K = -1.$$

Let K_j be the activated strike such that $S_\ell(q;K_j) \le S_t \le S_u(q;K_j)$. Then, as $q \to \infty$, the discrete portfolio

$$\widetilde{\Pi}_q := \sum_{i=1}^{N(q)} w_i\, P_q(\cdot;K_i)$$

converges pointwise to the continuous payoff $\Pi(S)$. Moreover, the weighted funding fees at strike K_j satisfies

$$\lim_{q\to\infty} w_j q = \frac{\sigma^2 S^2}{2} X'(S).$$

Lemma 4 constructs a portfolio of CI puts with discrete strikes and finite funding rates, with weights chosen such that, in the limit $q \to \infty$, the portfolio converges to the CFAMM-replicating portfolio defined in Theorem (2). The index j denotes the unique option that remains active (*i.e.*, in the holding region), while all other options are either exercised or dropped. Therefore, the funding fee received by the portfolio owner is $w_j q$.

As q becomes large, Lemmas 3 and 4 together imply that this funding fee converges to the instantaneous LVR of the corresponding CFAMM position. This leads to the following result:

▶ **Theorem 5** (Funding fee = LVR)**.** *Let* dLVR_t *denote the instantaneous change in the LVR of a CFAMM position, and let* dFee_t *denote the instantaneous funding income of its delta-replicating CI option portfolio* Π*. Then,*

$$\mathrm{dFee}_t = \mathrm{dLVR}_t \quad \textit{and} \quad \mathrm{Fee}|_0^T = \mathrm{LVR}|_0^T \qquad (\forall\, T > 0).$$

Therefore, the LVR of a liquidity position is precisely the CI funding premium of its delta-replicating options portfolio. Another way to look at this is that a CI option can be perfectly delta-hedged using a continuously rebalanced Black–Scholes-type portfolio of risky and stable assets [5]. As the funding rate q increases, the delta of this rebalancing portfolio converges to that of a CFAMM position. However, unlike the CI option, a CFAMM position does not compensate the liquidity provider via a funding stream. The discrepancy between the CFAMM position and its hedge is therefore exactly the foregone CI funding. This equality relies solely on closed-form expressions and the principle of self-financing, without invoking any additional modeling assumptions.

One advantage of this option-theoretic interpretation is that CI fee rates, as implied by the options market, provide both real-time and forward-looking estimates of LVR, enabling informed range management. Moreover, this framework permits the construction of liquidity profiles with nearly price-path-independent LVR. These features are demonstrated in the sections that follow. A further implication is that a market for CI options allows for static-weight delta-hedging of CFAMM positions – under the assumption of constant implied volatility – eliminating the need for frequent rebalancing, unlike with conventional American or European options.

6.1 CFAMM Position Replicating a Unit CI Option

Consider a concentrated–liquidity CFAMM band whose delta, at every price level, matches the delta of a *single* perpetual American CI *put* option. Specifically, choose the liquidity bounds $a < b$ such that

$$X(S) \;=\; V'(S) \;\equiv\; X_q\big(S; K_*\big) \qquad (S \in [a, b]), \tag{13}$$

$$a = S_l(q, K_*), \tag{14}$$

$$b = S_u(q, K_*). \tag{15}$$

for some strike K_* and finite fee rate q.

▶ **Theorem 6.** *The instantaneous rate of change of the LVR of the above CFAMM position is approximately equal to the funding fees of the unit put option, up to a bounded approximation error. That is, there exists a residual function* $\epsilon(t)$ *with* $|\epsilon(t)| \le rK_*$*, such that*

$$d\mathrm{LVR}_t = q\,\mathrm{d}t + \epsilon(t)dt,$$

Thus, the AMM liquidity position with the above delta profile suffers an *almost flat, price-path-independent, volatility-independent* LVR. Note that because $X_q(S; K_*)$ depends on the volatility parameter σ, as shown in Eq (10), the calibrated boundaries a, b – and thus the entire delta curve $X(S)$ – remain implicitly volatility-dependent. The residual error term, $\epsilon(\cdot)$, is bounded in magnitude by a constant and its relative magnitude is reported and discussed in Section 8. The above liquidity profile is useful for LPs who want to estimate forward LVR and compare it with the *expected future trading fees*. Lastly, constructing such an AMM profile requires estimates of future volatility. This can be approximated using a term structure of implied volatility gathered from the fixed-term options market. Section 8 discusses this in detail and estimates the approximation error arising from the calibration between fixed-term and perpetual options' volatilities.

7 Error Analysis of Discrete CI-Option Replication

In this section we quantify the approximation error that arises when the continuous–strike decomposition of a concentrated CFAMM is replaced by a *discrete* strip of perpetual American CI put options with *finite* q.

7.1 Sources of error

We isolate two drivers of error: *(i)* The installment rate q (large but finite), and *(ii)* the inter-strike spacing ΔK of the discrete strip. For a given pair $(q, \Delta K)$, we construct a strip of discrete options and measure the absolute difference between the target delta (of the CFAMM) and the strip's delta. This is done over the active price band $S \in [a, b]$ and its maximum and root-mean-square values are plotted.

7.2 Experimental methodology

We chose a concentrated liquidity AMM as our target CFAMM. Thus, the analytical delta $X(S) = L(1/\sqrt{S} - 1/\sqrt{b})$ for uniform liquidity on a price band $[a, b]$. We choose $a = 80$, and $b = 125$, and L chosen so that $X(a) = 1$, $X(b) = 0$. We discretize the replication weight such that on each strike interval $[K_i, K_{i+1}]$ we set

$$w_i = \int_{K_i}^{K_{i+1}} X'(K)\,\mathrm{d}K \;=\; X(K_{i+1}) - X(K_i), \tag{16}$$

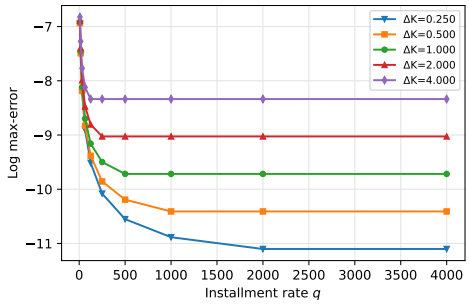
ensuring that the discrete weights sum to the continuous integral. For each strike, we compute the short CI put delta, clipped to $\{-1, 0\}$ outside its continuation band. Let $X_{\text{strip}}(S; q, \Delta K) = \sum_i w_i\, X_q(S; K_i)$ be the delta of the strip at S. Define the error

$$\varepsilon(S_j; q, \Delta K) = \left| X(S_j) - X_{\text{strip}}(S_j; q, \Delta K) \right|$$

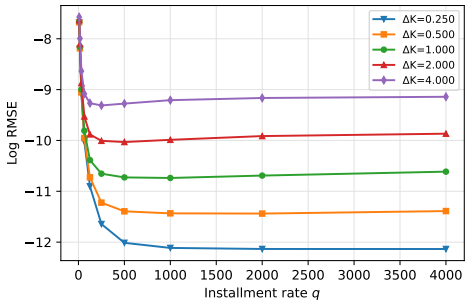
evaluated on grid $\{S_j\}_{j=1}^{N}$ where $N = 2000$. We consider two error metrics:

- *Maximum absolute error:* $\max_j \varepsilon(S_j; q, \Delta K)$.
- *Root-mean-square error (RMSE):* $\sqrt{N^{-1} \sum_j \varepsilon(S_j; q, \Delta K)^2}$.

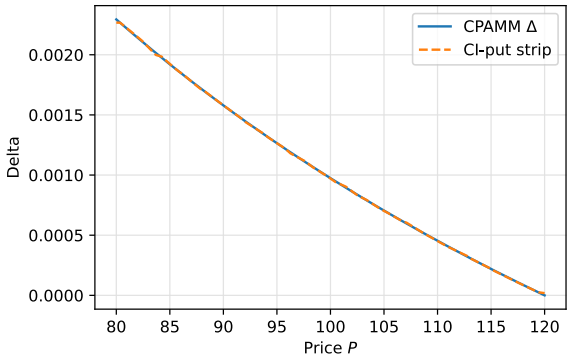
We evaluate a parameter sweep $(q, \Delta K) \in \{8, 16, 32, 64, 125, 250, 500, 1000, 2000, 4000\} \times \{0.25, 0.5, 1.0, 2.0, 4.0\}$.



(a) Log max error.



(b) Log RMSE error.



(c) CFAMM target delta (solid) versus discrete CI-put strip approximation (dashed) for $q = 250$, $\Delta K = 2$.

Figure 4 Log-Max absolute and RMSE delta-replication error versus installment rate q under different parameter settings.

7.3 Results

Figure 4a shows the logarithm of maximum absolute error versus q for five strike spacings. Similarly, Figure 4b displays the RMSE versus q for the same strike spacings. Both errors increase strictly with strike spacing for a given q across all installment rates. On the other hand, for a given strike spacing, both errors generally decrease with q with some exceptions. For large strike spacing, $\Delta K = \{1, 2, 4\}$, the RMSE error increases with q for large values, $q \geq 128$. Lastly, Figure 4c plots a representative delta curve (q=250, ΔK=2) against the CPAMM target delta.

7.4 Discussion

Both errors are below 10^{-3} and shrink as expected: a larger q steepens each put's delta, while a finer ΔK better resolves the continuous weight. The trade-off between the capital cost of a large q and the operational cost of a finer strike mesh can be balanced according to the LP's precision requirements.

8 Volatility Calibration for Perpetual CI Put Options

Perpetual CI *put* pricing requires a *constant* volatility parameter σ. Market quotes instead supply a term structure $\widehat{\sigma}(\tau)$ of implied volatility with time to expiration τ. In the following, we derive the effective implied volatility $\sigma_{\text{eff}}(q)$ for a perpetual American CI option with funding fee q using the market-implied term structure of at-the-money options.

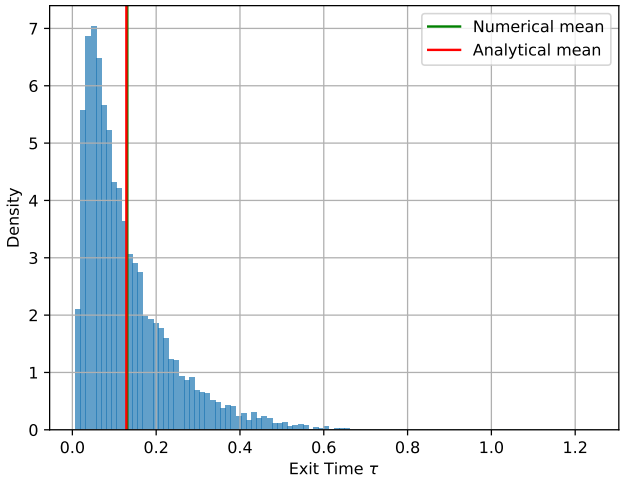


Figure 5 Distribution of first-exit time τ for $r = 2\%$, $\sigma = 67\%$, $K = 100$, $q = 5$.

8.1 Effective time horizon

For a CI put with rate q, the continuation band is $(S_\ell(q), S_u(q))$. It stays alive as long as the underlying spot price S_t remains within the continuation band. Otherwise, when $S_t = S_l$, the option is dropped, or when $S_t = S_u$, it is exercised by the holder.

Define the *first-exit time*

$$\tau(q) \;:=\; \inf\{t > 0 : S_t \notin (S_\ell(q), S_u(q))\},$$

i.e., the random horizon at which the CI position terminates. In probabilistic terms, $E[\tau(q)] = \bar{\tau}(q)$ is the *mean first-exit time* of a GBM between two absorbing boundaries.

▶ **Theorem 7.** *The closed-form solution of* $\bar{\tau}(q)$ *is*

$$\bar{\tau}(q) = \begin{cases} \frac{1}{\sigma^2} \ln\left(\frac{S_0}{S_l(q)}\right) \ln\left(\frac{S_u(q)}{S_0}\right) & \text{if } a = 0 \\ \frac{1}{a}\left[\ln\left(\frac{S_l(q)}{S_0}\right) + \ln\left(\frac{S_l(q)}{S_u(q)}\right) \frac{S_0^\kappa - S_l(q)^\kappa}{S_l(q)^\kappa - S_u(q)^\kappa}\right] & \text{if } a \neq 0 \end{cases} \tag{17}$$

where $a = r - \frac{\sigma^2}{2}$ *and* $\kappa = -\frac{2a}{\sigma^2}$.

Figure 5 plots the distribution of the first-exit time, τ for a CI put with $r = 2\%$, $\sigma = 67\%$, $S_0 = K = 100$, and $q = 5$. Here, $\bar{\tau} = 1.6$ months, $\sqrt{\mathrm{Var}(\tau)} = 0.11$, and $\mathbb{E}[|\tau - \bar{\tau}|] = 0.08$.

8.2 Practical estimation from ATM IVs

Given At-The-Money (ATM) implied volatilities $\hat{\sigma}(T)$ for fixed terms $T_1 < \ldots < T_n$ and a desired effective time horizon $\tau \in [T_i, T_{i+1})$, the squared constant volatility implied by the perpetual contract can be approximated by linearly interpolating the total variances $(T\hat{\sigma}^2(T))$ derived by the market implied volatilities:

$$\sigma_{\text{eff}}^2 \tau \approx \tilde{\sigma}_{\text{eff}}^2 \tau = \hat{\sigma}^2(T_i) T_i + \frac{\hat{\sigma}^2(T_{i+1}) T_{i+1} - \hat{\sigma}^2(T_i) T_i}{T_{i+1} - T_i} (\tau - T_i) \equiv w(\tau) \tag{18}$$

The effective squared volatility can be estimated ex ante for a desired first-exit time distribution:

$$\mathbb{E}[\tilde{\sigma}_{\text{eff}}^2] = \mathbb{E}\left[\frac{w(\tau)}{\tau}\right] \approx \frac{w(\bar{\tau})}{\bar{\tau}} \tag{19}$$

When $\bar{\tau}$ is a function of σ^2_{eff}, the problem becomes a fixed-point equation. Specifically, Eq. (19) must be solved for σ^2_{eff} as a function of itself: $\sigma^2_{\text{eff}} = \frac{w(\bar{\tau}(\sigma^2_{\text{eff}}))}{\bar{\tau}(\sigma^2_{\text{eff}})}$.

▶ **Theorem 8.** *The estimate* $\tilde{\sigma}^2_{\text{eff}} \approx \frac{w(\bar{\tau})}{\bar{\tau}}$ *yields root mean squared error and mean absolute deviation*

$$RMSE \leq M\sqrt{\text{Var}(\tau)} \tag{20}$$

$$MAD \leq M\mathbb{E}[|\tau - \bar{\tau}|] \tag{21}$$

where $M = \max_i \sup_{\tau \in [T_i, T_{i+1})} \left| \frac{d}{d\tau} \left(\frac{w(\tau)}{\tau} \right) \right|$.

Hence, when the total variance derived from market-implied volatilities (which are approximately linear in log-Moneyness $\log(K/S)$) has a small slope, the approximation error is small. Figure 6 plots the distribution of $\tilde{\sigma}^2_{\text{eff}}$, its mean, and the approximation $\frac{w(\bar{\tau})}{\bar{\tau}}$ (for the same CI put as in Figure 5) using ETH ATM 7-day, 30-day, 90-day, and 180-day IVs. Figure 7 plots the $RMSE$ and MAD (as a percentage of $\frac{w(\bar{\tau})}{\bar{\tau}}$) for the approximation $\tilde{\sigma}^2_{\text{eff}} \approx \frac{w(\bar{\tau})}{\bar{\tau}}$ for the period of Jan 2024-Feb 2024.

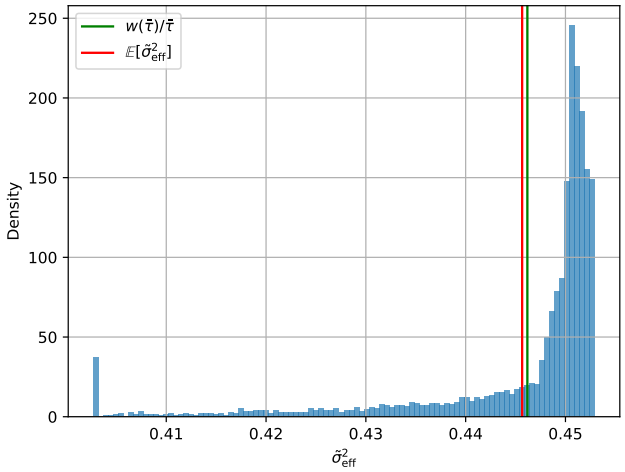


Figure 6 Distribution of σ^2 for $r = 2\%$, $\sigma = 61\%$, $K = 100$, $q = 5$ using ETH ATM IVs.

8.3 Interpretation for Liquidity Providers

The volatility calibration framework enables liquidity providers (LPs) to estimate future loss-versus-rebalancing (LVR) of a concentrated AMM position using observable option market information. By associating the funding fee q of a perpetual CI put with its expected lifetime $\bar{\tau}(q)$, and mapping this to market-implied volatilities, LPs can extract an estimate for the effective squared volatility $\tilde{\sigma}^2_{\text{eff}}$ that governs the dynamics of the position and its underlying asset.

As the funding rate q increases, the continuation band $[S_\ell(q), S_u(q)]$ narrows, leading to shorter expected lifetimes $\tau(q)$ for the CI put. Conversely, smaller q implies wider bands and longer-lived options. Since market-implied volatilities are typically flatter at long durations, we can observe that:

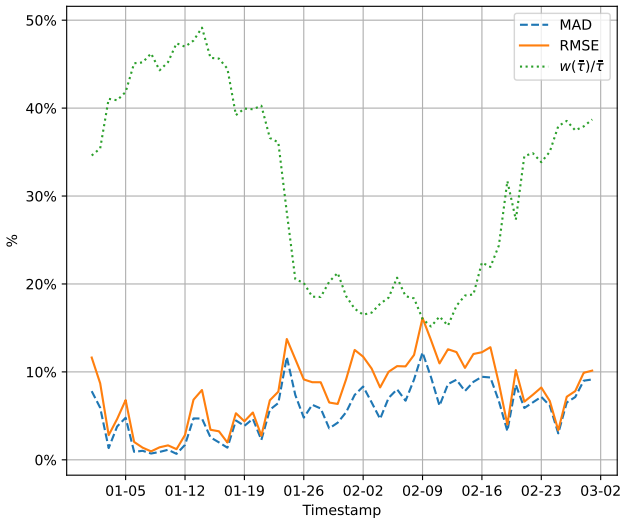


Figure 7 MAD and RMSE for $r = 5\%$, $K = 100$, $q = 40$, where σ^2_{eff} is derived by fixed point methods. IV data is from ETH ATM IVs for Jan-Feb 2024.

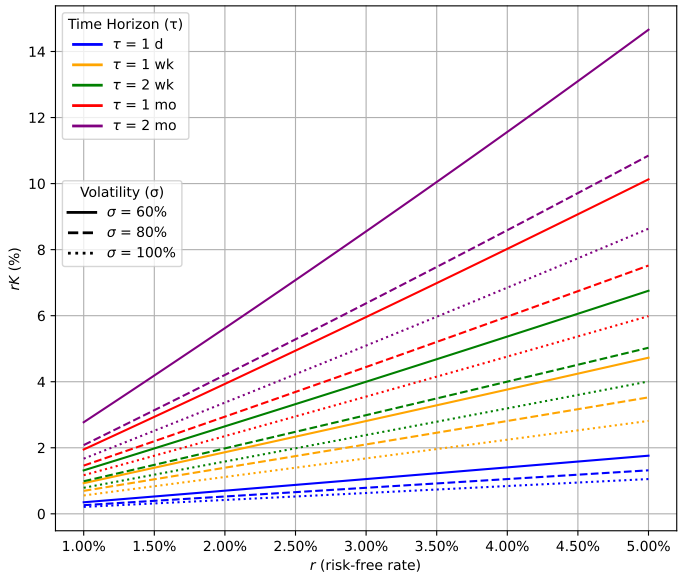


Figure 8 Residual upper bound rK as a percentage of q for various time horizons and σ_{eff} exposures.

- **Short durations** (high q) correspond to low maturity implied volatilities, where the IV curve is more curved and error-prone. More fine-grained market data is required here for better estimates.
- **Long durations** (low q) correspond to long-dated IVs, where the volatility curve is typically flatter. The estimate $\tilde{\sigma}^2_{\text{eff}} = \frac{w(\tau)}{\tau}$ is then less sensitive to the exact value of τ, so M is small and the approximation of the mean is more robust to variation in τ. Long durations also tend to have a tighter concentration of realised $\tilde{\sigma}^2_{\text{eff}}$ around its mean for

Table 1 Funding fee q, resulting CFAMM band (S_l, S_u), and residual bound rK, for $r = 0.05$, for desired $\bar{\tau}$ under varying σ_{eff} exposures.

		% of K				% of q
$\bar{\tau}$	σ_{eff}	q (token$_1$/yr)	$S_l(q)$	$S_u(q)$	Width	rK
1 d	60%	284%	97%	103%	6%	2%
1 d	80%	380%	96%	104%	8%	1%
1 d	100%	475%	95%	105%	10%	1%
1 wk	60%	106%	92%	109%	17%	5%
1 wk	80%	142%	90%	112%	22%	4%
1 wk	100%	178%	87%	115%	28%	3%
2 wk	60%	74%	89%	113%	24%	7%
2 wk	80%	99%	86%	118%	32%	5%
2 wk	100%	125%	83%	123%	40%	4%
1 mo	60%	49%	85%	120%	35%	10%
1 mo	80%	66%	80%	128%	47%	8%
1 mo	100%	84%	76%	136%	60%	6%
2 mo	60%	34%	79%	130%	51%	15%
2 mo	80%	46%	74%	142%	69%	11%
2 mo	100%	58%	68%	157%	88%	9%

the same reason, meaning the mean is a good ex ante estimate.

If a liquidity band is chosen to replicate the delta of a single perpetual CI put – using the squared volatility estimate – then the LP incurs a predictable LVR almost equal to the funding rate q. This transforms an otherwise stochastic adverse-selection cost into a predictable fixed cost per unit time, simplifying the LP's decision-making. For instance, given a desired expected time-horizon, an LP can estimate the effective term volatility, which in turn informs the liquidity band selection. Table 1 shows the different band widths corresponding to expected time horizons and effective term volatilities and Figure 8 shows the LVR $- q$ residual term upper bound. Conversely, given a liquidity band, an LP can estimate the position's expected effective time horizon using numerical fixed point methods. Combined with the results of Section 6, the effective term volatility estimation provides a practical framework for LPs to choose bands that realize a desired holding period $\tau(q)$ and predictable LVR, or to estimate expected forward-looking LVR for a desired liquidity band.

9 Conclusion

This work introduces a novel decomposition of a concentrated CFAMM position into a continuum of perpetual American continuous-installment (CI) put options, offering the first closed-form equivalence between the funding mechanics of perpetual American CI options and the loss-versus-rebalancing cost faced by concentrated AMM liquidity providers, providing as option-theoretic interpretation of LVR.

By exploiting the limiting behavior of CI option valuations as the installment rate grows large, the paper constructs delta-replicating portfolios that match the AMM exposure exactly. The analysis shows that the funding income from this replicating portfolio, absent in the AMM, is analytically equal to the LVR cost borne by the LP in the AMM. This time-invariant correspondence permits a forward-looking estimation of LVR and provides actionable design rules for selecting position width and shape.

Beyond theoretical insight, the framework yields practical tools for LPs. The discrete error analysis confirms that a small collection of finite-q CI puts suffices to replicate AMM delta within tight error bounds, making implementation of the replicating portfolio feasible for LPs wishing to immunize against LVR. Crucially, the analysis also shows that if a liquidity band is chosen to replicate the delta of a single perpetual CI put, the LP incurs a predictable LVR equal approximately to the funding rate q. This converts a stochastic adverse-selection cost into a predictable fixed cost per unit time, simplifying LP decision-making when it comes to position shape and width: using market-implied volatility curves, LPs can calibrate the position shape, width, and implied volatility to a desired expected holding period and LVR, or conversely, estimate a position's effective time horizon and LVR given a liquidity band.

This framework opens several avenues for further research. While the present analysis focuses only on liquidity bands centered around the current price (ATM), LPs may, in practice, deploy liquidity asymmetrically about the spot price to expose their positions to greater or lesser volatility. A resulting mismatch of spot and the replicated strike requires extended analysis to capture the entire volatility surface, as opposed to only considering the ATM volatility curve. Alternatively, if on-chain markets for CI options were developed, they could serve as direct hedging instruments and sources of IV data. Furthermore, the Black-Scholes model assumes the underlying asset experiences a constant volatility, which is not supported by market data. In reality, volatilities may be time-dependent or even stochastic. Models like the Heston model or Hull-White model, which extend upon Black-Scholes, may be applied here for analysis under dynamic volatility surfaces and to assess the model's sensitivity to deviations from constant volatility assumptions. Finally, the model assumes continuous-time trading and perfect liquidity. Future work will relax these assumptions to quantify the impact of transaction costs, slippage, and gas fees on the CI funding fee-LVR equivalence.

References

1 Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap V2 Core. `https://uniswap.org/whitepaper-v2.pdf`, 2020. Accessed: 2025-05-25.

2 Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 Core Whitepaper. `https://uniswap.org/whitepaper-v3.pdf`, 2021. Accessed: 2025-05-25.

3 Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

4 Vitalik Buterin. Ethereum white paper. *GitHub repository*, 1(22-23):5–7, 2013.

5 Pierangelo Ciurlia and Camilla Caperdoni. A note on the pricing of perpetual continuous-installment options. *Mathematical Methods in Economics and Finance*, 4(1):11–26, 2009.

6 Joseph Clark. The replicating portfolio of a constant product market. *Available at SSRN*, 2020. `doi:10.2139/ssrn.3550601`.

7 Joseph Clark. The replicating portfolio of a constant product market with bounded liquidity. *Available at SSRN*, 2021. `doi:10.2139/ssrn.3898384`.

8 DeFiLlama. DEX protocols. `https://defillama.com/protocols/dexs`. Accessed: 2025-05-26.

9 Jun Deng, Hua Zong, and Yun Wang. Static replication of impermanent loss for concentrated liquidity provision in decentralised markets. *Operations Research Letters*, 51(3):206–211, 2023. `doi:10.1016/j.orl.2023.03.002`.

10 Dune Analytics. Total number of liquidity pools. `https://dune.com/queries/3243692/5427065`. Accessed: 2025-05-25.

11 Gerald B Folland. *Real analysis: modern techniques and their applications*. John Wiley & Sons, 1999.

12 Masaaki Fukasawa, Basile Maire, and Marcus Wunsch. Weighted variance swaps hedge against impermanent loss. *Quantitative Finance*, 23:901–911, 2023. doi:10.1080/14697688.2023.2202708.

13 hagaetc. DEX metrics – Dune analytics dashboard. https://dune.com/hagaetc/dex-metrics. Accessed: 2025-05-26.

14 Alex Lipton, Vladimir Lucic, and Artur Sepp. Unified approach for hedging impermanent loss of liquidity provision. *CoRR*, 2024. arXiv:2407.05146.

15 Basile Maire and Marcus Wunsch. Market neutral liquidity provision. *Ledger*, 9:73–88, November 2024. doi:10.5195/ledger.2024.389.

16 Jason Milionis, Ciamac C Moallemi, Tim Roughgarden, and Anthony Lee Zhang. Automated market making and loss-versus-rebalancing. *CoRR*, 2024. arXiv:2208.06046.

17 Srisht Fateh Singh, Reina Ke Xin Li, Samuel Gaskin, Yuntao Wu, Jeffrey Klinck, Panagiotis Michalopoulos, Zissis Poulos, and Andreas Veneris. Modeling loss-versus-rebalancing in automated market makers via continuous-installment options, 2025. arXiv:2508.02971.

18 Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0. 8.13, 2018.

A Closed-Form of Perpetual American CI Put Option

Below, we provide closed-form expressions for the perpetual American CI put option. Define

$$\gamma_p := -\frac{2r}{\sigma^2}, \tag{22}$$

$$g := 1 + \frac{rK}{q}. \tag{23}$$

Then, the option price is

$$P_q(S;K) = \alpha_p S + \beta_p S^{\gamma_p} + \frac{q}{r}$$

where the constants α_p and β_p are given by

$$\alpha_p = \left(g^{1-1/\gamma_p} - 1\right)^{-1}, \tag{24}$$

$$\beta_p = -\frac{1}{\gamma_p}\left(\tfrac{q}{r+\sigma^2/2}\right)^{1-\gamma_p}\alpha_p^{\gamma_p}. \tag{25}$$

Let $X_q(S;K) = \frac{\partial}{\partial S}P_q(S;K)$ be the delta of put value, its expression is given by

$$X_q(S;K) = \alpha_p + \beta_p\gamma_p S^{\gamma_p - 1}.$$

Lastly, the lower and upper boundaries, S_ℓ and S_u respectively, have the following closed-form:

$$S_\ell = \frac{q}{r+\sigma^2/2}\left[g - g^{1/\gamma_p}\right],$$

$$S_u = \frac{q}{r+\sigma^2/2}\left[g^{1-1/\gamma_p} - 1\right].$$

B Proofs of Main Theorems

Proof of Lemma 1. Boundary collapse. Let $S_\ell(q)$ and $S_u(q)$ be the exercise and abandonment boundaries in (11) and (12). We will prove that

$$\lim_{q\to\infty} S_\ell(q) = \lim_{q\to\infty} S_u(q) = K.$$

Define $\varepsilon := rK/q$, so $g = 1 + \varepsilon$ and $\varepsilon \downarrow 0$ as $q \uparrow \infty$. Consider the second-order expansion $(1+\varepsilon)^a = 1 + a\varepsilon + \frac{1}{2}a(a-1)\varepsilon^2 + \mathcal{O}(\varepsilon^3)$. Applying it to the two exponents in (11) and (12) yields

$$S_\ell(q) = \frac{q\varepsilon}{r + \sigma^2/2}\left(1 - \tfrac{1}{\gamma_p}\right) + \mathcal{O}(q^{-1}),$$
$$S_u(q) = \frac{q\varepsilon}{r + \sigma^2/2}\left(1 - \tfrac{1}{\gamma_p}\right) + \mathcal{O}(q^{-1}).$$

Because $q\varepsilon = rK$ and $1 - \frac{1}{\gamma_p} = 1 + \sigma^2/(2r)$, both leading terms equal K. Therefore,

$$\begin{aligned}\lim_{q\to\infty} S_\ell(q) &= \lim_{q\to\infty} K + \mathcal{O}(q^{-1}) \\ &= K, \\ \lim_{q\to\infty} S_u(q) &= \lim_{q\to\infty} K + \mathcal{O}(q^{-1}) \\ &= K.\end{aligned}$$

Step-delta limit. The smooth–fit conditions (continuous first order derivative) on the boundaries of the holding region of the option valuation curve from Section 2.3.1 give $X_q(S_\ell; K) = -1$ and $X_q(S_u; K) = 0$. Because X_q is monotone increasing in S between the two boundaries and the interval $S_u(q) - S_\ell(q)$ collapses, we have the *pointwise* limit

$$X_\infty(S; K) := \lim_{q\to\infty} X_q(S; K) \ = -\mathbf{1}_{\{S<K\}}. \tag{26}$$

Thus, as the funding fees of a continuous-installment put becomes large, it transforms into a unit-step-delta contract. ◀

Proof of Theorem 2. For each K, the map $S \mapsto P_q(S; K)$ is continuously differentiable. Moreover, for fixed q, there exists a constant $c_q > 0$ such that $\left|\partial_S P_q(S; K)\right| \le c_q(1 + K)^{-2}|w(K)|$. Hence, the integrand is point-wise dominated by an L^1–function of K. Leibniz's rule [11] yields

$$\partial_S \Pi_q(S) = \int_0^\infty w(K)\, X_q(S; K)\, dK,$$

where $X_q := \partial_S P_q$ is the CI–put delta. For every K, we have

$$\begin{aligned}\lim_{q\to\infty} X_q(S; K) &= X_\infty(S; K) \\ &= -\mathbf{1}_{\{S<K\}}\end{aligned}$$

from Lemma 1. Let $\{q_n\}$ be a sequence such that $q_n \to \infty$. $|X_{q_n}(S; K)| \le 1$ for all n, S and K, and $X_{q_n}(S; K) \to X_\infty(S; K)$ pointwise in K, so dominated convergence theorem [11] applies:

$$\begin{aligned}\partial_S \Pi(S) &= \lim_{n\to\infty} \int_0^\infty w(K) X_{q_n}(S; K) dK \\ &= \int_0^\infty w(K)\, X_\infty(S; K)\, dK \\ &= -\int_S^\infty w(K)\, dK \\ &= V'(S) - V'(\infty) \\ &= V'(S).\end{aligned}$$

Therefore, $\Pi(S) - V(S)$ is delta-neutral. ◀

Proof of Lemma 3. Using closed forms for a CI put boundaries from Appendix A, $g := 1+\varepsilon$, $\varepsilon = rK/q$ and $\gamma_p = -2r/\sigma^2$:

$$S_\ell(q) = \frac{q}{r+\sigma^2/2}\big[g - g^{1/\gamma_p}\big],$$
$$S_u(q) = \frac{q}{r+\sigma^2/2}\big[g^{1-1/\gamma_p} - 1\big].$$

Setting $\alpha := 1/\gamma_p = -\sigma^2/(2r)$ and expanding to second order:

$$g^\alpha = 1 + \alpha\varepsilon + \tfrac{1}{2}\alpha(\alpha-1)\varepsilon^2 + \mathcal{O}(\varepsilon^3),$$
$$g^{1-\alpha} = 1 + (1-\alpha)\varepsilon - \tfrac{1}{2}(1-\alpha)\alpha\varepsilon^2 + \mathcal{O}(\varepsilon^3).$$

Substituting the above into the closed forms of S_ℓ and S_u cancels the first-order terms and the second-order coefficient becomes $\alpha(\alpha-1)$. Therefore, one obtains

$$S_u(q) - S_\ell(q) = \frac{\alpha(\alpha-1)r^2K^2}{(r+\sigma^2/2)\,q} + \mathcal{O}\big(q^{-2}\big). \tag{27}$$

Hence,

$$\lim_{q\to\infty} q\,\big(S_u(q) - S_\ell(q)\big) = \lim_{q\to\infty}\Big[\frac{\alpha(\alpha-1)r^2K^2}{(r+\sigma^2/2)} + \mathcal{O}\big(q^{-1}\big)\Big]$$
$$= \frac{\sigma^2K^2}{2}$$ ◀

Proof of Lemma 4.

Point-wise convergence. Both $\widetilde{\Pi}_q$ and Π *vanish for all* $S \geq b$. Moreover, the CI–put delta satisfies $0 \leq |X_q(S;K)| \leq 1$. Consequently $|\partial_S\widetilde{\Pi}_q(S)| = \big|\sum_i w_i\, X_q(S;K_i)\big| \leq \sum_i |w_i| = 1$ for every q and S. Let $i^\star = i^\star(S,q)$ be the (unique) index with $S \in [S_\ell(q;K_{i^\star}), S_u(q;K_{i^\star})]$. By Lemma (1), $X_q(S;K_{i^\star}) \to -\mathbf{1}_{\{S<K_{i^\star}\}}$, while $X_q(S;K_i) \to 0$ for $i \neq i^\star$. Dominated convergence therefore gives

$$\lim_{q\to\infty} \partial_S\widetilde{\Pi}_q(S) = -\int_a^b w(K)\,\mathbf{1}_{\{S<K\}}\,dK = X(S).$$

For any $S \leq b$

$$\widetilde{\Pi}_q(S) = \widetilde{\Pi}_q(b) - \int_S^b \partial_S\widetilde{\Pi}_q(u)\,du.$$

Because $\widetilde{\Pi}_q(b) = 0$ for all q and the integrands converge point-wise while being uniformly bounded by 1, dominated convergence implies $\widetilde{\Pi}_q(S) \to \Pi(S)$.

Limit of $w_j q$. From the statement of the Lemma,

$$w_j \;=\; X\big(S_u(q;K_j)\big) - X\big(S_\ell(q;K_j)\big)$$

Therefore,

$$\begin{aligned}
\lim_{q\to\infty} w_j q &= \lim_{q\to\infty} q\left[X\big(S_u(q;K_j)\big) - X\big(S_\ell(q;K_j)\big)\right] \\
&= \lim_{q\to\infty} q(S_u(q;K_j) - S_\ell(q;K_j))\frac{X\big(S_u(q;K_j)\big) - X\big(S_\ell(q;K_j)\big)}{(S_u(q;K_j) - S_\ell(q;K_j))} \\
&= \lim_{q\to\infty} q(S_u(q;K_j) - S_\ell(q;K_j)) \lim_{q\to\infty} \frac{X\big(S_u(q;K_j)\big) - X\big(S_\ell(q;K_j)\big)}{(S_u(q;K_j) - S_\ell(q;K_j))} \\
&= \lim_{q\to\infty} q(S_u(q;K_j) - S_\ell(q;K_j))X'(K_j) \\
&= \frac{\sigma^2 S_t^2}{2} X'(S_t)
\end{aligned}$$

◀

Proof of Theorem 5. From Lemma 4, we can approximate the continuous portfolio by a discrete portfolio of CI puts. For a specific funding fee q, at each time t, there is only one active option j with weight w_j, price $P_q(S_t, K_j)$. The total funding fee accured over $[t, t+dt]$ is:

$$\mathrm{dFee}_t^q = w_j q dt$$

Again from Lemma 4, we have $\lim_{q\to\infty} w_j q = \frac{\sigma^2 S_t^2}{2} X'(S_t)$. This implies that

$$\lim_{q\to\infty} \mathrm{dFee}_t^q = \lim_{q\to\infty} w_j q dt = \frac{\sigma^2 S_t^2}{2} X'(S_t) dt$$

This exactly matches Eq. (6). Hence,

$$\mathrm{dFee}_t = \mathrm{dLVR}_t$$

Integrating over $t \in [0, T]$, we get:

$$\mathrm{Fee}_t|_0^T = \int_0^T \mathrm{dFee}_t = \int_0^T \mathrm{dLVR}_t = \mathrm{LVR}|_0^T$$

◀

Proof of Theorem 6. Equation (7) gives

$$\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 P_q}{\partial S^2} + rS\frac{\partial P_q}{\partial S} - r\,P_q \;=\; q, \qquad S \in (S_\ell, S_u).$$

and

$$\begin{aligned}
\mathrm{dLVR}_t &= \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 P_q}{\partial S^2} dt \\
&= qdt - r(S\frac{\partial P_q}{\partial S} - P_q)dt
\end{aligned}$$

Therefore, the residual term $\epsilon(t) = -r(S\frac{\partial P_q}{\partial S} - P_q)$. Note that we use S instead of S_t for brevity. $\epsilon(\cdot)$ is a function of t.

Bounding $|\epsilon(t)|$. In the region $S \in (S_\ell, S_u)$,

$$\frac{\partial \epsilon(t)}{\partial S} = -rS\frac{\partial^2 P_q}{\partial S^2}$$

Since $\frac{\partial^2 P_q}{\partial S^2}$ is always non-negative,

$$\frac{\partial \epsilon(t)}{\partial S} = -rS\frac{\partial^2 P_q}{\partial S^2} \leq 0$$

Therefore, $\epsilon(t)$ is monotonically decreasing with S. Moreover, when evaluated at $S = S_\ell$,

$$\begin{aligned}\epsilon(t) &= -r((-1 \cdot S_\ell) - (K_* - S_\ell)) \\ &= rK_*.\end{aligned}$$

and when $S = S_u$,

$$\epsilon(t) = 0.$$

Hence, $|\epsilon(t)| \leq rK_*$. ◀

C Effective Time Horizon $\bar{\tau}(q)$

The proof of Theorem 7 can be found in the extended version [17].

D Volatility Estimation Error Bounds

Proof of Theorem 8. The root mean squared error of the estimate $\tilde{\sigma}^2_{\text{eff}} \approx \frac{w(\bar{\tau})}{\bar{\tau}}$ is

$$RMSE = \sqrt{\mathbb{E}\left[\left(\tilde{\sigma}^2_{\text{eff}} - \frac{w(\bar{\tau})}{\bar{\tau}}\right)^2\right]}$$

Let $f(\tau) = \tilde{\sigma}^2_{\text{eff}} = \frac{w(\tau)}{\tau}$, which is differentiable almost everywhere, with

$$f'(\tau) = \frac{m_i\tau - w(\tau)}{\tau^2} \qquad m_i \equiv \frac{\hat{\sigma}^2(T_{i+1})T_{i+1} - \hat{\sigma}^2(T_i)T_i}{T_{i+1} - T_i} \qquad f(0) = m_0$$

f is Lipschitz continuous, so let $M = \max_i \sup_{\tau \in [T_i, T_{i+1})} |f'(\tau)|$. Then, $|f(\tau) - f(\bar{\tau})| \leq M|\tau - \bar{\tau}|$ and

$$RMSE = \sqrt{\mathbb{E}[(f(\tau) - f(\bar{\tau}))^2]} \leq \sqrt{\mathbb{E}[M^2(\tau - \bar{\tau})^2]} = \sqrt{M^2 \operatorname{Var}(\tau)} = M\sqrt{\operatorname{Var}(\tau)} \tag{28}$$

Similarly, the mean absolute deviation is

$$MAD = \mathbb{E}\left[|f(\tau) - f(\tilde{\tau})|\right] \leq M\mathbb{E}[|\tau - \bar{\tau}|] \tag{29}$$

◀

Mechanism Design for Automated Market Makers

T-H. Hubert Chan[1]
University of Hong Kong, Hong Kong

Ke Wu
University of Michigan, Ann Arbor, MI, USA
Carnegie Mellon University, Pittsburgh, PA, USA

Elaine Shi
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Blockchains have popularized automated market makers (AMMs), applications that run on a blockchain, maintain a pool of crypto-assets, and execute trades with users governed by some pricing function. AMMs have also introduced a significant challenge known as the Miner Extractable Value (MEV). Specifically, miners who control the contents and sequencing of transactions in a block can extract value by front-running and back-running users' transactions, creating arbitrage opportunities that guarantee them risk-free returns. MEV not only harms ordinary users, but more critically, encourages miners to auction off favorable transaction placements to users and arbitragers. This has fostered a more centralized off-chain eco-system, departing from the decentralized equilibrium originally envisioned for the blockchain infrastructure layer.

In this paper, we consider how to design AMM mechanisms that eliminate MEV opportunities. Specifically, we propose a new AMM mechanism that processes all transactions contained within a block according to some pre-defined rules, ensuring that some constant potential function is maintained after processing the batch. We show that our new mechanism satisfies two tiers of guarantees. First, for legacy blockchains where each block is proposed by a single (possibly rotating) miner, we prove that our mechanism satisfies arbitrage resilience, i.e., a miner cannot gain risk-free profit. Second, for blockchains where the block proposal process is decentralized and offers sequencing-fairness, we prove a strictly stronger notion called strategy proofness – roughly speaking, we guarantee that any individual user's best response is to follow the honest strategy.

Our results complement prior works on MEV resilience in the following senses. First, prior works have shown impossibilities to address MEV entirely at the consensus level. Our work demonstrates a new paradigm of mechanism design at the application (i.e., smart contract) layer to ensure provable guarantees of strategy proofness. Second, many works have attempted to augment the underlying consensus protocol with extra properties such as sequencing fairness. While most previous works heuristically argued why these extra properties help to mitigate MEV, our work demonstrates in a mathematically formal manner how to leverage such consensus-level properties to aid the design of strategy-proof mechanisms.

2012 ACM Subject Classification Theory of computation → Algorithmic mechanism design; Theory of computation → Algorithmic game theory

Keywords and phrases Mechanism design, game theory, strategy proof, blockchain

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.7

Related Version *Full Version*: `https://arxiv.org/abs/2402.09357` [25]

Funding *T-H. Hubert Chan*: T-H. Hubert Chan was partially supported by the Hong Kong RGC grants 17203122 and 17202121.
Ke Wu: Ke contributed to this work while she was a Ph.D. student at CMU. See the grant acknowledgements of Elaine Shi.

[1] Author ordering is randomized.

© T-H. Hubert Chan, Ke Wu, and Elaine Shi;
licensed under Creative Commons License CC-BY 4.0

Elaine Shi: This work is in part supported by NSF awards 2212746, 2044679, 1704788, a Packard Fellowship, a generous gift from the late Nikolai Mushegian, a gift from Google, and an ACE center grant from Algorand Foundation.

1 Introduction

Blockchains have popularized decentralized finance (DeFi), with one of its key applications being Decentralized Exchanges (DEX) based on Automatic Market Makers (AMMs) [14]. As of March 2021, the top six AMMs, including Uniswap, Balancer, and others, collectively held approximately $15 billion in crypto assets [46]. A typical AMM exchange maintains a pool of capital called the "liquidity pool" with two crypto-assets X and Y. A smart contract specifies the rules how users can trade assets with the pool. For example, one commonly adopted rule is a *constant-product potential function* defined as follows. Let $\mathsf{Pool}(x, y)$ denote the pool's state where $x \geq 0$ and $y \geq 0$ represent the units of X and Y held by the pool, respectively. A constant product potential requires that $x \cdot y = C$ for some constant $C > 0$. This means that if a user buys δx amount of X from the pool, it needs to pay $-\delta y$ amount of Y such that $(x - \delta x)(y - \delta y) = C$.

DeFi applications such as AMMs have introduced opportunities for miners to profit, often in a risk-free manner, by front-running and/or back-running the users' transactions, a phenomenon known as Miner Extractable Value (MEV). Despite the decentralized nature of blockchains, the block proposal process in mainstream consensus protocols remains centralized. For each block, a single selected miner[2] has unilateral control over which transactions are included and their sequencing. By exploiting this capability, miners can profit, often in a risk-free manner. For example, in a sandwich attack [37, 41, 46, 51], a miner identifies a victim user attempting to purchase a crypto asset X at a maximum price of r, and inserts a $\mathsf{Buy}(X)$ transaction just before the victim's buy order and a $\mathsf{Sell}(X)$ transaction immediately after. Since purchasing X increases its price, the miner effectively buys at a lower price through front-running, forces the victim to buy at the worst possible price r, and then sells at a higher price through back-running, locking in a profit. Beyond sandwich attacks, miners can also take advantage of more sophisticated arbitrage opportunities to profit [37, 41, 49].

MEV is widely recognized as one of the most important challenges for blockchains today for several reasons. First, since MEV is extracted at the expense of users, it effectively increases the barrier of entry for ordinary users to engage with DeFi applications. Second, MEV undermines the stability and security of the underlying consensus protocol [24, 41, 50, 51]. Specifically, miners may be incentivized to fork the blockchain if doing so offers higher MEV rewards than standard block rewards. Third, the block producer's power in deciding the block contents and sequencing has given rise to an off-chain economy. Block producers enter private contracts with arbitragers and users alike, offering them favorable positions in the block. These private off-chain contracts have led to a centralizing effect in the underlying layer 1 (i.e., the consensus layer), causing the *de facto* layer 1 [31] to operate in a manner that significantly departs from the intended design, and its equilibrium behavior is not understood. A recent empirical measurement showed that today, more than 85% of the Ethereum blocks are built by two block producers [47].

[2] In this paper, no matter whether the underlying consensus is proof-of-stake or proof-of-work, we generically refer to a consensus node that produces blocks as a "miner" or a "block producer".

1.1 Our Results and Contributions

It would have been compelling if there existed a way to solve the MEV problem entirely at the consensus layer, without having to modify the existing applications. Unfortunately, the impossibility results in several previous works [13,30] can be interpreted to mean that solving the MEV problem (in its most general form) entirely at the consensus layer, subject to today's architecture, is impossible. On the other hand, many works aimed to offer strengthened guarantees at the consensus layer such as sequencing fairness [2,10,34–36] or some form of privacy [17,29]. While it is widely believed that these properties help to mitigate MEV, there has been relatively little formal investigation on how we can take advantage of these extra consensus-level properties in mechanism design.

Therefore, in this paper, we ask the following natural questions:

- Instead of working entirely at the consensus level, can we rely on mechanism design at the application level (i.e., smart contract level) to obtain provable guarantees of MEV resilience?
- How do strengthened guarantees at the consensus layer aid the design of strategy-proof mechanisms at the application layer?

Specifically, we ask these questions in the context of Automated Market Makers (AMMs) which represent one of the most important DeFi applications. We now summarize our results and contributions.

A mechanism design approach towards mitigating MEV. Applying the philosophy of mechanism design at the application layer, we want to design an AMM mechanism that removes MEV opportunities and provides strategy proofness by construction.

We devise a new AMM mechanism (to be executed as a smart contract on chain) with the following abstraction. In our mechanism, the pool holds two crypto-assets X and Y. A user can trade with the pool by posting a buy/sell *order* specifying how much of X (or Y) they want to buy/sell, and their worst acceptable exchange rate. When a new block arrives, the mechanism takes the block of orders as input, and applies an allocation rule to all orders contained within the block. The allocation rule decides which orders are partially or completely satisfied, and at what price. The mechanism maintains the following invariant: the pool's beginning state denoted $\mathsf{Pool}(x_0, y_0)$ and end state $\mathsf{Pool}(x_1, y_1)$ is guaranteed to satisfy some "natural" potential function Φ (e.g., the constant-product function mentioned above).

Our mechanism offers two tiers of guarantees depending on whether the properties of the underlying consensus. Specifically, we consider two models: 1) the *plain* model, capturing today's mainstream consensus protocols where for any particular block, the inclusion and sequencing of transactions are determined by a single, possibly strategic block producer; and 2) the *weak fair-sequencing* model, intended to capture a new generation of consensus protocols that offer sequencing fairness guarantees [2,10,34–36], e.g., through a decentralized sequencer. Our mechanism achieves the following desirable, two-tier properties:

1. **Arbitrage free** in the plain model. We guarantee that no arbitrager (e.g., user, block producer, or any intermediary) can gain risk-free profit, *even when the arbitrager (e.g., block producer) has unilateral control over the block contents and transaction sequencing.* Here, risk-free profit happens when an arbitrager can gain in one asset without losing in another with probability 1.
2. **Strategy proofness** in the weak fair-sequencing model. In the weak fair-sequencing model, our mechanism not only achieves arbitrage resilience, but also guarantees strategy proofness. Specifically, strategy proofness means that users are incentivized to report

their true demand and true belief of the relative value of the two crypto-assets, and no strategic behavior allows a user to gain. Later, we prove that *strategy proofness is a strictly stronger notion than arbitrage resilience* (Fact 1).

Our weak fair-sequencing model is meant to capture a decentralized sequencer that sequences the orders based on their arrival times (importantly, not based on the orders' submission time). While this model places additional constraints on the strategy space in comparison with the plain model, it **does not prevent front-running** and thus **does not trivialize** the mechanism design problem. Notably, in this model, a strategic user or miner can still wait for a victim to submit its order, and then immediately submit a dependent order. The strategic order can even front-run the victim's order if the strategic user's network is faster, similar to a rushing attack in the cryptography literature [19, 20].

Our work is also *among the first to formally articulate, from a mechanism designer's perspective, how extra properties at the consensus level lend to the design of strategy-proof mechanisms at the smart contract layer.*

Conceptual contributions. We put forth new modeling and definitions, which capture a mechanism design problem of a decentralized nature. In comparison with the classical mechanism design literature, our model and strategy spaces capture the "permissionless" nature of blockchains. Specifically, a strategic player may not only report its valuation/demand untruthfully, but also inject fake orders or post multiple orders. Such strategies are possible because the mechanism does not have a-priori knowledge of the number or the identities of the bidders. Compared to closely related works, our model circumvents the strong impossibilities shown by Ferreira and Parkes [30], because we relax some unrealistic restrictions they impose – see Section 1.2 for more details. Therefore, we believe that our model is better suited for capturing real-world mechanism design at the smart contract layer, particularly for AMMs. Our new model and definitions naturally give rise to many interesting open questions which we discuss in Section 1.3.

1.2 Comparison with Related Work

Comparison with most closely related work. Ferreira and Parkes [30] showed that in an overly pessimistic model as explained below, achieving arbitrage resilience is impossible, let alone strategy proofness. However, their impossibility result holds only in an overly stringent model that does not reflect the real-life design space. Specifically, when interpreted in our new framework, their impossibility holds only if the AMM mechanism has to respect the following specific structure: sort the incoming orders according to some rules (called "verifiable sequencing rules" in their paper), and run a legacy AMM contract that processes the sorted orders sequentially, such that the constant potential function must be maintained after executing each order. In comparison, in our model, the constant potential function only needs to be maintained at the end of processing the entire batch. Because of their strong impossibility result, Ferreira and Parkes [30] showed how to achieve a weaker guarantee in their model, that is, if the miner made risk-free profit, then the user should enjoy a price that is at least as good as if its order were the only one in the block.

Li et al. [39] inherit the same model as Ferreira and Parkes [30], and they study what is the profit-maximizing strategy for the miner and the implications for the users when the miner adopts the optimal strategy. Like Ferreira and Parkes [30], they adopt an overly restrictive model which requires them to give up on achieving arbitrage resilience, let alone strategy proofness.

One of the contributions we make is exactly to recognize why the existing models are too stringent and unrealistic, and suggest a better model for the study of MEV-resilient mechanism design.

Batch clearing at uniform price does not guarantee strategy proofness. A line of works explored the idea of batch clearing at uniform price. We stress that batch clearing at uniform price [1, 21, 22] does not automatically guarantee strategy proofness. For example, suppose there are many eligible orders and the mechanism can clear only a subset of them. If the mechanism selects the subset based on the declared valuation, then a strategic user can lie about its valuation to get selected.

The concurrent and independent work of Canidio and Fritsch [21, 22] suggested batch-clearing at a uniform price and using a different potential function than Uniswap's constant-product function. Their approach satisfies arbitrage resilience, but it does not satisfy strategy proofness *even when the miner is trusted to behave honestly*, for the reason stated above. In fact, Canidio and Fritsch [21, 22] does not even fully specify which subset of orders to clear when there are many eligible candidates. The work of Zhang et al. [48] also observed that batch clearing alone does not imply strategy proofness. In fact, they investigated the optimal strategy under batch clearing. The prior work of Ramseyer et al. [43] also considered batch exchanges that clear at a uniform price. Like Canidio and Fritsch [21, 22], their work does not provide strategy proofness, even when the miner is fully trusted. Besides batch trading at uniform pricing, other forms of batch trading [23] have also been considered, but they also do not satisfy our notion of strategy proofness.

Related works that do not address MEV. Milionis, Moallemi, and Roughgarden [40] consider how to design the demand curve for a market maker to maximize profit and meanwhile incentivize truthful reporting. Their work is of a completely different nature than ours, since they *do not aim to address the problem of MEV*. Specifically, they consider a simple model where users directly submit orders to the market maker. They *do not consider any arbitrage strategy* where users or miners try to front-run or back-run others' orders to make profit.

Bartoletti et al. [15] studied the miner's optimal MEV strategy under transaction reordering. Their work also does not provide a solution to mitigate or address MEV.

Understanding the impact of MEV. A line of works have empirically or theoretically investigated the profitability or impact of MEV [11, 12, 16, 37, 41, 42, 49, 52].

Empirical approaches towards mitigating MEV. Another line of work suggest that the users themselves take action to mitigate MEV, either by setting their slippage limits more cleverly [50], or by exploiting arbitrage opportunities themselves to lower their transactional costs [32]. There are also various blog posts on online forums that suggest alternative designs [33, 38]. However, these works are empirical and do not lend to the theoretical understanding of the equilibrium behavior of the eco-system.

Both academic research and real-world blockchain projects have made an effort to build decentralized sequencers [2, 10, 34–36], or encrypted mempools [17, 29]. The former approach removes the ability for a single block proposer to decide the block contents and sequencing, and achieves some form of sequencing fairness [34–36][3]. The latter approach allows users to

[3] Sequencing fairness is also commonly referred to as "order fairness". In this paper, we use the term

submit transactions in committed or encrypted format, which makes it harder for miners to front-run and back-run transactions. However, from a mechanism design perspective, we still lack mathematical understanding to what extent these new consensus/cryptographic abstractions can help us mitigate MEV and achieve strategy-proof DeFi mechanisms. In this sense, our work is *among the first to mathematically articulate how to rely on "sequencing fairness" to achieve strategy-proofness by construction.*

Sequencing fairness. A line of works [2,10,34–36] have studied how to achieve order fairness in consensus. Numerous blockchain projects are also building decentralized sequencers [3–8] which can be one approach for achieving sequencing fairness. Some works [9,28] have also pointed out the price of sequencing fairness such as loss in welfare. Improving the underlying mechanisms for achieving sequencing fairness is outside the scope of this paper. We also leave it as future work to study how to optimize social welfare under strategy proofness.

Other related works. There is a recent line of work on transaction fee mechanism (TFM) design [26,44,45]. This line of work aims to design mechanisms such that users, miners, and user-miner coalitions are incentivized to behave honestly. However, the current modeling approach of this line of work captures only the utilities at the consensus layer. They cannot capture ordering and application-level MEV. The recent work of Bahrani et al. [13] showed strong impossibility results for fully solving this problem at the TFM-layer alone. In this sense, our work complements the line of work on TFM design by taking an application-level (i.e., smart-contract-level) approach towards achieving strategy proofness by construction.

1.3 Scope and Open Questions

Just like the recent literature including Ferreira and Parkes [30] and Li et al. [39], the scope of the present paper is restricted to how defend against MEV in a *standalone* two-asset AMM mechanism. We begin with the standalone setting because it serves as a necessary basis for understanding the compositional setting with multiple instances. It is also a widely adopted approach in the mechanism design and cryptography literature to begin with the standalone setting first. We leave it as an interesting open question how to achieve provable game-theoretic guarantees in a compositional setting where multiple instances can interact with each other. We stress that in general, unlike in the cryptography literature where there are composable notions of security [19,20] which makes composition worry-free, most notions in game theory do not naturally compose, and composition is typically treated on a case-by-case basis.

Our new model and definitions give rise to many interesting open problems. One interesting question is how to extend our results to AMMs with multiple assets. Another interesting question is whether it is possible to achieve the stronger notion of strategy proofness without relying on sequencing fairness. Currently, our model assumes that all the orders are submitted in the clear, and we thus define strategy proofness in the ex post setting. A future direction is to understand how to define and achieve strategy proofness in an MPC-assisted model [45] or an "encrypted mempool" [17,29] model where transactions are submitted in encrypted or committed format. Another interesting question is whether we can design strategy-proof AMM mechanisms when the execution syntax is atomic rather than partial fulfillment like what we consider.

"sequencing fairness" to avoid collision with the usage of "order" to mean a trade proposal.

We believe that the modeling work in our paper helps to lay the theoretical groundwork for exploring questions such as above in the future.

2 Definitions

2.1 Swap Mechanism for AMMs

A swap mechanism for a pair of assets (X, Y) has a state (also called the *pool state*) denoted $\mathsf{Pool}(x, y)$ where x and y are non-negative values that represent the amount of each asset currently held by the mechanism. A user can submit an order to trade with the mechanism in two ways: either buy X and pay in Y, or buy Y and pay in X. Suppose the user buys δx units of X and pays δy units of Y, then the updated state after the trade will become $\mathsf{Pool}(x - \delta x, y + \delta y)$.

Order. Each order is of the form (t, v, r, α) where

- $t \in \{\mathsf{Buy}(X), \mathsf{Buy}(Y), \mathsf{Sell}(X), \mathsf{Sell}(Y)\}$ is the type of the order indicating that the user wants to buy or sell and which asset;
- v is a non-negative value that denotes the maximum amount of the user wants to buy or sell;
- r denotes the user's acceptable exchange rate, i.e., the user believes that each unit of X is worth r units of Y. For example, if the order is of type $\mathsf{Buy}(X)$, then the user is willing to pay at most r units of Y for each unit of X; if the type is $\mathsf{Sell}(Y)$, then $1/r$ is the minimum asking price in X for each unit of Y.
- α is an arbitrary string denoting any additional auxiliary information, e.g., the submitter's identity, timestamping information, position in the block, and so on.

Note that given an order of the form $(\mathsf{Sell}(Y), v, r, _)$[4], another way to view it is that the user wants to buy X; it is willing to pay at most r units of Y for each unit of X; moreover, it wants to buy as many units of X as possible subject to a capital of v units of Y. Henceforth, for an order of the type $\mathsf{Buy}(X)$ or $\mathsf{Sell}(X)$, we say that X is the *primary asset* of the order.

Swap mechanism. A possibly randomized (partial fulfillment) swap mechanism should define the following rules[5]:

- *Honest strategy.* Given a user's private type T, the initial state $\mathsf{Pool}(x, y)$, the honest strategy, often denoted $HS(x, y, T)$, outputs a vector of orders the user should submit. A user's private type T can contain information such as how many units of X and Y it currently holds, and the user's private valuation of the exchange rate between X and Y.
- *Allocation rule.* The allocation rule receives as input an initial state $\mathsf{Pool}(x, y)$, a list of orders, and for each order (t, v, r, α), it outputs the following:
  - the amount $v' \in [0, v]$ of primary asset that has been fulfilled – note that the fulfillment can be partial;
  - an average exchange rate $r' > 0$ at which the order was fulfilled. For a $\mathsf{Buy}(X)$ order, it means that the user pays $v' \cdot r'$ units of Y in exchange for v' units of X. For a $\mathsf{Sell}(Y)$ order, the user obtains v'/r' units of X for the v' units of Y sold. We require that for a $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$ order, $r' \le r$, i.e., the purchase price cannot be higher than the specified maximum rate r; and for a $\mathsf{Buy}(Y)/\mathsf{Sell}(X)$ order, $1/r' \le 1/r$.

[4] Here, the ignore symbol _ means that we are ignoring the content of this field in the current context.
[5] In Section 5, we pose the open question of how to achieve strategy proofness for all-or-nothing fulfillment mechanisms.

In a real-world instantiation, the pool state is recorded on the blockchain, and the allocation rule is executed on the blockchain (e.g., in the form of a smart contract).

Invariant on pool state. We consider swap mechanisms that satisfy the following invariant on pool state. Given some initial state $\mathsf{Pool}(x, y)$, and the outcome output by the allocation rule, one can uniquely determine the ending state $\mathsf{Pool}(x', y')$. We require that initial and ending pool state must satisfy some constant potential function, that is, $\Phi(x', y') = \Phi(x, y)$. We define potential function and requirements on the potential function below.

Potential function. We consider swap mechanisms that respect a constant potential function $\Phi(\cdot, \cdot)$. Specifically, suppose the pool's initial state is $\mathsf{Pool}(x, y)$, and changes to $\mathsf{Pool}(x', y')$ after the mechanism processes a batch of orders. Then, it must be that

$$\Phi(x, y) = \Phi(x', y').$$

In practice, the most widely adopted approach is a constant-product market maker where $\Phi(x, y) = x \cdot y$. In other words, suppose the initial pool state is (x, y) and some user buys δx amount of X, then it must pay $-\delta y$ units of Y where δy can be calculated by solving the following equation:

$$(x - \delta x)(y - \delta y) = xy.$$

Assumptions on the potential function. We assume the standard assumption that the potential function $\Phi(\cdot, \cdot)$ is increasing, differentiable, and concave.

Market exchange rate. In our swap mechanism, we will make use of the notion of a market exchange rate, as defined below.

▶ **Definition 2.1** (Market exchange rate)**.** *Given a pool state* $\mathsf{Pool}(x, y)$*, the current market exchange rate is defined as*

$$r(x, y) = \frac{\partial \Phi / \partial x}{\partial \Phi / \partial y}(x, y).$$

Intuitively, it means that to buy an infinitesimally small dx *amount of* X*, we need to pay* $r(x, y) \cdot dx$ *units of* y*.*

Throughout this paper, whenever we say rate, it always means how much y one has to pay per unit of x rather than the other way around.

2.2 Arbitrage Resilience

Arbitrage resilience means that an arbitrager has no strategy such that it gets a net gain in one asset without any loss in the other.

▶ **Definition 2.2** (Arbitrage resilience)**.** *We say that a mechanism satisfies arbitrage resilience iff given any initial pool state, any input vector of orders, with probability* 1 *over the random coins of the mechanism, the following must hold: there does not exist a subset of orders whose joint outcomes result in* $\delta x \geq 0$ *net gain in* X *and* $\delta y \geq 0$ *net gain in* Y*, such that at least one of* δx *and* δy *is strictly greater than* 0*.*

The definition above is consistent with Ferreira and Parkes [30]'s notion of no risk-free return, although their scheme cannot guarantee no risk-free return (i.e., arbitrary resilience), whereas ours does.

▶ Remark 2.3. If a mechanism satisfies Definition 2.2, it means that it satisfies arbitrage resilience in a very strong sense: i.e., an arbitrager (e.g., block producer) cannot make risk-free profit *even when it can 1) fully control the block contents; 2) control the sequencing of orders within the block; 3) inject its own orders; and 4) drop others' orders.*

Indeed, the mechanism described later in this paper will satisfy arbitrage resilience even when the underlying consensus does not provide any sequencing fairness guarantees.

2.3 Plain Model

The plain model is meant to capture mainstream consensus protocol today where the contents of each individual block is determined by a single block producer responsible for proposing or mining that block.

Strategy space in the plain model. In the plain model, we assume that a strategy user or miner with intrinsic type (t, v, r, α) may engage in the following strategies:

- Post zero or multiple arbitrary orders which may or may not reflect its intrinsic type – this captures strategies that involve misreporting valuation and demand, as well as posting of fake orders;
- Censor honest users' orders – this captures a strategic miner's ability to exclude certain orders from the assembled block;
- Arbitrarily misrepresent its own auxiliary information field α, or even modify the α field of honest users' orders – meant to capture the ability of a miner to decide the sequencing of the transactions within a block, where the arrival-time and position information may be generically captured by the auxiliary information field α.
- Decide its strategy *after* having observed honest users' orders.

The coalition of a miner with 0 demand and a user with some positive demand as captured by its type $(t, v > 0, r, \alpha)$ can simply be viewed as a single strategic player with type $(t, v > 0, r, \alpha)$.

2.4 Weak Fair-Sequencing Model

We define a weak fair-sequencing model, meant to capture a new generation of consensus protocols that employ a decentralized sequencer and offers some form of sequencing fairness [2, 10, 34–36]. Such a decentralized sequencer will sequence the transactions based on their (approximate) arrival times. We stress that even in the weak fair-sequencing model, it is possible for a strategic user to observe a victim's order, post a dependent order, and have the dependent order race against and front-run the victim's order. Such a front-running attack can succeed especially when the strategic user's network is faster than the victim's. In particular, we stress that the weak fair-sequencing model is sequencing orders based on their *arrival times, not the time of the submission of these orders.*

Recall that a user's intrinsic *type* is of the form (t, v, r, α) where (t, v, r) denotes the user's true valuation and budget. In the weak fair-sequencing model, we will use the α field to encode the order's arrival time – a smaller α means that the user arrives earlier.

We shall assume that under honest strategy, a user's order should always be populated with the correct α whose value is determined by nature, and equal to the time at which the order is generated plus the user's network delay. A strategic user is allowed to delay the submission of its order.

Strategy space in the weak fair-sequencing model. We consider the following strategy space in the weak fair-sequencing model:

- A strategic user or miner with intrinsic type (t, v, r, α) is allowed to post zero or multiple bids of the form $(_, _, _, \alpha')$ as long as $\alpha' \geq \alpha$. This captures misreporting valuation and demand, posting fake orders, as well as delaying the posting of ones' orders.
- The strategic user or miner can decide its strategy *after* observing honest users' orders.

Compared to the plain model, the weak fair-sequencing model imposes some restrictions on the strategy space. Specifically, in the plain model, a strategic user or miner can arbitrarily modify the α field of its own order or even others' orders, and a strategic miner may censor honest users' orders. In the weak fair-sequencing model, a strategic user or miner can no longer under-report its α, cannot modify honest users' α, and cannot censor honest users' orders, because the sequencing of the transactions is determined by the underlying decentralized sequencer.

Importantly, despite these constraints on the strategy space, the weak fair-sequencing model still permits front-running-style attacks as mentioned earlier, and thus mechanism design remains non-trivial even under the slightly restricted strategy space.

2.5 Strategy Proof

Defining a user's preference among outcomes through a partial ordering. To define strategy proofness, we first need to define a ranking system that expresses a user's preference among different outcomes.

We can use a pair $(\delta x, \delta y)$ to denote the outcome, meaning that the user has a net gain of δx in X, and it has a net gain of δy in Y (where a net loss is captured as negative gain). Consider two outcomes $(\delta x_0, \delta y_0)$ and $(\delta x_1, \delta y_1)$, and suppose that the user's intrinsic type is $T = (\mathsf{Buy}(X), v, r, _)$. Naturally, for such a user, outcome $(\delta x_1, \delta y_1)$ is at least as good as $(\delta x_0, \delta y_0)$, henceforth denoted $(\delta x_0, \delta y_0) \preceq_T (\delta x_1, \delta y_1)$, if one of the following is true:

- $\delta x_0 \leq \delta x_1$, $\delta y_0 \leq \delta y_1$. In other words, relative to $(\delta x_0, \delta y_0)$, the user gains no less in either asset in the latter outcome $(\delta x_1, \delta y_1)$.
- $\delta x_0 \leq \delta x_1 \leq v$ (or $q \leq \delta x_1 \leq \delta x_0$), and $r(\delta x_1 - \delta x_0) \geq \delta y_0 - \delta y_1$. In other words, the latter outcome $(\delta x_1, \delta y_1)$ is closer to satisfying the demand v, and moreover, the user paid at most r *marginal* price for each *extra* unit of X in the latter outcome.

▶ Remark 2.4 (Why define a partial ordering rather than a real-valued utility). The reader may wonder why we do not define a real-valued utility which would have given a total ordering on all outcomes. The reason why we define only a partial ordering and allow some outcomes to be incomparable is because a strategic user (e.g., whose intrinsic demand is to buy up to v units of X) can act arbitrarily, which may cause its net gain δx in X to be either negative, or greater than the intrinsic demand v. We allow some of these outcomes to be incomparable. For example, suppose relative to $(\delta x_0, \delta y_0)$, the latter outcome $(\delta x_1, \delta y_1)$ buys some extra units at a margial price better than the specified rate r, but it overshoots the intrinsic demand, then these two outcomes are incomparable.

Finally, the case for other types including $\mathsf{Buy}(Y)$, $\mathsf{Sell}(X)$, and $\mathsf{Sell}(Y)$ types, a partial ordering can be symmetrically defined – we give the full definition of the partial ordering in the online full version. Moreover, the online full version also gives more explanations why we choose to define a partial ordering to rank the outcomes rather than a real-valued utility.

Definition of strategy proofness. Since in our paper, we consider deterministic mechanisms, we will define strategy proofness only for a deterministic mechanism. Note that the definition can easily be extended to randomized mechanism using suitable notions of stochastic dominance.

In the definition below, we use $HS(T)$ to denote the honest strategy of a user with intrinsic type T – for a direct-revelation mechanism, the honest strategy is simply to reveal the user's true type. Further, we use $\mathsf{out}^u(x_0, y_0, \mathbf{b})$ to denote the outcome of user u when the mechanism is executed over initial pool state $\mathsf{Pool}(x_0, y_0)$, and a vector of orders $\mathbf{b}$.

▶ **Definition 2.5** (Strategy proofness). *Given a deterministic swap mechanism, we say that it satisfies strategy proofness (w.r.t. some partial ordering relation $\preceq_T$), iff for any initial pool state $\mathsf{Pool}(x_0, y_0)$, for any vector of orders $\mathbf{b}_{-u}$ belonging to all other users except u, for any intrinsic type T of the strategic user u, for any possible strategic order vector $\mathbf{b}'$ of the user u, either $\mathsf{out}^u(x_0, y_0, \mathbf{b}_{-u}, HS(T)) \succeq_T \mathsf{out}^u(x_0, y_0, \mathbf{b}_{-u}, \mathbf{b}')$ or $\mathsf{out}^u(x_0, y_0, \mathbf{b}_{-u}, HS(T))$ and $\mathsf{out}^u(x_0, y_0, \mathbf{b}_{-u}, \mathbf{b}')$ are incomparable w.r.t. $\preceq_T$.*

More intuitively, the definition says that *no strategic play can result in a strictly better outcome than the honest strategy.*

Strategy proofness implies arbitrage resilience. The fact below shows that strategy proofness implies arbitrage resilience.

▶ **Fact 1.** *Suppose that the potential function Φ is increasing. We have that strategy proofness implies arbitrage resilience.*

Proof. We can prove the contra-positive, that is, a mechanism that is not arbitrage free cannot be strategy proof. Suppose that the mechanism is not arbitrage free. This means that there exists a list of orders $S = \{(t_i, v_i, r_i, \alpha_i)\}_i$ such that under honest execution, a subset of the orders $S' \subseteq S$ will enjoy $\delta x \geq 0$ and $\delta y \geq 0$, and at least one of the two is strictly positive. Now imagine that there is a world that consists of the orders $S \backslash S'$. In this case, a strategic user or miner with 0 demand can inject a set of fake orders S', and clearly this strategic behavior has positive gain, thus violating strategy proofness. Note that this strategy works even in the weak fair-sequencing model, as long as the strategic user's inherent arrival time α is no larger than the orders in S'. ◀

Since our plain-model mechanism gives an example that satisfies arbitrage resilience but not strategy proof, we conclude that strategy proofness is *strictly* stronger than arbitrage resilience. Specifically, at the beginning of Section 3.3, we explain why we cannot achieve strategy proof in the plain model. The strategies presented in this section also help to illustrate why the stronger notion of strategy proofness is more desirable.

3 Our Swap Mechanism

3.1 Construction

Our swap mechanism has two phases. In phase 1 (line 3c and 4c), the mechanism matches $\mathsf{Buy}(X)$ orders with $\mathsf{Buy}(Y)$ orders, and (partially) executes them at the initial rate r_0, such that at the end, there is no change to the initial pool state $\mathsf{Pool}(x_0, y_0)$. Phase 2 (line 3d and 4d) is a $\mathsf{Buy}(X)$-only phase, in which a sequence of $\mathsf{Buy}(X)$ orders (or $\mathsf{Buy}(Y)$ orders) are (partially) executed one by one. In phase 2, when the mechanism attempts to execute an order, it will execute as much as possible until either the demand has been fulfilled, or the new market price has reached the asking price. The details of the mechanism are described in Figure 1.

Our swap mechanism

Input: A current pool state $\mathsf{Pool}(x_0, y_0)$, and a vector of orders $\mathbf{b}$. Since the mechanism does not make use of the auxiliary information field, we simply assume each order is a tuple of the form (t, v, r).

Mechanism:

1. Let $r_0 := r(x_0, y_0)$ be the initial exchange rate. Ignore all $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$ orders whose specified rate $r < r_0$, and ignore all $\mathsf{Buy}(Y)/\mathsf{Sell}(X)$ orders whose specified rate $r > r_0$. Let $\mathbf{b}'$ be the remaining orders.
2. Let $\sigma = \sum_{(t,v,r)\in\mathbf{b}'} \beta(t, v, r)$ where $\beta(t, v, r) = \begin{cases} v & \text{if } t = \mathsf{Buy}(X) \\ -v & \text{if } t = \mathsf{Sell}(X) \\ -v/r_0 & \text{if } t = \mathsf{Buy}(Y) \\ v/r_0 & \text{if } t = \mathsf{Sell}(Y) \end{cases}$
   We call $\sigma \geq 0$ the $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-dominant case, and $\sigma < 0$ the $\mathsf{Buy}(Y)/\mathsf{Sell}(X)$-dominant case.
3. The $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-dominant case (if $\sigma \geq 0$):
   a. Sort $\mathbf{b}'$ such that all the $\mathsf{Buy}(Y)/\mathsf{Sell}(X)$ orders appear in front of the $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$ orders. Write the resulting list of orders as $\{(t_i, v_i, r_i)\}_{i\in[n']}$.
   b. Henceforth we assume that there exists an index $j \in [n']$ such that $\sum_{i=1}^{j} \beta(t_i, v_i, r_i) = 0$. If not, we can find the smallest index $j \in [n']$ such that $\sum_{i=1}^{j} \beta(t_i, v_i, r_i) > 0$, and split the j-th order into two orders $(t_j, v_{j,0}, r_j)$ and $(t_j, v_{j,1}, r_j)$, resulting in a new list with $n' + 1$ orders, such that $v_{j,0} + v_{j,1} = v_j$, and moreover, index j of the new list satisfies this condition.
   c. Phase 1: Fully execute the first j orders at the initial rate r_0.
   d. Phase 2: For each $i \geq j + 1$ in sequence, fulfill as much of the i-th remaining order as possible, that is, pick the largest $v \leq v_i$ such that subject to the constant-function market maker Φ, the new market rate $r \leq r_i$ if v units are to be executed; execute v units of the i-th order.
4. The $\mathsf{Buy}(Y)/\mathsf{Sell}(X)$-dominant case is symmetric (if $\sigma < 0$):
   a. Sort $\mathbf{b}'$ such that all the $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$ orders appear after the $\mathsf{Buy}(Y)/\mathsf{Sell}(X)$ orders. Write the resulting list of orders as $\{(t_i, v_i, r_i)\}_{i\in[n']}$.
   b. Henceforth we assume that there exists an index $j \in [n']$ such that $\sum_{i=1}^{j} \beta(t_i, v_i, r_i) = 0$. If not, we can find the smallest index $j \in [n']$ such that $-\sum_{i=1}^{j} \beta(t_i, v_i, r_i) > 0$, and split the j-th order into two orders $(t_j, v_{j,0}, r_j)$ and $(t_j, v_{j,1}, r_j)$, resulting in a new list with $n'+1$ orders, such that $v_{j,0}+v_{j,1} = v_j$, and moreover, index j of the new list satisfies this condition.
   c. Phase 1: Fully execute the first j orders at the initial rate r_0.
   d. Phase 2: For $i \geq j + 1$ in sequence, fulfill as much of the i-th remaining order as possible, that is, pick the largest $v \leq v_i$ such that subject to the constant-function market maker Φ, the new market exchange rate $r \geq r_i$ if v units are to be executed; execute v units of the i-th order.

Figure 1 Our swap mechanism.

In the sorting steps (Lines 3a and 4a[6]), we may need to break ties among identical orders. We suggest two approaches for tie-breaking:

- In the presence of a centralized block proposer (i.e., when the arbitrager can be in full control of block creation), we suggest *random tie-breaking*. Note that the random tie-breaking is enforced by the mechanism (i.e., the smart contract). The random coins needed should come from a trusted source at the consensus layer, e.g., through the use of a fair coin toss protocol [18, 27]. This ensures that the miner or block producer cannot auction off favorable positions in the block to arbitragers or users.
  Note that in the plain model, we aim to achieve only arbitrage-free and not strategy proofness. So although it may seem like a user can adopt a Sybil strategy and submit fake bids to game the random tie-breaking, such strategies do not actually violate the arbitrage-free property..
- If the block proposal process is decentralized and ensures sequencing fairness, we suggest tie-breaking according to the time of arrival. Section 3.3 and Section 4 show that this approach allows us to achieve strategy proofness in the weak fair-sequencing model.

3.2 Proof of Arbitrage Resilience

We now prove that our swap mechanism satisfies arbitrage resilience regardless of how ties are broken in Lines 3a and 4a. As mentioned earlier, the arbitrage resilience property holds even when the arbitrager is in full control of block creation, can drop or inject orders, and can control the sequencing of orders within the block.

▶ **Theorem 3.1** (Arbitrage resilience)**.** *The swap mechanism in Figure 1 satisfies arbitrage resilience. In particular, this holds no matter how ties are broken in Lines 3a and 4a.*

Proof. We prove it for the $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-dominant case, since the $\mathsf{Buy}(Y)/\mathsf{Sell}(X)$-dominant case is symmetric. The mechanism essentially does the following. In phase 1, it (partially) executes a set of orders all at the initial rate r_0, such that there is no change to the initial pool state $\mathsf{Pool}(x_0, y_0)$. In phase 2, it executes only $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$ orders. Due to increasing marginal cost (Fact 2), in Phase 2, all the (partially) executed $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$ orders enjoy a rate that is at least r_0. Therefore, all the (partially) executed $\mathsf{Buy}(Y)/\mathsf{Sell}(X)$ orders enjoy a rate of r_0, and all the (partially) executed $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$ orders enjoy a rate that is r_0 or greater. Thus, it cannot be the case that there is a net gain in one asset without any loss in the other. ◀

3.3 A Refinement of the Mechanism for the Weak Fair-Sequencing Model

Why the mechanism is NOT strategy-proof in the plain model. As mentioned, our swap mechanism in Figure 1 does not fully specify how to break ties in the sorting steps of in Lines 3a and 4a. Moreover, our arbitrage resilience property does not care how the tie-breaking is done.

However, if we are not careful about the tie-breaking the resulting mechanism may not satisfy strategy proofness. For example, imagine that the tie-breaking is based on the α field of the order, which encodes the time at which the order is submitted. In this case, a strategic miner or user can simply make its own order have a small α to enjoy a better price, rather than truthfully reporting α.

[6] Note that the sorted order produced by (3a) will be consumed in the subsequent steps (3b), (3c), and (3d); similarly, the sorted order of (4a) are consumed in (4b), (4c), and (4d).

Refinement in the fair-sequencing model. In the weak fair-sequencing model, the arrival order α is decided by an underlying consensus protocol that provides some form of sequencing fairness. In other words, the consensus protocol itself records the approximate time at which each order is first seen. In practice, this arrival time is dependent on when the user submits the its order to the network, and its network delay. A sequencing-fair consensus protocol cannot prevent a user (or miner) from delaying the submission of its order. It also cannot prevent a strategic user with an extremely fast network from front-running a victim's order. Specifically, the strategic user can still observe what the victim submits, and then immediately submits a dependent order. If the strategic user's network is faster than the victim's, the strategic order may have an earlier arrival time than the victim's order! However, sequencing fairness from the consensus does tie the hands of strategic players in the following weak manner (hence the name "weak sequencing fair"): if a strategic user's order is generated at time α, it cannot pretend that the order was generated at $\alpha' < \alpha$ even if it has an extremely fast network with a delay of 0.

We can refine the mechanism in Figure 1 as follows to achieve strategy proofness in the weak fair-sequencing model (see also Section 4):

> For tie-breaking in the sorting steps (Lines 3a and 4a), we now require that the sorting be stable, that is, in the sorted outcome, the relative ordering among all identical $\mathsf{Buy}(Y)$ orders must respect the arrival order encoded in the α field; the same holds for all $\mathsf{Buy}(X)$ orders.

We will assume that a user's honest strategy is to honestly report its type including the α field, that is, $HS(t, v, r, \alpha)$ simply outputs a single order (t, v, r, α).

▶ **Theorem 3.2** (Strategy proofness in the weak fair-sequencing model)**.** *Suppose Φ is concave, increasing, and differentiable. In the weak fair-sequencing model, the above refined swap mechanism is strategy proof (see Definition 2.5).*

The proof of Theorem 3.2 is provided in Section 4.

4 Proof of Strategy Proofness in the Weak Fair-Sequencing Model

4.1 Useful Facts

We first prove a few useful facts.

▶ **Fact 2** (Increasing marginal cost)**.** *Suppose that Φ is increasing, differentiable, and concave. Given two pool states $\mathsf{Pool}(x, y), \mathsf{Pool}(x', y')$ such that $\Phi(x, y) = \Phi(x', y')$, and $x' \leq x$, it must be that $r(x, y) \leq r(x', y')$. In other words, the price of X goes up if the pool has less supply of X.*

Proof. Suppose $\Phi(x, y) = C$. Since Φ is increasing, by Lemma B.1 of [30], the potential function Φ defines a bijective decreasing function $h(\cdot)$ such that $\Phi(z, h(z)) = C$. Moreover, since Φ is concave, the induced function $h(\cdot)$ is convex (Lemma B.2 of [30]). Observe that Φ is differentiable, so $r(x, y) = -h'(x)$. Therefore, $r(x, y) \leq r(x', y')$ for $x' \leq x$ by the convexity of h. ◀

▶ **Fact 3** (No free lunch)**.** *Suppose $(\delta x, \delta y)$ is the outcome resulting from the execution of a single order in the swap mechanism in Figure 1. If at least one of δx and δy is non-zero, then $\delta x \cdot \delta y < 0$.*

4.2 Proof

We now prove Theorem 3.2. Our mechanism is deterministic, so a deterministic strategy yields a deterministic outcome. Recall that a user has a partial ordering among the outcomes. Henceforth, if two outcomes satisfy $(\delta x_1, \delta y_1) \succeq (\delta x_2, \delta y_2)$, we say that $(\delta x_1, \delta y_1)$ is *at least as good as* $(\delta x_2, \delta y_2)$.

Suppose that the strategic user u's type is $(t^*, v^*, r^*, \alpha^*)$, its strategic order vector is $\mathbf{b}_u := \{(t_j, v_j, r_j, \alpha_j)\}_j$, the initial state is $\mathsf{Pool}(x_0, y_0)$, and the order vector from all other users is $\mathbf{b}_{-u}$.

▶ **Fact 4.** *Given the initial state* $\mathsf{Pool}(x_0, y_0)$*, the order vector from other users* $\mathbf{b}_{-u}$*, and a strategic order vector* $\mathbf{b}_u := \{(t_j, v_j, r_j, \alpha_j)\}_j$*, there exists an alternative strategic vector* $\mathbf{b}'_u$ *which contains only* $\mathsf{Buy}(X)$ *and* $\mathsf{Sell}(X)$*-type (or* $\mathsf{Buy}(Y)$ *and* $\mathsf{Sell}(Y)$*-type) order, such that the outcome of* $\mathbf{b}'_u$ *is the same as the outcome of* $\mathbf{b}_u$*.*

Proof. Since the mechanism is deterministic, given the initial state $\mathsf{Pool}(x_0, y_0)$, order vector from all other users $\mathbf{b}_{-u}$, and the strategic order vector $\mathbf{b}_u$, one can compute the whole order execution trace of the mechanism. For an order $b = (\mathsf{Sell}(Y), v_j, r_j, \alpha_j)$ in $\mathbf{b}_u$, if it is not executed, then it can be replaced with a $(\mathsf{Buy}(X), 0, r_j, \alpha_j)$. If it is partially fulfilled, let $(x_{\text{start}}, x_{\text{end}})$ denote the amount of asset X in the pool right before and right after b is executed, respectively. Let r_{end} denote the market exchange rate at x_{end}. Then b can be replaced with an order $(\mathsf{Buy}(X), x_{\text{start}} - x_{\text{end}}, r_{\text{end}}, \alpha_j)$, without changing the outcome. Similarly, we can replace a $\mathsf{Buy}(Y)$-type order with a $\mathsf{Sell}(X)$-type order without changing the outcome. The fact thus follows. ◀

▶ **Lemma 4.1.** *Suppose* Φ *is concave, increasing, and differentiable. For any strategic order vector* $\mathbf{b}_u$*, there exists a single order* b'_u *such that 1)* b'_u *results in an outcome at least as good as* $\mathbf{b}_u$*; 2) the arrival time used in* b'_u *is no earlier than the earliest arrival time in* $\mathbf{b}_u$*; and 3) either* $b'_u = (_, 0, _, _)$ *or* b'_u *would be completely fulfilled under* $\mathsf{Pool}(x_0, y_0)$ *and* $\mathbf{b}_{-u}$*.*

Proof. To prove this lemma, we first show that we can coalesce all the $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-type orders into one, and all the $\mathsf{Sell}(X)/\mathsf{Buy}(Y)$-type orders into one, as stated in the following claim.

▷ Claim 4.2. Suppose Φ is concave, increasing, and differentiable. For any strategic order vector $\mathbf{b}_u$, if it contains both $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-type and $\mathsf{Sell}(X)/\mathsf{Buy}(Y)$-type orders, there exists another order vector $\mathbf{b}'_u$ which contains a single $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-type order and a single $\mathsf{Sell}(X)/\mathsf{Buy}(Y)$-type order such that 1) $\mathbf{b}'_u$ results in an outcome at least as good as $\mathbf{b}_u$; 2) the arrival times used in $\mathbf{b'}_u$ are no earlier than the earliest arrival time in $\mathbf{b}_u$; and 3) an order in $\mathbf{b}'_u$ is either of the form $(_, 0, _, _)$ or it would be completely fulfilled under $\mathsf{Pool}(x_0, y_0)$ and $\mathbf{b}_{-u}$.

Proof. According to Fact 4, we can assume that $\mathbf{b}_u$ contains only $\mathsf{Buy}(X)$ and $\mathsf{Sell}(X)$-type orders. Let $\mathbf{b}_{\mathsf{Sell}}$ and $\mathbf{b}_{\mathsf{Buy}}$ denote the vector of $\mathsf{Sell}(X)$-type and $\mathsf{Buy}(X)$-type orders in $\mathbf{b}_u$, respectively. Without loss of generality, assume that given $\mathbf{b}_{-u}$ and $\mathbf{b}_u$, we have the $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-dominant case, (the $\mathsf{Buy}(Y)/\mathsf{Sell}(X)$-dominant case is symmetric).

For the $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-dominant case, all $\mathsf{Sell}(X)$-type orders in $\mathbf{b}_{\mathsf{Sell}}$ will be executed at the initial exchange rate r_0. Therefore, consider an order $b = (\mathsf{Sell}(X), v, r, \alpha)$, where v denotes the total units of orders in $\mathbf{b}_{\text{Sell}}$, r is the minimum asking rate in $\mathbf{b}_{\text{Sell}}$, and α is the earliest arrival time in $\mathbf{b}_u$. Then a strategic order vector $(\mathbf{b}_{\text{Buy}}, b)$ results in the same outcome as $\mathbf{b}_u$.

Now consider an order $b' = (\mathsf{Buy}(X), v', r', \alpha')$, where v' is the total units of order executed in $\mathbf{b}_{\text{Buy}}$, r' is the maximum asking rate in $\mathbf{b}_{\text{Buy}}$, and α' is the earliest arrival time of the order in $\mathbf{b}_{\text{Buy}}$ that is partially fulfilled. Let $\mathbf{b}'_u = (b', b)$. Compared to the units executed in $\mathbf{b}_{\text{Buy}}$, the units executed in b' have an earlier or the same arrival time. In addition, the asking rate r' in b' is larger than or equal to that in $\mathbf{b}_{\text{Buy}}$. Therefore, all units in b' will be executed. Moreover, by the increasing marginal cost (Fact 2), the exchange rate for b' is no more than the average exchange rate for all orders in $\mathbf{b}_{\text{Buy}}$. This means that $\mathbf{b}'_u$ results in an outcome that is at least as good as $(\mathbf{b}_{\text{Buy}}, b)$ according to the partial ordering. By the transitivity, $\mathbf{b}'_u$ results in an outcome that is at least as good as $\mathbf{b}_u$. ◁

Next, we show that for any order vector $\mathbf{b}_u$ that contains a single $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-type order and a single $\mathsf{Sell}(X)/\mathsf{Buy}(Y)$-type order, we can remove the part that "cancels off", and substitute it with a single order. This is formally stated in the following claim.

▷ Claim 4.3. Suppose Φ is concave, increasing, and differentiable. For any order vector $\mathbf{b}_u$ that contains a single $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-type order and a single $\mathsf{Sell}(X)/\mathsf{Buy}(Y)$-type, there exists a single order b'_u such that 1) b'_u results in an outcome at least as good as $\mathbf{b}_u$; 2) the arrival time in b'_u is no earlier than the earliest arrival time in $\mathbf{b}_u$; and 3) b'_u is either of the form $(_, 0, _, _)$ or it would be completely fulfilled under $\mathsf{Pool}(x_0, y_0)$ and $\mathbf{b}_{-u}$.

Proof. Because of Fact 4, we assume that the strategic order vector $\mathbf{b}_u$ contains a single $\mathsf{Buy}(X)$-type order $(\mathsf{Buy}(X), v_b, r_b, \alpha_b)$ and a single $\mathsf{Sell}(X)$-type order $(\mathsf{Sell}(X), v_s, r_s, \alpha_s)$. By our assumption, both orders are fully executed. If either v_b or v_s is zero, the result follows trivially; henceforth, we assume that both are non-zero. Similarly, if either $r_b < r_0$ or $r_s > r_0$, then the result also follows trivially. henceforth, $r_b \geq r_0$ and $r_s \leq r_0$.

We prove it assuming the $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-dominant case case under the strategic order vector $\mathbf{b}_u$, since the argument for the $\mathsf{Buy}(Y)/\mathsf{Sell}(X)$-dominant case is symmetric. For the $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$-dominant case, if it were possible to execute all orders at r_0, then there is more demand in terms of only $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$ than $\mathsf{Sell}(X)/\mathsf{Buy}(Y)$. In this case, Phase 1 executes all $\mathsf{Sell}(X)/\mathsf{Buy}(Y)$ orders at r_0, and Phase 2 executes only $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$ orders.

Case $v_s \geq v_b$. Under the original $\mathbf{b}_u$, the user would sell v_s units of X at r_0 and would buy v_b units of X at a rate of r_0 or greater. Now, suppose we replace $\mathbf{b}_u$ with a single order $b'_u = (\mathsf{Sell}(X), v_s - v_b, r_s, \alpha_s)$. Under b'_u, it is still the case that all $\mathsf{Sell}(X)$ orders are completely executed at r_0.

Hence, we can decompose the original $\mathbf{b}_u$ equivalently into the following steps: (i) first execute b'_u; (ii) sell v_b units at rate r_0; (iii) buy back v_b units at rate r_0 or greater. Since steps (ii) and (iii) together will incur a non-negative loss in Y (but create no change in X), user u's outcome under b'_u is at least as good as $\mathbf{b}_u$.

Case $v_s < v_b$. Consider the original $\mathbf{b}_u$ which consists of $(\mathsf{Sell}(X), v_s, r_s, \alpha_s)$ and $(\mathsf{Buy}(X), v_b, r_b, \alpha_b)$. We will analyze what happens when we replace these two orders with a single order $b'_u = (\mathsf{Buy}(X), v_b - v_s, r_b, \alpha_b)$. Suppose in Phase 1 of the execution with $\mathbf{b}_u$, user u sells v_s units of X and buys $v' \leq v_b$ units of X at rate r_0. We separate the rest of the proof into two cases.

1. **Case $v' \geq v_s$.** For the execution with $\mathbf{b}_u$: In Phase 1, the net effect is to buy $v' - v_s$ units of X at rate r_0. In Phase 2, the user u buys the remaining $v_b - v'$ units starting at rate r_0.

   The execution with b'_u can be viewed as follows: In Phase 1', $v' - v_s$ units of $\mathsf{Buy}(X)$ will be executed at rate r_0. Then in Phase 2', the mechanism executes the rest $v_b - v'$ units in b'_u starting at rate r_0.

Hence, the two scenarios are equivalent, and the two outcomes are the same.

2. **Case $v' < v_s$.** We will view the execution of the orders $(\mathsf{Sell}(X), v_s, r_s, \alpha_s)$ and $(\mathsf{Buy}(X), v_b, r_b, \alpha_b)$ as follows.
   - *Phase 1:* User u sells v_s units of X and buy v' units of X at a rate of r_0, and it gains $(v_s - v') \cdot r_0$ units of Y in return.
   - *Phase 2a:* Some non-negative amount of $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$ orders from other users are executed at a starting market rate of r_0, let $v_{\text{other}} \geq 0$ be the units of X purchased. Note that if $v' > 0$, then $v_{\text{other}} = 0$.
     At the end of this phase, the market rate $r_1 \geq r_0$ by increasing marginal cost.
   - *Phase 2b:* Starting at rate $r_1 \geq r_0$, user u buys $v_s - v'$ units of X, which changes the market rate to $r_2 \geq r_1$.
   - *Phase 2c:* Starting at rate r_2, the user buys $(v_b - v_s)$ units of X, changing the market rate to r_3.

   The new execution involving b'_u can be viewed as the following:
   - *Phase 1'*: The $\mathsf{Buy}(X)/\mathsf{Sell}(Y)$ orders of other users executed in the original Phase 1 cannot all be executed in the new Phase 1. In particular, the last $v_s - v'$ units of X cannot be fulfilled in Phase 1', and will be pushed to Phase 2a' – henceforth, we call this portion the residual.
   - *Phase 2a'*: The mechanism attempts to execute the residual from Phase 1' at a starting rate of r_0. The amount fulfilled must be at most $v_s - v'$.
   - *Phase 2b'*: The mechanism attempts to execute the (partial) orders originally considered in Phase 2a, at a starting price that is at least r_0. At most v_{other} units of X can be fulfilled. The ending market rate must be at least r_1.
   - *Phase 2c'*: The mechanism attempts to execute b'_u. Observe that the total units of X fulfilled in the original Phase 2a, 2b, and 2c is $v_{\text{other}} + v_b - v'$, and the total units of X fulfilled in the new Phase 2a', 2b', and 2c' is at most $(v_s - v') + v_{\text{other}} + v_b - v_s = v_{\text{other}} + v_b - v'$. Therefore, it must be that all of b'_u can be fulfilled and the ending market rate is at most $r_3 \leq r_b$.

   Henceforth, we use the notation $\mathsf{Pay}(\text{Phase }*)$ to denote user u's payment in terms of Y in some phase. When the pay is negative, it means a gain in Y. Observe that

$$\mathsf{Pay}(\text{Phase 1}) + \mathsf{Pay}(\text{Phase 2b}) \geq 0, \quad \mathsf{Pay}(\text{Phase 2c}) \geq \mathsf{Pay}(\text{Phase 2c'}).$$

   Therefore,

$$\mathsf{Pay}(\text{Phase 1}) + \mathsf{Pay}(\text{Phase 2b}) + \mathsf{Pay}(\text{Phase 2c}) \geq \mathsf{Pay}(\text{Phase 2c'}).$$

   Observe that in the above, the left-hand side represents u's total payment in Y under the original $\mathbf{b}'_u$, and the right-hand side represents u's total payment in Y under the new b'_u. ◁

Lemma 4.1 follows by combining Claim 4.2 and Claim 4.3. ◀

▶ **Lemma 4.4.** *Suppose Φ is concave, increasing, and differentiable. Given any initial state $\mathsf{Pool}(x_0, y_0)$, any order vector $\mathbf{b}_{-u}$, any true arrival time α^* of user u, given an order b'_u with an arrival time later than α^*, there exists another order b_u with an arrival time exactly α^*, and moreover, user u's outcome under b_u is at least as good as its outcome under b'_u, and b_u is either completely executed or of the form $(_, 0, _, _)$.*

Proof. Due to Fact 4, we may assume that b_u is either a $\mathsf{Buy}(X)$ or $\mathsf{Sell}(X)$ order. We prove it for a $\mathsf{Buy}(X)$ order, since the case for a $\mathsf{Sell}(X)$ order is symmetric. Let v be the amount of X bought by b'_u. We shall assume that $v > 0$ since the case $v = 0$ is trivial. Consider an order $b_u = (\mathsf{Buy}(X), v, +\infty, \alpha^*)$. Clearly, b_u will buy v units of X.

We consider the following cases:

- b_u buys $v_0 > 0$ units at r_0 in Phase 1, and then buys $v_1 \geq 0$ units in Phase 2 at a starting rate of r_0 and an ending rate of $r_1 \geq r_0$. In this case, by delaying the arrival time, b'_u can buy at most $v'_0 \leq v_0$ units in Phase 1 at r_0, and it needs to buy $v - v'_0 \geq v_1$ in Phase 2. Therefore, for b'_u, the starting rate in Phase 2 is r_0, and the ending rate must be at least r_1. Therefore, the average price paid per unit in b_u is no worse than the average price paid per unit in b'_u.
- b_u buys all v units in Phase 2. Since b'_u delayed the arrival, the order can be considered no earlier than b_u. Thus, before the mechanism tries to execute b'_u at least as many units of X will have been bought (by all users) as when the mechanism tries to execute b_u. This means b'_u will have an average price no better than b_u. ◀

Due to Lemmas 4.1 and 4.4, it suffices to consider strategies that submit a single order, declare the true arrival time α^*, and moreover, either the order has a 0 amount or it will be completely executed under $\mathsf{Pool}(x_0, y_0)$ and $\mathbf{b}_{-u}$ – henceforth, we call such strategies as *admissible, single-order* strategies. We can complete the proof of Theorem 3.2 by showing the following lemma.

▶ **Lemma 4.5.** *Suppose Φ is concave, increasing, and differentiable. For any admissible and single-order strategy S, the honest strategy results in an outcome that is at least as good as or incomparable to strategy S.*

Proof. Below we complete the proof of Lemma 4.5.

We prove it for the case when user u's type is either $(\mathsf{Buy}(X), v^*, r^*, \alpha^*)$ or $(\mathsf{Sell}(X), v^*, r^*, \alpha^*)$. The case for $\mathsf{Buy}(Y)/\mathsf{Sell}(Y)$ is symmetric. Given two outcomes $\mathsf{out}_0 = (\delta x_0, \delta y_0)$ and $\mathsf{out}_1 = (\delta x_1, \delta y_1)$ and a true demand δx^* for X, we say that they are *on the same side* of the goal δx^* iff $(\delta x_0 - \delta x^*) \cdot (\delta x_1 - \delta x^*) \geq 0$. We say that out_0 is *at least as close as* out_1 towards the goal iff $|\delta x_0 - \delta x^*| \leq |\delta x_1 - \delta x^*|$, and we say that out_0 is *closer* to the goal than out_1 iff $|\delta x_0 - \delta x^*| < |\delta x_1 - \delta x^*|$.

Our natural partial ordering relation implies the following:

R1. Suppose $(\delta x_0, \delta y_0)$ and $(\delta x_1, \delta y_1)$ are on the same side of the goal, and δx_0 is at least as close as δx_1 towards the goal. Moreover, if $(\delta x_0 - \delta x_1) \cdot (\delta y_1 - \delta y_0) < 0$, then $(\delta x_1, \delta y_1) \not\geq (\delta x_0, \delta y_0)$.

R2. If δx_0 and δx_1 are on the same side of the goal, δx_0 is closer than δx_1 to the goal, and moreover, $\delta y_1 - \delta y_0 > r^* \cdot (\delta x_0 - \delta x_1)$, then $(\delta x_0, \delta y_0) \not\geq (\delta x_1, \delta y_1)$.

R3. If δx_0 and δx_1 are on different sides of the goal, and moreover, $\delta y_1 - \delta y_0 \geq r^* \cdot (\delta x_0 - \delta x_1)$, then $(\delta x_0, \delta y_0) \not\geq (\delta x_1, \delta y_1)$.

Due to Fact 4, we may assume that the strategic order must be of the type $\mathsf{Buy}(X)$ or $\mathsf{Sell}(X)$. Further, as shown in the following fact, we can in fact assume that the strategic order adopts the true time of arrival α^*, i.e., declaring a later time never helps.

Henceforth, let $(\delta x, \delta y)$ and $(\delta x', \delta y')$ denote the honest and strategic outcomes, respectively.

Case 1. Either user u has a true demand of 0 units, or the strategic order is opposite the direction of its true demand, i.e., if its type is $\mathsf{Buy}(X)$, it submits a single $\mathsf{Sell}(X)$ order; or vice versa. It must be that $\delta x \cdot \delta x' \leq 0$. Further, the honest outcome and strategic outcome

must be on the same side of the true demand, and the honest outcome is at least as close to the goal as the strategic outcome. Our mechanism guarantees that either (i) $\delta x = \delta x' = 0$, or (ii) at least one of $\delta x'$ and $\delta x'$ is non-zero. In case (i), $\delta y = \delta y' = 0$, and the the two outcomes are the same.

In case (ii), because of no free lunch (Fact 3), at least one of the inequalities $\delta x \cdot \delta y \leq 0$ and $\delta x' \cdot \delta y' \leq 0$ must be strict; moreover, when an equality holds, both arguments of the product must be zero. Because $\delta x \cdot \delta x' \leq 0$, this implies that $(\delta x - \delta x')(\delta y - \delta y') < 0$; by the above rule R1, the honest outcome is at least as good as or incomparable the strategic one.

Case 2. The user u has a non-zero amount of true demand, and moreover, the strategic order is in the same direction of the true demand. We consider the following cases.

- *Case 2a:* $\delta x = \delta x'$. By admissibility, the strategic order declares the same arrival time as the honest one; hence, if the orders from both strategies get executed for a non-zero amount, both execution will start at the same market exchange rate. Hence, it must be the case that $(\delta x, \delta y) = (\delta x', \delta y')$.
  Henceforth, we assume that $\delta x \neq \delta x'$.
- *Case 2b:* The honest outcome and the strategic outcome are on the same side of the goal, and the honest outcome closer to the goal than the strategic outcome; this case includes the scenario that the honest outcome is exactly at the goal. Since we can assume that the strategic order has the same arrival time as the honest order, the difference of $|\delta x - \delta x'|$ units are traded at a marginal price at least as good as r in the honest outcome. Due to the third rule of the natural partial ordering, the honest outcome is at least as good as the stategic one.
- *Case 2c:* The honest outcome and the strategic outcome are on the same side of the goal, and the strategic outcome is closer to the goal than the honest outcome, i.e., $|\delta x| < |\delta x'| \leq |\delta x^*| = v^*$.
  This means that the honest outcome has not reached the goal of user u. Under the honest strategy, after user u's order has been executed (or attempted to be executed), the state of the market is such that if a further non-zero portion of the order is executed, this portion will incur an average rate of strictly worse than r^*. In the case of $\mathsf{Buy}(X)$, this is strictly larger than r^*; in the case of $\mathsf{Sell}(X)$, this is strictly less than r^*.
  Since the strategic order declares the same arrival time as the honest order, the difference of $|\delta x' - \delta x| > 0$ units must be traded at an average rate strictly worse than than r^* in the strategic outcome. By rule R2, the strategic outcome is not at least as good as the honest outcome. However, because of no free lunch, the two outcomes are actually incomparable.
- *Case 2d:* The honest outcome and the strategic outcome are on different sides of the goal, i.e., $|\delta x| < |\delta x^*| = v^* < |\delta x'|$.
  In this case, $\delta x \neq 0$. Similar to case 2c, under the honest strategy, after user u's order has been executed, the state of the market is such that if a further non-zero portion of the order is executed, this portion will incur an average rate of strictly worse than r^*.
  Because the strategic order declares the same arrival time as the honest one, the difference of $|\delta x' - \delta x|$ units must be traded at an average rate of strictly worse than than r^* in the strategic outcome. By rule R3, the strategic outcome cannot be at least as good as the honest outcome. ◀

5 Conclusion

In this paper, we propose new models for studying mechanism design for DeFi applications. Unlike the prior work of Ferreira and Parkes [30] and others' [39] which assume that the mechanism on the blockchain must be a first-in-first-out mechanism, we allow the the

mechanism designer to specify the mechanism running on the blockchain. This allows us to circumvent the strong impossibility results of Ferreira and Parkes [30]. Depending on assumption on the underlying blockchain, we consider two possible strategies spaces. If the underlying blockchain does not enjoy sequencing fairness, we assume that the strategic user (or miner) can post orders after observing honest users' orders, insert fake orders, censor honest users' orders, and control the sequencing of the orders in the block. If the underlying blockchain enjoys sequencing fairness, we assume that the strategic user can do all of the above; however, it cannot censor honest users' orders, nor can it under-report its arrival time.

We design a novel mechanism that achieves arbitrage resilience (which was deemed impossible under Ferreira and Parkes [30]'s model), and additionally achieves strategy proofness if the underlying consensus offers sequencing fairness.

Our paper raises many interesting directions for future work. For example, can we achieve strategy proofness without relying on the sequencing fairness assumption? Can we extend the results to multi-asset swaps? Can we optimize social welfare and revenue under strategy proofness? Another interesting direction is whether we can achieve strategy proofness under an all-or-nothing fulfillment model, that is, any order is either completely fulfilled or not executed at all.

References

1 https://cow.fi/cow-protocol.

2 The espresso sequencer. https://hackmd.io/@EspressoSystems/EspressoSequencer.

3 https://www.metis.io/decentralized-sequencer.

4 https://www.zeeve.io/blog/why-would-a-layer2-decentralize-its-sequencer/.

5 https://ethglobal.com/showcase/decentralized-sequencers-for-optimism-rollup-1o98i.

6 https://morph.ghost.io/introduction-to-decentralized-sequencer-network/.

7 https://docs.morphl2.io/docs/how-morph-works/decentralized-sequencers/morph-decentralized-sequencer-network/.

8 https://blog.kroma.network/decentralized-sequencers-d2a4aeaf1084?gi=ee7408176d62.

9 https://ethglobal.com/talks/ordering-so-fair-it-ain-t-fair-ordering-a5b60.

10 Challenging periods reimagined: The key role of sequencer decentralization. https://ethresear.ch/t/challenging-periods-reimagined-the-key-role-of-sequencer-decentralization/15110.

11 Guillermo Angeris, Alex Evans, and Tarun Chitra. A note on bundle profit maximization, 2021.

12 Kushal Babel, Philip Daian, Mahimna Kelkar, and Ari Juels. Clockwork finance: Automated analysis of economic security in smart contracts. In *IEEE Symposium on Security and Privacy*, 2023.

13 Maryam Bahrani, Pranav Garimidi, and Tim Roughgarden. Transaction fee mechanism design in a post-mev world. In *6th Conference on Advances in Financial Technologies, AFT 2024, September 23-25, 2024, Vienna, Austria*, volume 316 of *LIPIcs*, pages 29:1–29:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.AFT.2024.29.

14 Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. A theory of automated market makers in defi. *CoRR*, 2021. arXiv:2102.11350.

15 Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch Lafuente. Maximizing extractable value from automated market makers. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 3–19, Berlin, Heidelberg, 2022. doi:10.1007/978-3-031-18283-9_1.

16 Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch Lafuente. Maximizing extractable value from automated market makers. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, 2022. doi:10.1007/978-3-031-18283-9_1.

17 Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, Paper 2022/898, 2022. URL: https://eprint.iacr.org/2022/898.

18 Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. Randpiper reconfiguration-friendly random beacons with quadratic communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 3502–3524, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3460120.3484574.

19 R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

20 Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 2000.

21 Andrea Canidio and Robin Fritsch. Batching trades on automated market makers. In *AFT*, 2023.

22 Andrea Canidio and Robin Fritsch. Arbitrageurs' profits, lvr, and sandwich attacks: batch trading as an amm design response. *CoRR*, 2024. doi:10.48550/arXiv.2307.02074.

23 Andrea Canidio and Felix Henneke. Fair combinatorial auction for blockchain trade intents: Being fair without knowing what is fair, 2024. arXiv:2408.12225.

24 Miles Carlsten, Harry Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

25 T-H. Hubert Chan, Ke Wu, and Elaine Shi. Mechanism design for automated market makers. https://arxiv.org/abs/2402.09357.

26 Hao Chung and Elaine Shi. Foundations of transaction fee mechanism design. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3856–3899. SIAM, 2023. doi:10.1137/1.9781611977554.CH150.

27 Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. SPURT: Scalable distributed randomness beacon with transparent setup. In *IEEE Symposium on Security and Privacy*, 2022. URL: https://eprint.iacr.org/2021/100.

28 Theo Diamandis and Guillermo Angeris. A note on the welfare gap in fair ordering, 2023. doi:10.48550/arXiv.2303.15239.

29 Justin Drake. Encrypted mempools. https://www.youtube.com/watch?v=XRM0CpGY3sw.

30 Matheus Venturyne Xavier Ferreira and David C. Parkes. Credible decentralized exchange design via verifiable sequencing rules. In *STOC*, 2023.

31 Tivas Gupta, Mallesh M Pai, and Max Resnick. The centralizing effects of private order flow on proposer-builder separation, 2023. arXiv:2305.19150.

32 Lioba Heimbach and Roger Wattenhofer. Eliminating sandwich attacks with the help of game theory. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, 2022.

33 Josojo. Mev capturing amm (mcamm). https://ethresear.ch/t/mev-capturing-amm-mcamm/13336.

34 Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. In *APKC '22: Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop*, 2022.

35 Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus, 2021.

36 Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *CRYPTO*, pages 451–480, 2020. doi:10.1007/978-3-030-56877-1_16.

37 Kshitij Kulkarni, Theo Diamandis, and Tarun Chitra. Towards a theory of maximal extractable value I: constant function market makers. *CoRR*, abs/2207.11835, 2022. doi:10.48550/arXiv.2207.11835.

38 F. Leupold. Cow native amms (aka surplus capturing amms with single price clearing). https://forum.cow.fi/t/cow-native-amms-aka-surplus-capturing-amms-with-single-price-clearing/1219/1.

39 Yuhao Li, Mengqian Zhang, Jichen Li, Elynn Chen, Xi Chen, and Xiaotie Deng. Mev makes everyone happy under greedy sequencing rule. In *Proceedings of the 2023 Workshop on Decentralized Finance and Security*, DeFi '23, pages 9–15. Association for Computing Machinery, 2023. doi:10.1145/3605768.3623543.

40 Jason Milionis, Ciamac C. Moallemi, and Tim Roughgarden. A myersonian framework for optimal liquidity provision in automated market makers. *CoRR*, abs/2303.00208, 2023. doi:10.48550/arXiv.2303.00208.

41 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *43rd IEEE Symposium on Security and Privacy, SP*, 2022.

42 Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. Attacking the defi ecosystem with flash loans for fun and profit. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I*, 2021. doi:10.1007/978-3-662-64322-8_1.

43 Geoffrey Ramseyer, Mohak Goyal, Ashish Goel, and David Mazières. Augmenting batch exchanges with constant function market makers, 2023. arXiv:2210.04929.

44 Tim Roughgarden. Transaction fee mechanism design. In *EC*, 2021.

45 Elaine Shi, Hao Chung, and Ke Wu. What can cryptography do for decentralized mechanism design? In *ITCS*, volume 251 of *LIPIcs*, pages 97:1–97:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ITCS.2023.97.

46 Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *ACM Computing Surveys*, 55(11), 2023. doi:10.1145/3570639.

47 Sen Yang, Kartik Nayak, and Fan Zhang. Decentralization of Ethereum's Builder Market. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1512–1530, Los Alamitos, CA, USA, May 2025. IEEE Computer Society. doi:10.1109/SP61157.2025.00157.

48 Mengqian Zhang, Yunhao Li, Xinyuan Sun, Elynn Chen, and Xi Chen. Computation of optimal mev in decentralized exchanges. https://mengqian-zhang.github.io/papers/batch.pdf.

49 Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. On the just-in-time discovery of profit-generating transactions in defi protocols. In *IEEE Symposium on Security and Privacy, SP*, 2021.

50 Liyi Zhou, Kaihua Qin, and Arthur Gervais. A2MM: mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. *CoRR*, abs/2106.07371, 2021. arXiv:2106.07371.

51 Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc Viet Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *IEEE Symposium on Security and Privacy*, 2021.

52 Patrick Zust. Analyzing and preventing sandwich attacks in ethereum. Bachelor's thesis.

Strategic Analysis of Just-In-Time Liquidity Provision in Concentrated Liquidity Market Makers

Bruno Llacer Trotti
Federal University of Rio de Janeiro (UFRJ), Brazil

Weizhao Tang
Carnegie Mellon University, Pittsburgh, PA, USA

Rachid El-Azouzi
Avignon University, France

Giulia Fanti
Carnegie Mellon University, Pittsburgh, PA, USA

Daniel Sadoc Menasché
Federal University of Rio de Janeiro (UFRJ), Brazil

Abstract

Liquidity providers (LPs) are essential figures in the operation of automated market makers (AMMs); in exchange for transaction fees, LPs lend the liquidity that allows AMMs to operate. While many prior works have studied the incentive structures of LPs in general, we currently lack a principled understanding of a special class of LPs known as Just-In-Time (JIT) LPs. These are strategic agents who momentarily supply liquidity for a single swap, in an attempt to extract disproportionately high fees relative to the remaining passive LPs. This paper provides the first formal, transaction-level model of JIT liquidity provision for a widespread class of AMMs known as Concentrated Liquidity Market Makers (CLMMs), as seen in Uniswap V3, for instance. We characterize the landscape of price impact and fee allocation in these systems, formulate and analyze a non-linear optimization problem faced by JIT LPs, and prove the existence of an optimal strategy. By fitting our optimal solution for JIT LPs to real-world CLMMs, we observe that in liquidity pools (particularly those with risky assets), there is a significant gap between observed and optimal JIT behavior. Existing JIT LPs often fail to account for price impact; doing so, we estimate they could increase earnings by up to 69% on average over small time windows. We also show that JIT liquidity, when deployed strategically, can improve market efficiency reducing slippage for traders, albeit at the cost of eroding passive LP profits by up to 44% per trade on average.

2012 ACM Subject Classification Information systems → Electronic commerce

Keywords and phrases Concentrated Liquidity Market Makers, Uniswap, Just-in-Time

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.8

Related Version *Full Version*: https://arxiv.org/abs/2509.16157

Supplementary Material
Software (Source Code): https://github.com/brunoCCOS/JITUniswapOptimization

Funding This work was partially funded by CAPES; by FAPERJ under grant E-26/204.268/2024; and by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil, under grants 403601/2023-1 and 315106/2023-9. It was also supported in part by the National Science Foundation under grant CNS-2325478, as well as by the Initiative for Cryptocurrencies and Contracts (IC3) and the CyLab Secure Blockchain Initiative, together with their respective industry sponsors.

© Bruno Llacer Trotti, Weizhao Tang, Rachid El-Azouzi, Giulia Fanti, and Daniel Sadoc Menasché;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Recent years have seen increased adoption of Decentralized Exchanges (DEXs) [29] and Automated Market Maker (AMM) protocols [30], which automatically set trade prices and execute trades; prominent markets running AMMs include Uniswap [1], SushiSwap [24], and PancakeSwap [22]. At the time of writing, the annual trading volume on AMM-based exchanges exceeds hundreds of billions of dollars [19].

Liquidity providers (LPs) are users who lend tokens to specific pools, thereby providing liquidity for trades in exchange for trading fees. LPs are central to the effective operation of AMMs; consequently, many prior works have sought to analyze their behavior [7, 9, 12, 21, 25]. For example, several of these works characterize conditions under which LPs are incentivized to contribute liquidity, and strategies for doing so to optimize returns [8, 25].

Despite a growing body of work analyzing the incentives of LPs in AMMs, to our knowledge, few have studied an important class of LPs known as *Just-in-Time Liquidity Providers* (JIT LPs) [11, 26]. These are informed strategic agents who aim to extract profit by providing liquidity in a highly targeted manner. Rather than holding a liquidity position over a period of time, a JIT LP provides liquidity for a single trade. This can be executed by a sandwich attack [10] on the *target* transaction by executing a swap that involves frontrunning the target transaction with an "add liquidity" transaction and backrunning it with a "withdraw liquidity" transaction. By temporarily capturing a higher share of liquidity, the JIT trader can earn a larger portion of the trading fee that is paid to all LPs pro rata to each LP's liquidity share.[1]

Although JIT LPs may appear to operate at the margins of an AMM, they play a crucial role in the ecosystem. They act as a frictional force against price movements, they help pools stay aligned with external market opportunities, and they reduce slippage for traders [26, 28].[2] At the same time, they introduce adversarial costs to passive LPs and arbitrageurs. The prevalence of JIT LPs is expected to grow with the emergence of technologies such as Flashbots [17] and Uniswap V4 [2], which introduce optimized mechanisms for JIT operations (e.g., hooks, bundled transactions). However, we currently lack a theoretical understanding of JIT liquidity provision and its effects on AMMs.

This work fills a current gap in the literature by developing and analyzing a principled model of JIT LP incentives for an important class of AMMs known as *Concentrated Liquidity Market Makers (CLMMs)*. Introduced in Uniswap V3 [1], CLMMs allow LPs to invest liquidity over fine-grained price ranges; the LP's liquidity is only used to support trades when the AMM price lies within the specified range. This helps LPs reduce risk due to large price fluctuations. Today, CLMMs are widely used in Uniswap V3 and the mechanism is generalized in emerging Uniswap V4 markets [2].

Under a model of the CLMM paradigm, we formulate and analyze the JIT resource allocation optimization problem, study its solution properties analytically, and provide an algorithm for computing optimal strategies (§5). Our analysis reveals new insights into how optimal JIT strategies depend on transactional and pool-specific parameters.

We summarize our contributions as follows.

- **Price impact landscape:** We analyze how swap-induced price changes affect LP revenue and formally characterize the conditions under which liquidity provision leads to gains or losses. These effects are often called "impermanent loss"; we instead use the term "price

[1] Note that the "trading fee" (pool fee) is a percentage of the swap value paid by the trader to liquidity providers, whereas the "transaction fee" (gas fee) is paid to Ethereum validators to include the transaction in a block. In this paper, except otherwise noted, fees refer to trading fees.

[2] Slippage is a phenomenon by which traders lose money in an AMM due to price changes over the course of a single trade.

impact" to reflect that they do not always result in losses (Def. 2). This analysis yields a precise description of the parametric region in which LPs benefit from price movements (§2.3). This result holds for both *passive* (regular) LPs and JIT LPs.

- **Formulation and analysis of utility optimization for JIT LPs:** We model the decision-making process of JIT LPs as a non-linear optimization problem and define a transaction-level utility function that captures the trade-off between fee revenue and price impact. This formulation enables a rigorous exploration of the strategy and utility space (§2.2.1 and §4) . We classify optimal JIT behavior across three swap scenarios – overpriced, arbitrage-driven, and overshooting trades – and present an algorithm for identifying optimal liquidity positions (§5 and §6).
- **Empirical insights on real-world JIT performance:** Using on-chain data, we report three key empirical findings (§6). First, JIT traders today appear to allocate trades suboptimally, without accounting for price impact; our experiments suggest that this unawareness reduces JIT profit by up to 41% on average. Second, although JIT LPs currently account for a small share of total fee revenue, most of JIT revenue comes from price impact (93.7%) rather than extracting fees from passive LPs. Third, JIT liquidity provision can become a serious competitor to passive LPs by reducing their fee income by up to 40%, while slightly reducing the cost of slippage for traders.

2 Background and Model

2.1 Introduction to AMMs

Automated Market Makers (AMMs) enable decentralized token exchanges through algorithmic pricing instead of traditional order books. We begin by explaining one of the most common classes of AMMs known as constant product market makers (CPMMs) [6]. A CPMM maintains a liquidity pool of tokenized assets X and Y. After initialization, the CPMM maintains a pool price q, which is the price of token Y in terms of token X. When an LP enters the CPMM, they contribute an amount of liquidity L, which comprises token amounts x of X and y of Y, according to the following rule:

$$x = \frac{L}{\sqrt{q}}, \qquad y = L\sqrt{q}. \tag{1}$$

Consequently, the token amounts x, y of each liquidity position of amount L always follow the constant product rule $xy = L^2$. Additionally, when there exist multiple liquidity positions $\{(x_i, y_i, L_i)\}_{i=1}^N$, the corresponding summations also follow the constant product rule

$$x_{\text{total}} \cdot y_{\text{total}} = \left(\sum_{i=1}^N x_i\right) \cdot \left(\sum_{i=1}^N y_i\right) = \left(\sum_{i=1}^N L_i\right)^2 = L_{\text{total}}^2. \tag{2}$$

A CPMM supports three basic operations – mint (add liquidity), burn (remove liquidity), and swap (trade).

- Mint and burn transactions are executed by an LP who specifies liquidity L to add/remove, and will deposit/withdraw token amounts (x, y) following (1). q remains constant.
- Swaps are executed by a trader who specifies an amount of either token to pay. Without loss of generality, assume the trader pays Δx tokens of type X. L remains unchanged for each liquidity position, so the trader will get Δy Y tokens following (2):

  $$(x_{\text{total}} + \Delta x)(y_{\text{total}} - \Delta y) = L_{\text{total}}^2.$$

  As a result, the pool has a new price $q' = (y_{\text{total}} - \Delta y)/(x_{\text{total}} + \Delta x)$ and the equivalent token amount (x', y') of each liquidity position L will update by plugging q' into (1).

Slippage is an important concept in this domain; it refers to the difference between the execution price of a trade and the price expected at the time of trade initiation. In the context of AMMs, slippage arises from the fact that large trades move the price along the bonding curve, resulting in a marginally worse exchange rate for each successive unit traded. Slippage is an inherent byproduct of price impact in AMMs.

▶ **Example 1** (Slippage). Consider an AMM with 100 ETH and 10,000 USDC, so $q = \frac{\#\text{ETH}}{\#\text{USDC}} = 100$. Suppose a trader wishes to buy 10 ETH. Due to the constant-product formula $x \cdot y = k$, the required USDC input is such that $100 \times 10{,}000 = (100 - 10) \times y_{\text{final}}$, i.e., $y_{\text{final}} \approx 11{,}111.11$. Thus, the trader must pay approximately 1,111.11 USDC to receive 10 ETH, implying an average price of 111.11 USDC per ETH – higher than the original pool price. The slippage is: $(111.11 - 100)/100 = 11.11\%$. This increase reflects the adverse price movement caused by the trader's own order.

2.2 Concentrated Liquidity Market Makers

Since the emergence of CPMMs, more complex AMM designs have been proposed; among the most widespread of these is Concentrated Liquidity Market Makers (CLMMs) [14]. They allow LPs to choose not only the amount of liquidity they wish to add to the pool, but also the price range in which the liquidity is active. When an LP adds liquidity to the CLMM, the LP creates a *liquidity position* by specifying a tuple (L, a, b), where L is the amount of liquidity, and (a, b) represents the price range in which the position is effective. During a trade, if the AMM price exits the specified range (a, b) of an LP's position, that liquidity will not be used to support the trade outside the specified price range. At price q, this liquidity position reserves token amounts

$$x = x^{(a,b)}(q) \triangleq L \cdot \left(\frac{1}{\sqrt{\hat{q}_{a,b}}} - \frac{1}{\sqrt{b}} \right), \qquad y = y^{(a,b)}(q) \triangleq L \cdot \left(\sqrt{\hat{q}_{a,b}} - \sqrt{a} \right), \tag{3}$$

where $\hat{q}_{a,b} = \min\{b, \max\{a, q\}\}$ is the projection of the price q onto the interval (a, b). By setting $(a, b) = (0, \infty)$, this relation reduces to (1), which implies that a CPMM is a special case of a CLMM when every LP chooses the full price range. However, in general cases where $0 < a < b < \infty$, the reserves of the CLMM liquidity position remain constant with $x = 0$ when $q \geq b$ and with $y = 0$ when $q \leq a$. This implies that 1) the position is unused when q moves out of range (a, b), and 2) there exists a maximum reserve of token X and token Y for such positions.

A CLMM also supports mint, burn and swap operations, as long as they are consistent with the total reserves in the pool (e.g. do not try to swap tokens that do not exist). A swap outputs Δy in the following steps:

1. **Calculate the new price q'**: Since, for each position, x is a continuous and monotonic function of q, the total liquidity reserve x_{total} is also continuous and monotonic in q. Therefore, we can determine the post-swap price q' by solving for the value of q' that satisfies $x_{\text{total}}(q') = x_{\text{total}}(q) + \Delta x(q')$, which is well-defined due to the invertibility of $x_{\text{total}}(\cdot)$. Provided that q' remains within the union of all active price ranges, the swap can be successfully executed (refer to [1] for implementation details).
2. **Update the token reserve**: Each position keeps (L, a, b) constant, so we recalculate x' and y' by plugging q' into (3). The change in the second token, Δy, can then be obtained by summing all y' values.

By concentrating liquidity within a narrower price range, CLMMs enable each active position to support a higher volume of trades with the same amount of capital. This increased capital efficiency was designed to reduce price slippage for traders - as larger trades can be absorbed with smaller price movements [1]- and to allow LPs to extract more from their money, since they can have a higher liquidity, therefore a higher share of the fees, using the same budget by picking a narrower range. At the same time, when the market price exits a position's specified range, that liquidity becomes inactive or "frozen", and only the remaining active positions support the trade. As a result, traders of such trades may experience higher slippage compared to a scenario where liquidity is distributed across the entire price spectrum.

In practice, CLMMs discretize the price axis using a system of *ticks*. Let T be the set of possible ticks, $T = \{t_i \mid i = 0, \dots, M\}$, with $M \in \mathbb{Z}_{>0}$.[3] As a result, the set of valid price ranges is given by $\mathcal{R} = \{(a, b) \in T \times T \mid a < b\}$. Liquidity providers are restricted to choosing ranges $(a, b) \in \mathcal{R}$, so liquidity remains constant between consecutive ticks (t_m, t_{m+1}). Within each such interval (or across consecutive intervals with identical liquidity), price changes follow the structure defined in (3), which reveals a tight coupling between price movement and the liquidity available in the pool.

2.2.1 Fees

One of the incentives for liquidity providers to participate in the pool is the collection of fees. Each transaction pays a fee proportional to the input amount, and this fee is distributed pro-rata among all providers with active liquidity in the corresponding price range. The fee amount varies across pools, each characterized by a parameter $\alpha \in (0, 1)$ that determines the fraction of the input amount collected as a trading fee.

In a CLMM, every time a provider mints a position (L, a, b), they increase the liquidity present in every tick $t_m \in (a, b)$. To assess the liquidity available at a specific tick m, we sum the liquidity of all positions that include it. Formally, we define the liquidity provided at tick m by a provider $n \in [N]$ as $P_{n,m}$ and calculate the total liquidity as:

$$P_m = \sum_{n \in [N]} P_{n,m} \tag{4}$$

Let $\delta = (\delta_m)_{m=1}^{M}$ denote the vector of fees (in dollars) collected per tick during a swap. Then, $\delta_m \triangleq \Delta x_m \cdot \alpha \cdot p_x$, where Δx_m is the amount of tokens exchanged in tick m

$$\Delta x_m = P_{n,m} \left(\frac{1}{\sqrt{\hat{q}'_m}} - \frac{1}{\sqrt{a_m}} \right).$$

As discussed above, the total fee from a swap is distributed pro-rata among all providers with active liquidity. The fee earned by provider n is given by:

$$\mathcal{F}_n = \sum_{m=1}^{M} \delta_m \cdot \frac{P_{n,m}}{P_m},$$

where δ_m is the total fee from swap m and $P_{n,m}$ is the liquidity contributed by passive provider n over tick m.

[3] Ticks are defined by exponential spacing: $T = \{t_i \mid i = 0, \dots, M\}$, where $t_i = 1.0001^{\tau(i-\iota)}$ with $\tau, M \in \mathbb{Z}_{>0}$ and $\iota \in \mathbb{Z}$.

2.3 Price Impact (a.k.a. Impermanent Loss)

Prior literature commonly assumes that the price ratio between two tokens is stable and externally set by the market, given by $q_{\text{market}} = \frac{p_x}{p_y}$, where p_x and p_y denote the market prices of tokens X and Y, respectively [3, 4, 25]. However, the actual price within a CLMM pool is determined endogenously – it is only changed by trades in the pool. If it does not coincide with the market price, arbitrage activities become profitable. At equilibrium – when no arbitrage opportunities remain – the two values align, such that $q = q_{\text{market}}$. This alignment is often assumed in theoretical analyses, particularly in the absence of active participants like JIT LPs and arbitrageurs.

The divergence between the pool price q and the market price thus represents both a risk and an opportunity. On one hand, price mismatches expose LPs to impermanent loss – the difference in value between a liquidity position and the equivalent token holdings outside the pool. On the other hand, such mismatches create arbitrage opportunities for sophisticated agents, including JIT LPs. To encompass both favorable and adverse outcomes of price divergence, we adopt a more general term and refer to the impermanent loss effect as *price impact* throughout the remainder of this paper.

Consider an LP n who mints a position when the pool price is q and later withdraws it at pool price q'. Let p_x and p_y denote the external market prices of tokens X and Y at the time of minting, and let p'_x and p'_y be their respective market prices at the time of withdrawal. Following (3), we denote by $x_n(q)$ and $y_n(q)$ the token amounts associated with the position of LP n when the pool price is q (at minting), and by $x_n(q')$ and $y_n(q')$ the corresponding amounts when the price is q' (at withdrawal). The dollar value of the position at the time of minting and at the time of withdrawal is:

$$V_{\text{mint}}(L, p_x, p_y, q) = p_x \cdot x_n(q) + p_y \cdot y_n(q), \quad V_{\text{withdraw}}(L, p'_x, p'_y, q') = p'_x \cdot x_n(q') + p'_y \cdot y_n(q'). \tag{5}$$

▶ **Definition 2** (Absolute price impact). *The* absolute price impact *is the net change in value of the position:*

$$\mathcal{C}_n(\Delta x, L; p_x, p_y, p'_x, p'_y, q, q') \triangleq V_{mint}(L, p_x, p_y, q) - V_{withdraw}(L, p'_x, p'_y, q'). \tag{6}$$

In what follows, to simplify notation we omit the explicit dependencies on parameters of the above quantities, e.g., referring to $\mathcal{C}_n(\Delta x, L; p_x, p_y, p'_x, p'_y, q, q')$ simply as $\mathcal{C}_n$. Note that a *negative* price impact ($\mathcal{C}_n < 0$) corresponds to a *gain* for the liquidity provider relative to holding the assets outside the pool. Conversely, a *positive* price impact implies a relative loss.

▶ **Definition 3** (Relative price impact). *The* relative price impact *is the ratio of this value difference to the initial position value:*

$$\mathbf{PI} \triangleq \mathcal{C}_n / V_{mint}. \tag{7}$$

Throughout the rest of this paper, we omit the LP subscript n when the context is clear, and refer to quantities such as $x_n(q)$, $y_n(q)$ simply as $x(q)$, $y(q)$.

3 When is Price Impact Beneficial to LPs?

As explained previously, an LP n gains when the fees accrued $\mathcal{F}_n$ are larger than the price impact $\mathcal{C}_n$. However, LPs may struggle to predict these tradeoffs for three reasons: (1) Price fluctuations and transaction patterns are inherently unpredictable. (2) Given information about price and transaction fluctuations, we currently lack closed-form expressions showing

when LPs will benefit as a function of transaction size and current market conditions. (3) Even if we had knowledge of items (1) and (2), most LPs invest on a slow timescale (days or even weeks [16]), over which it is not possible to take advantage of per-transaction profits.

A key observation is that Just-In-Time (JIT) LPs do not suffer from the 1st and 3rd constraints – they can act quickly enough to move on a per-transaction basis using current price data. Hence, if we can resolve item (2), JIT LPs can determine with some certainty when CLMM transactions will be profitable.

In this section, we resolve challenge (2) by formally characterizing the conditions under which $\mathcal{C}$ is non-positive or positive in CLMMs; we show that the answer depends on whether the pool price moves toward or away from the prevailing market price. Our results in this section are not specific to JIT LPs, but as mentioned previously, passive LPs may not be able to take advantage of these results, as they move their liquidity on much slower time scales.

Assumptions. While deriving the following results, we assume that the market prices of tokens X and Y remain constant throughout the trade, i.e., $p_x = p'_x$ and $p_y = p'_y$. This is because the trade occurs instantaneously when the block including the transaction is finalized on the blockchain. In this case, the price impact simplifies to the difference in the dollar value of the LP's position before and after the trade, evaluated using fixed market prices. According to Definition 2:

$$\mathcal{C}(\Delta x, L; p_x, p_y, q, q') = p_x\left(x(q) - x(q')\right) + p_y\left(y(q) - y(q')\right) = -p_x\Delta x(q,q') - p_y\Delta y(q,q'). \quad (8)$$

In what follows, to simplify notation we refer to $\mathcal{C}(\Delta x, L; p_x, p_y, q, q')$ simply as $\mathcal{C}$.

3.1 Conditions for Favorable Price Impact

Our first main result consists of necessary and sufficient conditions under which a price change $q \to q'$ results in non-positive price impact. This threshold condition delineates precisely when an LP's position becomes favorable or unfavorable relative to simply holding the tokens at market prices

▶ **Theorem 4** (Threshold condition for price impact). *Let $\hat{q} \triangleq \max\{a, \min\{b, q\}\}$, and similarly for $\hat{q}'$. The change in pool price relates to price impact as follows:*

Case 1: *$q' < q$: $\mathcal{C} \le 0$ if and only if*

$$\frac{1}{\hat{q}}\left(\frac{p_x}{p_y}\right)^2 \ge \hat{q}'. \quad (9)$$

Case 2: *$q < q'$: $\mathcal{C} \le 0$ if and only if*

$$\frac{1}{\hat{q}}\left(\frac{p_x}{p_y}\right)^2 \le \hat{q}'. \quad (10)$$

Proof sketch. (Full proof in [18]) We establish conditions under which an LP experiences non-positive price impact. Replacing $p_x = p'_x, p_y = p'_y$ into Eq.(7):

$$\mathbf{PI} = 1 - \frac{V_{\text{withdraw}}(L, p_x, p_y, q')}{V_{\text{mint}}(L, p_x, p_y, q)} \le 0 \iff 1 \le \frac{p_x x_n(q') + p_y y_n(q')}{p_x x_n(q) + p_y y_n(q)}. \quad (11)$$

Therefore, the inequality holds iff $V_{\text{withdraw}} \ge V_{\text{mint}}$. The result follows from substituting Eq. (3) into the above expression, followed by algebraic manipulation. ◀

Based on this result, we can establish sufficient conditions for either favorable or adverse price impact. These are summarized in the following corollary.

▶ **Corollary 5** (Gains under diverging prices and losses under converging prices). *When initial and final prices do not cross p_x/p_y, gains and losses are determined by the direction of the price movement:*

- *Gain under diverging prices:* $\mathcal{C} \leq 0$ *if* $q' < q \leq p_x/p_y$ *or* $q' > q \geq p_x/p_y$*, i.e., if a trade moves the AMM price* away *from the external market price, the LP experiences gains.*
- *Loss under converging prices:* $\mathcal{C} \geq 0$ *if* $q < q' < p_x/p_y$ *or* $q > q' > p_x/p_y$*, i.e., if a trade moves the AMM price* towards *the external market price, the LP experiences losses.*

▶ Remark 6 (No-loss if pool price initially aligned with market). Note that if the pool price is initially aligned with the market price, i.e., $q = p_x/p_y$, then the price impact $\mathcal{C}$ experienced by a liquidity provider is *always* non-positive: $\mathcal{C} \leq 0$. This immediately follows from the fact that there is gain under diverging prices (first case considered in the above corollary), and if the market price is initially aligned with pool price, prices will diverge.

Intuitively, when the pool price moves toward the market price (first case in Corollary 5), traders obtain better execution, acquiring tokens from the pool at prices closer to fair market value – resulting in a loss for LPs. Conversely, when the pool price moves away from the market price (second case in Corollary 5), traders face worse execution, effectively overpaying and enriching LPs.

3.2 The Effects of Liquidity

We analyze the limiting behavior of price impact as available liquidity changes. Suppose we have a price range (a, b) which is currently covered by total liquidity L by passive LPs. A JIT LP is considering adding liquidity to this range; intuitively, when liquidity L tends to zero, the position is too small to have any meaningful participation, and the price impact vanishes. Conversely, as $L \to \infty$, the position becomes large enough that the price becomes almost static. Formally, we establish the following result

▶ **Lemma 7** (Limiting behavior of price impact)**.** *Let* Δx *be the quantity of token* X *exchanged in a trade. Consider* $q \in (t_m, t_{m+1}]$ *the current pool price and consider* (L, a, b) *to be the single position of some provider* n *such that* $q \in (a, b)$*, in words, consider* L *to be all the liquidity minted before the transaction by some provider* n*. Then:*

$$\lim_{L\to\infty} \mathcal{C} = \Delta x \cdot (p_y \cdot q - p_x), \quad \textit{and} \quad \lim_{L\to 0} \mathcal{C} = 0.$$

Proof sketch. (Full proof in [18]) The proof follows from Eq. (8) and the following facts derived in [1] (see also Eq. (3)): $\frac{\Delta x}{L} \triangleq \Delta \frac{1}{\sqrt{q}} = 1/\sqrt{q'} - 1/\sqrt{q}$ and $\frac{\Delta y}{L} \triangleq \Delta \sqrt{q} = \sqrt{q'} - \sqrt{q}$. Then, $\Delta y = -\Delta x \sqrt{q'}\sqrt{q}$. As $L \to \infty$, we have $q' \to q$. In the limit, $\Delta y \to -q\Delta x$, and $\mathcal{C} \to -p_x \Delta x + p_y q \Delta x$ (see Eq. (8)). As $L \to 0$, the LP's position is infinitesimal, yielding negligible participation in the trade and thus $\mathcal{C} \to 0$. ◀

This result reinforces the intuition that liquidity acts as a dampener of price impact: as liquidity increases, the pool becomes less sensitive to trade-induced price shifts, and the value transfer is increasingly governed by the degree of misalignment between pool and market prices. To illustrate the broader implications of price dynamics and their relationship with price impact, we present the following example.

▶ **Example 8.** Figure 1 provides a geometric interpretation of price impact in the (q, q')-plane, where the market price ratio is fixed at $p_x/p_y = 1$. The space is partitioned into colored regions: green for $\mathcal{C} \leq 0$ (favorable to LPs), and red for $\mathcal{C} \geq 0$ (unfavorable to LPs). Corollary 5 provides sufficient conditions for LPs to experience non-positive price impact – Regions VI and II. Conversely, Corollary 5 also identifies conditions under which price impact is non-negative – Regions VII and III. More intricate behavior arises when the price crosses

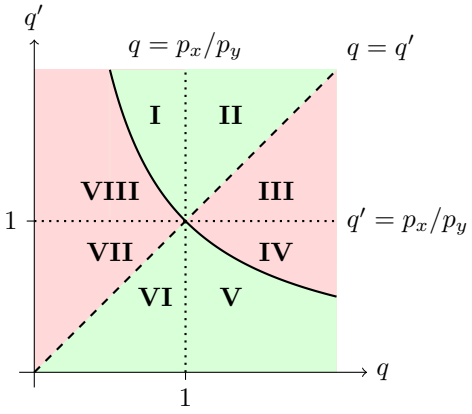


Figure 1 Partition of the (q, q')-space with $p_x/p_y = 1$; black curve corresponds to $q' = 1/q$. Red and green regions correspond to $\mathbf{PI} \geq 0$ and $\mathbf{PI} \leq 0$, respectively.

the market value (Theorem 4). For example, in Region I ($q' > p_x/p_y > q$ and $q' > 1/q$), price impact is favorable; in Region VIII ($q' < 1/q$), it becomes unfavorable. A symmetric situation arises for Regions IV and V depending on whether $q' < 1/q$.

Analogous to our analysis of price impact, we can show that the fee function is continuous with respect to the liquidity variable and converges to a finite limit as liquidity grows large. This result is formalized in the following lemma

▶ **Lemma 9** (Effect of Liquidity). *Let q be the current price in a CLMM, where $q \in (t_m, t_{m+1}]$. Suppose we have a target trade that pays Δx X tokens to the pool and asks for Y tokens, which can be supported by the CLMM. Immediately before the trade, a JIT LP adds liquidity $L \in [0, \infty)$ to price range (a, b) where $a \leq t_m < t_{m+1} \leq b$. Let $q'(< q)$ be the resulting price after the trade as a function of L. Then, q' is continuous, strictly decreasing in L, and $\lim_{L\to\infty} q' = q$.*

Proof sketch. (Full proof in [18]) We model the final price q' after the swap as a function of added liquidity L. The swap demand Δx must be fulfilled across ticks, so we express q' as an implicit function of L. As L increases, the same Δx has less price impact, so $q' \uparrow$. The mapping is strictly decreasing in L and continuous due to the functional form of the AMM and the monotonicity of the inverse square root. As $L \to \infty$, $q' \to q$, meaning the AMM price becomes increasingly resistant to change. ◀

▶ Remark 10. Note that, since $P_m = \sum_{i\in[N]} P_{n,m}$, Lemma 9 is also true for any $P_{n,m}$; that is, $\lim_{P_{n,m}\to\infty} q' = q \; \forall n$ such that $t_m < q \leq t_{m+1}$. This means that it is sufficient for **any** provider to inject high amount of liquidity in order to force the limit condition.

Lemma 9 highlights one of the central messages of this work: by concentrating liquidity when and where it is needed, JIT LPs can dampen price volatility, controlling slippage and choosing how the price will change. This empowers the JIT LP to optimize their revenue by balancing earned fees and losses due to price changes (See §2.3).

4 Just-In-Time Liquidity Allocation: An Optimization Perspective

We now define the formal structure of the decision process faced by a JIT LP observing a pending transaction. We begin by explaining the relevant properties of JIT LPs (§4.1),

which informs our model, including the strategy space available to the JIT LP (§4.2.1), and the utility function that governs decision-making. Finally, we establish conditions under which an optimal strategy exists (§4.2.2).

4.1 Background on Just-In-Time LPs

Unlike ordinary LPs who passively leave their liquidity in the liquidity pool, Just-In-Time (JIT) LPs react to market conditions in real-time to extract profit [26]. We next present two important aspects of JIT LP behavior, and explain how we model them.

(1) Per-transaction optimization. JIT LPs operate by monitoring the public *mempool*[4] [27, 29] for trade opportunities. This allows JIT LPs to selectively decide whether to provide liquidity for each trade before the trade is validated and finalized on the blockchain.

Once a JIT LP identifies a favorable trade, it can submit a *bundle*, which is a set of one or more transactions submitted together for inclusion in a block, typically via a relay to a block builder (e.g., via FlashBots [17]). These bundles are guaranteed atomic execution: the entire bundle either succeeds or fails as a group, ensuring riskless, single-trade provisioning. Exploiting this atomicity, the JIT LP can execute various operations-e.g., *sandwich attacks*, which mint liquidity just before a swap and burn it immediately after [28].

Modeling Implications: JIT LPs exploit this information asymmetry to compute their expected utility for a given trade. Based on this foresight, they can make optimal, *transaction-level* decisions about whether to mint and burn liquidity, factoring in both trading fees and the impact of price movements.

This naturally induces a two-stage optimization framework: first, passive LPs select their liquidity allocation strategy for a longer period [25]; second, for each observed trade, JIT LPs react by solving an online optimization problem to maximize net expected revenue, potentially adjusting for inclusion costs such as gas fees or auction bids. Our model focuses on this second optimization.

(2) Competition among JIT LPs. With multiple JIT LPs potentially competing over the same trade, each constructs and submits a bundle. The bundle includes a tip to the block builder, who selects bundles to maximize the total value extractable from the block by adjusting transaction ordering, inserting additional transactions, or censoring transactions.

This creates a sealed-bid auction: each JIT LP submits a bundle with a bid, without visibility into others' bids, and the builder selects the one that maximizes their expected revenue, i.e., only the highest-bidding bundle is included. The winning JIT LP earns a portion of the fees and pays a tip bounded below by the second-highest bid. This mechanism forms the basis of our transaction-level utility model, where we analyze how JIT LPs optimize their strategy under fixed surplus and uncertain inclusion.

Modeling Implications: Modeling this, recall that for a swap of Δx tokens of X and fee rate α, the total fee generated by the swap is $\delta = \alpha \cdot p_x \cdot \Delta x$, where p_x is the market price of token X in dollars. Since both Δx and p_x are visible in the mempool, δ is known to all JIT LPs in advance. As a result, they compete over a fixed surplus, strategically offering portions of δ as tips to the block builder to secure inclusion.

[4] The mempool is a public set of pending transactions propagated through the blockchain peer-to-peer network.

4.2 Model

Formally, we represent the JIT optimization problem as a tuple $([N], \mathcal{U}, \mathcal{S}, \theta, \rho)$. Here, $[N]$ denotes the set of all passive liquidity providers in the pool. The strategy space $\mathcal{S}$ encapsulates all feasible actions available to the JIT LP, while $\mathcal{U}$ is the utility function mapping each strategy to a real-valued profit – computed as the difference between earned fees, incurred price impact, and bidding costs. The vector θ includes all relevant swap parameters, such as trade size, fee rate, price information, and the liquidity distribution of passive providers. Finally, ρ is a constant that denotes the JIT budget.

Note that the JIT optimization problem assumes that the passive LP positions are established a priori. We denote by $\boldsymbol{s}$ the strategy profile of passive LPs, encoded as a vector of liquidity allocations across ticks: $\boldsymbol{s} = (L_i, a_i, b_i)_{i \in [N]}$. The aggregate effect of the passive LP strategies $\boldsymbol{s}$ induces a liquidity distribution function $P = (P_m)_{m \in M}$, where each P_m denotes the total passive liquidity available at tick m. Formally, P is derived via a deterministic mapping from $\boldsymbol{s}$. In the remainder of this section, we use $\boldsymbol{s}$ to refer to the underlying strategies and P to denote the induced state of the pool.

4.2.1 Strategy Space $\mathcal{S}$

A JIT LP's choice of action consists in selecting one position[5] (L, a, b) to be minted immediately before a swap and burned immediately after. Since the entire state of the pool is observable at decision time, the JIT LP can make an informed choice based on current market and pool conditions. To formalize this, we define the swap as a 6-tuple $\theta \triangleq (\Delta x, q, P, p_x, p_y, \alpha)$, where:

- Δx is the amount of token X being exchanged by the trader in the target trade;
- q is the current pool price;
- $P = (P_m)_{m \in M}$ is the per-tick liquidity distribution from passive LPs;
- p_x and p_y are the external market prices (in dollars) of tokens X and Y, respectively;
- α is the pool's fee rate.

Let q^* denote the price after the swap without the presence of JIT LPs. We define

$$\mathcal{R}^{(q,q^*)} \triangleq \mathcal{R} \cap \left\{ (a,b) \middle| \forall m : a \leq t_m < t_{m+1} \leq b,\ [t_m, t_{m+1}] \cap (\min\{q, q^*\}, \max\{q, q^*\}) \neq \varnothing \right\}.$$

Intuitively, $\mathcal{R}^{(q,q^*)}$ includes all feasible price ranges that do not contain a subrange $[t_m, t_{m+1}]$ that is never "touched" when price moves from q to q^*. We claim that the JIT LP only considers price ranges in $\mathcal{R}^{(q,q^*)}$ for their choice of liquidity position. Otherwise, we can break the position (L, a, b) down to two positions (L, a, c) and (L, c, b) where one of their ranges does not intersect with the price moving range $(\min\{q, q^*\}, \max\{q, q^*\})$. Let it be (a, c) without loss of generality [25]. During the trade, (L, a, c) will incur fee income $\mathcal{F} = 0$ and zero token amount change, which further implies price impact $\mathcal{C} = 0$ since p_x and p_y are both constant. Hence, in comparison with an action that chooses (L, c, b) instead, the (L, a, b) has equal utility impact while wasting budget on (a, c). Accordingly, we define the JIT strategy space as:

$$\mathcal{S} = \left\{ (L,a,b) \;\middle|\; (a,b) \in \mathcal{R}^{(q,q^*)},\ L \cdot V_{(a,b)} \leq \rho,\ q \in [a,b] \right\} \cup \{\perp\},$$

where $\perp$ denotes non-participation, ρ denotes the LP's budget, and $V_{(a,b)}$ is the dollar cost per unit of liquidity at price q over the interval (a, b).

[5] We do not consider multiple-position actions for two main reasons: 1) it is commonly assumed in the literature for JIT to mint constant liquidity [28], and 2) the data provided in [25] reveals that transactions from 2024 to 2025 by JIT LPs include only a single position.

4.2.2 Utility Function

The objective of the JIT LP is to maximize a utility function $\mathcal{U}$, defined as the net profit obtained from participating in a given swap. This profit captures the trade-off between fees earned and the price impact incurred. Given a strategy $s = (L, a, b) \in \mathcal{S}$ and swap parameters θ, the utility is defined as:

$$\mathcal{U}(s;\theta) = \mathcal{F}(s;\theta) - \mathcal{C}(s;\theta), \tag{13}$$

where $\mathcal{F}(s;\theta)$ is the total fee income and $\mathcal{C}(s;\theta)$ denotes the price impact associated with the strategy. The JIT LP aims to solve:

$$s^* = \arg\max_{s\in\mathcal{S}} \mathcal{U}(s;\theta). \tag{14}$$

In practice, a JIT LP needs to additionally pay a cost v for placing bids in auctions (such as Flashbots [17]) to land the sandwich attack on the trade transaction. Ideally, we should incorporate the auction in our model. Two natural approaches are (1) to model the game among multiple JIT LPs, or (2) model the probability of winning the bid as a function of v. Both approaches require empirical study on historical auctions that involves not only auction winners, but more importantly, losers. Such data is not currently accessible, to the best of our knowledge, and it is unrealistic to make ad hoc assumptions. Hence, for simplicity, we assume v to be a constant that does not relate to the choice of liquidity position. As a result, we can easily extend our model by updating the budget constraint as $L \cdot V_{(a,b)} + v \le \rho$ and the utility function as $\mathcal{U}(s;\theta) = \mathcal{F}(s;\theta) - \mathcal{C}(s;\theta) - v$.

Given a specific strategy $s \in \mathcal{S}$ and the current swap context, we can compute the total utility associated with executing the strategy. From this point forward, we omit explicit dependence on θ and s whenever the context is clear.

The total fee earned by the JIT LP is decomposed tick-wise. Since the JIT LP deploys a single position (L, a, b), let L be the liquidity of the JIT on tick m. The JIT's share of the fees at tick $m \in M$ is:

$$\mathcal{F}_m \triangleq \underbrace{\Delta x_m \alpha p_x}_{\text{Fee } =\delta_m} \cdot \frac{L}{\sum_{i\in[N]} P_{i,m} + L}, \qquad \mathcal{F} \triangleq \sum_{m\in M} \mathcal{F}_m. \tag{15}$$

Let t_m denote the price at tick m. Recall that α, p_x are trade parameters, and that Δx_m depends on q' and denotes the amount of token X that is injected at each tick m (see §2.2.1). As the swap affects only the ticks crossed during execution, the tick-level fee δ_m is nonzero only for ticks m such that $t_m \in (q, q')$. Under these conditions, the fee allocation simplifies to:

$$\mathcal{F}_m = \delta_m \cdot \frac{L}{\sum_{i\in[N]} P_{i,m} + L}, \qquad \mathcal{F} = \sum_{m\in M \mid t_m\in(a,b)} \mathcal{F}_m. \tag{16}$$

From it, we can show the following lemma:

▶ **Lemma 11.** *Consider an LP n with a single position (L, a, b), such that $q \in (a, b)$. Let there be a swap that pays $\Delta x > 0$ X tokens. The total fee earned by the provider is given by:*

$$\mathcal{F}(L) = \sum_{\forall m | t_m\in(a,b)} \delta_m \cdot \frac{L}{L + \tilde{P}_{n,m}},$$

where $\tilde{P}_{n,m} = \sum_{i\neq n} P_{i,m}$ is the total liquidity in tick m without $L = P_{n,m}$. In addition, $\mathcal{F}(L)$ is continuous in L and is bounded above by $\alpha \cdot \Delta x$,

$$\lim_{L\to\infty} \mathcal{F}(L) \le \alpha \cdot \Delta x. \tag{17}$$

Proof sketch. (Full proof in [18].) We express the total fee earned by the LP as a sum over ticks, weighted by the LP's share of liquidity (16). Since each fee component depends continuously on L through the liquidity share, the full fee function is continuous in L. As $L \to \infty$, the LP's share of each tick approaches 1, but the total fee is still bounded above by $\alpha \cdot \Delta x$, since that is the maximum total fee generated by the swap. Hence, the limit exists and is bounded. ◀

The price impact $\mathcal{C}$ is likewise decomposed across ticks as:

$$\mathcal{C}_m \triangleq p_x \cdot (x(\hat{q}_m) - x(\hat{q}'_m)) + p_y \cdot (y(\hat{q}_m) - y(\hat{q}'_m)), \quad \mathcal{C} \triangleq \sum_{m \in M \mid t_m \in (a,b)} \mathcal{C}_m, \tag{18}$$

where $\hat{q}_m = \min\{t_{m+1}, \max\{q, t_m\}\}$ is the entry price, $\hat{q}'_m$ the exit price within tick m and $x(q)$, $y(q)$ are given by Eq. (3).

This decomposition allows the utility to be expressed in tick-wise form:

$$\mathcal{U}_m = \mathcal{F}_m - \mathcal{C}_m, \quad \mathcal{U} = \sum_{m \in M \mid t_m \in (a,b)} \mathcal{U}_m. \tag{19}$$

Recall that all parameters $\theta \triangleq (\Delta x, q, P, p_x, p_y, \alpha)$ are fixed by the given trade, and since q' depends on (L, a, b), the only free variables are $L, (a, b)$. Using Lemmas 9, 7, and 11, we establish the following theorem:

▶ **Theorem 12.** *The utility function $\mathcal{U}$, as defined in Eq.* (19)*, attains a global maximum over the strategy space $\mathcal{S}$. That is, there exists an optimal strategy $s^* \in \mathcal{S}$ such that $\mathcal{U}(s^*) \geq \mathcal{U}(s)$ for all $s \in \mathcal{S}$.*

Proof sketch. (Full proof in [18].) The utility function $\mathcal{U}$ depends on liquidity L and the chosen price range (a, b). Since tick prices are discrete, there are finitely many such intervals. For a fixed interval, $\mathcal{U}(L)$ is continuous due to the continuity of both fee ($\mathcal{F}$) and cost ($\mathcal{C}$) functions in L. Also, $\mathcal{F}$ is bounded above while $\mathcal{C}$ grows at most linearly in L, constrained by a budget ρ. Therefore, the domain of feasible L is compact, and $\mathcal{U}$ attains a maximum on it. Optimizing over all intervals proves existence of an optimal strategy. ◀

This guarantees that for any swap, there exists a profit-maximizing strategy for the JIT LP. In the following section, we propose an algorithm to compute this strategy and empirically explore how it behaves across different swap conditions.

5 An Algorithmic Solution to the JIT Optimization Problem

We now present an algorithmic approach for computing the optimal strategy $s^* = (L^*, a^*, b^*)$ that maximizes the utility $\mathcal{U}$ of a JIT LP in response to a given target swap. This is made possible by Theorem 12, which guarantees the existence of a global optimum.

The utility function can be challenging to optimize since there are many parameters and the utility function is nonconcave in general. In addition, we lack a closed-form formula for q', which is calculated algorithmically. To address these challenges, we use empirical observations to reduce the search space. First, we note that θ, the fixed trade parameters, are known beforehand by the JIT trader; this turns the utility into a function of three variables: a, b, L.

In theory, the JIT trader could choose any (a, b) interval, of which there are $O(M^2)$. However, due to the monotonicity of q' in the liquidity, the original price q cannot move farther than q^*, where q^* is the anticipated post-swap price in the *absence* of JIT intervention.

Hence, we limit ourselves to tick ranges $(a, b) \in \mathcal{R}^{(q,q^*)}$. While this does not reduce the asymptotic worst-case complexity of our search, in our data we observe that for 95% of transactions, there are at most 2 ticks between q and q^* (see §6). Hence, enumerating the intervals in $\mathcal{R}^{(q,q^*)}$ is practically feasible.

Algorithm 1 provides a computational method to search for the optimal JIT liquidity provisioning strategy in response to an observed swap. The algorithm iterates over all candidate price intervals $(a, b) \in \mathcal{R}^{(q,q^*)}$ (line 3). For each interval (a, b), it solves a one-dimensional, non-linear, non-concave optimization problem to identify the liquidity level L^* that maximizes the JIT LP's utility:

$$L^*(a, b; \theta) = \arg\max_L \left(\mathcal{F}^{(a,b)}(L; \theta) - \mathcal{C}^{(a,b)}(L; \theta) \right) \tag{20}$$

subject to the budget constraint $p_x x^{(a,b)}(q') + p_y y^{(a,b)}(q') \leq \rho$

$$\mathcal{F}^{(a,b)}(L; \theta) = \sum_{m \in M | t_m \in (a,b)} \Delta x_m \cdot \alpha \cdot \frac{L}{L + P_m} \cdot p_x \tag{21}$$

$$\mathcal{C}^{(a,b)}(L; \theta) = \sum_{m \in M | t_m \in (a,b)} \left(-p_x \Delta x_m + p_y \Delta x_m \hat{q}_m \cdot \frac{L + P_m}{\Delta x_m \sqrt{\hat{q}_m} + L + P_m} \right) \cdot \frac{L}{L + P_m}. \tag{22}$$

If the resulting utility exceeds the current maximum, the optimal strategy is updated accordingly (lines 6–8). After examining all admissible ranges, the algorithm returns the best strategy $s^* = (L^*, a^*, b^*)$ (line 11) that maximizes the JIT LP's profit in the given swap scenario. The optimal liquidity pair (a^*, b^*) and the corresponding optimal liquidity L^* are given by $L^* = \arg\max_L \mathcal{F}^{(a^*,b^*)}(L; \theta) - \mathcal{C}^{(a^*,b^*)}(L; \theta)$ such that $\mathcal{U}^{(a^*,b^*)}(L^*; \theta) = \mathcal{U}(s^*; \theta) \geq \mathcal{U}(s; \theta) \; \forall \; s \in \mathcal{S}$, where s^* is given by the triplet $s^* = (L^*, a^*, b^*)$ (see §4.2.1).

As long as we can find the global optimum of a bounded continuous univariate function, Algorithm 1 returns an optimal strategy to maximize JIT utility. In our experiments, we tried both Particle Swarm Optimization (PSO) and binary search to solve for L^*. Since in most cases the optimal solution was to use the entire budget, both algorithms found the best strategy, but in non-trivial cases, PSO performed better at the cost of more computational power. Our implementation of Algorithm 1 can be found on GitHub.[6]

▶ Remark 13 (Token Directionality). All derivations in this section assume $\Delta x > 0$, i.e., token X is being sold. To handle the symmetric case $\Delta y > 0$, where token Y is sold instead, we define a transformation $Y(\theta) = (\Delta y, 1/q, P, p_y, p_x, \alpha)$. This mapping ensures that all formulas and results continue to hold under the appropriate relabeling of tokens.

5.1 Strategic Archetypes in CLMM Swaps

We classify trades into three canonical scenarios – referred to as *strategic archetypes* – that capture the dominant patterns and analytical results of Theorems 4 and 12.

- **Overpriced trade:** The trade moves the pool price away from the market price, resulting in the purchase of the more expensive token. Formally, this occurs when $q' < q \leq p_x/p_y$ or $q' > q \geq p_x/p_y$. According to Corollary 5, such movements are associated with non-positive price impact, meaning the LP benefits (Figure 2(a1)).

[6] https://github.com/brunoCCOS/JITUniswapOptimization

Algorithm 1 JIT Optimal Strategy Search.

Require: Swap parameters $\theta = (\Delta x, q, P, p_x, p_y, \alpha)$ Budget ρ
Ensure: Optimal strategy $s^* = (L^*, a^*, b^*)$
Initialize $\mathcal{U}_{\max} \leftarrow -\infty$
Set $s^* \leftarrow \perp$
for each $(a, b) \in \mathcal{R}^{(q,q^*)}$ **do**
  Solve for optimal liquidity: $L^*(a, b; \theta)$ [Optimization (20)]
  Evaluate resulting utility: $\mathcal{U} \leftarrow \mathcal{F}^{(a,b)}(L^*; \theta) - \mathcal{C}^{(a,b)}(L^*; \theta)$
  if $\mathcal{U} > \mathcal{U}_{\max}$ **then**
    Update best utility: $\mathcal{U}_{\max} \leftarrow \mathcal{U}$
    Update best strategy: $s^* \leftarrow (L^*, a, b)$
  end if
end for
return s^*

- **Arbitrageur trade:** The trade brings the pool price closer to the market price by acquiring the cheaper token. This corresponds to $q < q' \leq p_x/p_y$ or $q > q' \geq p_x/p_y$. As shown in Corollary 5, this condition implies positive price impact, i.e., a loss for LPs. Although this trade is optimal for arbitrageurs, it is typically unprofitable for JIT LPs unless the fees compensate for the loss (Figure 2(a2)).
- **Overshoot trade:** A corrective trade crosses the market price, starting by buying the cheaper token but pushing the pool price past the market value, ultimately overpaying. This configuration – $q < p_x/p_y < q'$ or $q > p_x/p_y > q'$ – lies on the boundary conditions specified in Theorem 4. In this case, whether the price impact is favorable or not depends on whether the post-trade price q' crosses the threshold condition in (9) or (10) (Figure 2(a3)).

Recall that q is the pool price of token Y in terms of token X, so $1/q$ is the pool price of X in Y. When $q > p_x/p_y$, token Y is overpriced in the pool relative to the market, implying that X is relatively cheaper on-chain.

5.2 Optimized Investment Strategies by Archetype

Among all archetypes, participating in *overpriced trades* yields the highest utility. These trades allow JIT LPs to sell tokens above market value across the entire chosen range (a, b).

In *overshoot trades*, JIT LPs can profit by targeting only the favorable segment – i.e., by minting liquidity only in the price range where $\Delta x < 0$ and $q < p_x/p_y < q'$. By concentrating liquidity narrowly in this segment, JIT LPs increase their share of fees while avoiding loss-inducing zones. Figure 2(b) gives an illustrative example; the initial and final price sandwich the market price $q < p_x/p_y < q'$, so the trade is "overshooting" the price. Therefore, in all ticks between $(q, p_x/p_y)$ the price impact will be negative; hence the best strategy is to mint liquidity only on feasible ranges inside $(p_x/p_y, q')$ where the price impact is positive.

In contrast, engaging in *arbitrageur trades* is generally unfavorable unless the fee revenue is high enough to offset the incurred price impact. The following result formally establishes a sufficient condition under which fees are insufficient to cover price impact:

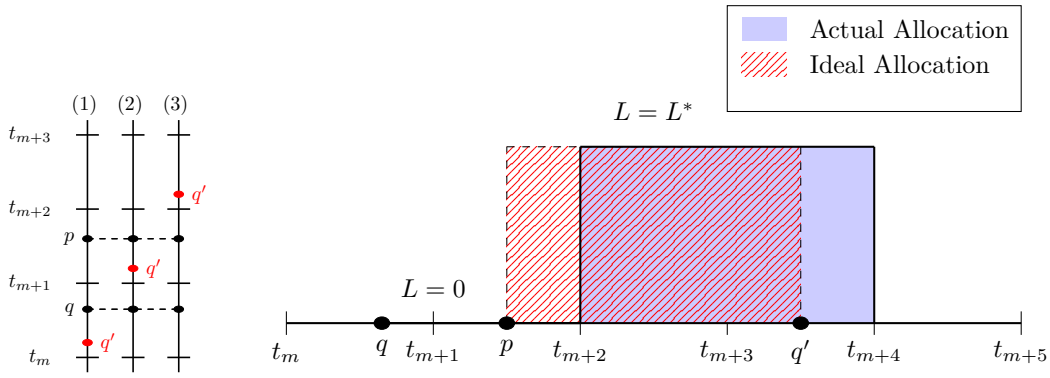


Figure 2 (a) Illustration of the three kind of trades. Each vertical line represents a price axis with different positions for q, p, q' which fully characterizes each type of trade. (1) **Overpriced** trade: q' is further away from p than q. (2) **Arbitrageur** trade: New price q is closer to the market price. (3) **Overshoot** trade. (b) Liquidity distribution across ticks as a function of price. In this overshoot trade example, the optimal strategy is to allocate the entire budget immediately after the pool price crosses the market price. Formally, all liquidity should be placed within the interval (p, q'), where $p = p_x/p_y$, as price impact is non-positive in this region. Since liquidity can only be minted at discrete tick levels, the optimal placement corresponds to the nearest available ticks – in this case, (t_{m+2}, t_{m+4}). This behavior extends to both overpriced and arbitrage trades, depending on tick granularity and pool conditions.

▶ **Proposition 14** (Fees Insufficient to Offset Price Impact). *For a transaction with swap parameters θ (definition in §4), suppose without loss of generality that $\Delta x > 0$. For a price range (a, b), if*

$$1 + \alpha < \frac{\Delta y_m \cdot p_y}{\Delta x_m \cdot p_x} \quad \forall m \in (p(a), p(b)),$$

i.e., if the gained value of Y tokens over the paid value of X tokens is more than the nominal fee rate, then utility is always negative: $\mathcal{U}^{(a,b)} < 0$.

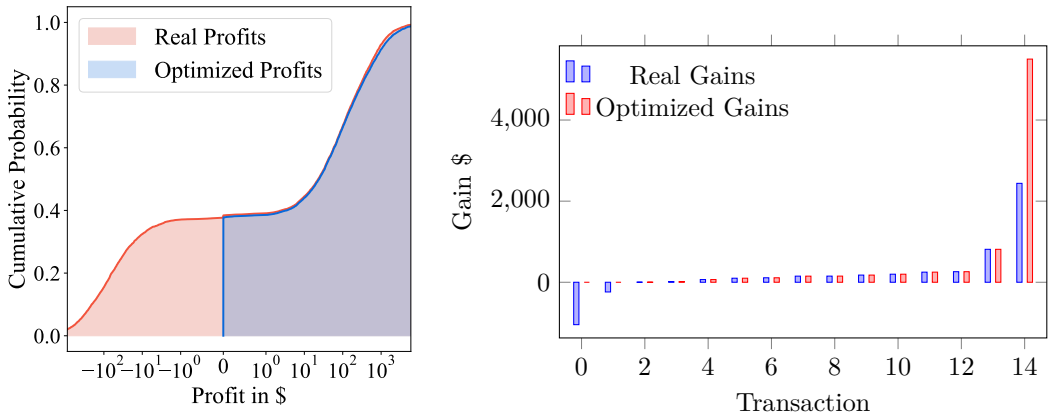
The proof of the above proposition can be found in [18].

6 Empirical Results Using On-Chain Data

In this section, we empirically evaluate the theoretical insights using data from Uniswap V3.

Data Collection. We collected data from a USDC/WETH[7] pool over 6 months: January-June 2024, which comprises 1,013,147 total transactions. The data was collected from the Allium platform [5]. From this data, we extracted all JIT transactions, which were identified by using the same criterion as [11], i.e. matching transactions which were sandwiched by a mint and burn transaction by the same provider. The resulting number of JIT swaps was 6,829. In this section, our results are generated by taking real JIT transactions and simulating our optimized version of JIT investment strategies. We assume throughout that a JIT transaction is realized if and only if it was realized in the source data (i.e., we do not simulate different results of bidding auctions among JIT LPs). To simulate a JIT budget, we

[7] 0x88e6a0c2ddd26feeb64f039a2c41296fcb3f5640 pool hash



(a) CDF of profit for a real JIT LP vs. optimized strategy. Optimized JIT avoids costly trades with high price impact.

(b) Sample of 15 transactions comparing absolute profits. While real JITs occasionally perform optimally, losses arise from engaging in losing trades and sometimes suboptimal allocation.

Figure 3 Comparison between actual and optimized JIT LP behavior. (a) focuses on aggregate performance, while (b) zooms in on individual transactions.

always assume that the JIT original transaction uses the trader's entire budget. Doing this, we ensure that our simulations do not use more money than the real execution, so our best results come exclusively from better resource utilization.

6.1 Results

We observe three main findings from our empirical evaluation:

1. JIT LPs in the pool we studied currently invest suboptimally and could significantly increase their profits. (§6.1.1)
2. JIT LP utility is primarily driven by price impact rather than fees. (§6.1.2)
3. If JIT LPs were to optimize their investment strategies, it would help JIT LPs and traders, at the expense of passive LPs. (§6.1.3)

6.1.1 Suboptimal Investment Strategies

Empirical data reveals that real-world JIT LPs underperform significantly relative to their potential gains, as illustrated in Figure 3. Many JIT agents engage in transactions where the adverse effects of price movements outweigh the fees earned, resulting in negative net utility (Figure 3(a)). In our sample of 6,884 JIT transactions, real JIT LPs could have earned up to 69% more than their current profit if they had adopted Algorithm 1. Our optimized allocation strategy not only helps JIT LPs identify when *not* to invest, but it can also increase returns on already-profitable transactions; for example, Figure 3(b) shows a random sample of 15 JIT transactions, comparing their realized gain in $ and the estimated gain under Algorithm 1. Our optimized strategy never loses money, and it sometimes increases gains relative to the current strategy of JIT traders. These observations highlight the importance of modeling price dynamics and underscore the need for strategy optimization beyond mere fee maximization.

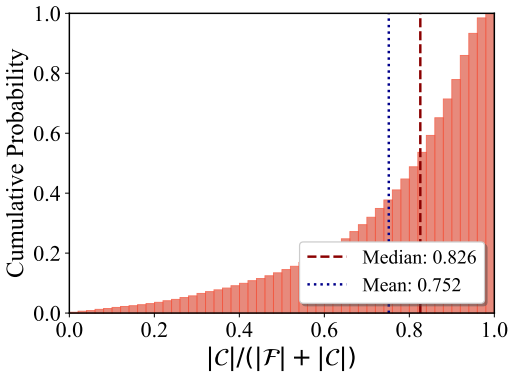


Figure 4 CDF of the proportion of JIT returns resulting from price impact as opposed to fees, i.e., $\frac{|\mathcal{C}|}{|\mathcal{C}|+|\mathcal{F}|}$. In most JIT transactions, price impact has a much greater impact on returns than fees, accounting for 75% of the total returns on average.

6.1.2 JIT Utility Is Driven Primarily by Price Impact

Our analysis shows that the primary driver of JIT profitability is not fee income but price impact (Figures 4). Optimized JIT strategies leverage price dislocations to extract value, often outperforming naive approaches that focus solely on fees. This illustrates the value of our study relative to prior works that focus on optimizing fee share [25].

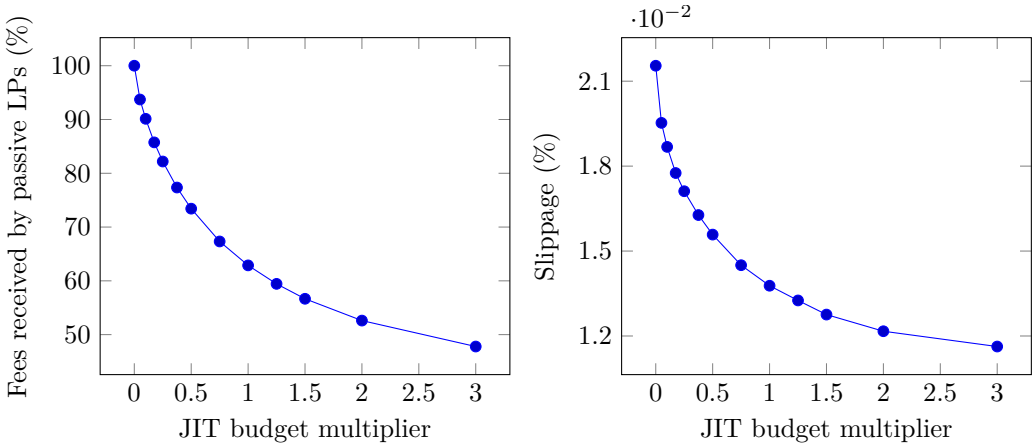
6.1.3 Market-Level Implications of Optimized JIT Behavior

Currently, the participation of JIT LPs in CLMMs is limited, and their share of fee revenue remains negligible when compared to passive LPs; in our dataset, JIT LPs claimed less than 2% of total fees paid in the system. However, simulations suggest that increased adoption of optimized JIT strategies could substantially reshape the CLMM ecosystem. In particular, JIT LPs affect both the distribution of fees among liquidity providers and the execution quality experienced by traders. Figures 5(a) and 5(b) illustrate two key market-level effects. As JIT LPs increase their capital allocation, they capture a growing share of the fee revenue, which reduces the earnings of passive LPs by up to 44% per trade, when the JIT budget is large (Figure 5(a)). At the same time, traders benefit from improved execution due to reduced price impact, as the injected liquidity narrows the effective spread and enables more efficient absorption of large orders (Figure 5(b)).

Averaging these values across transactions reveals two trends: the marginal benefit to passive LPs decreases with JIT participation, while traders benefit from reduced slippage. Together, these results indicate that although optimized JIT activity may diminish returns for passive LPs, it can improve capital efficiency and trading experience in CLMMs. This reinforces the view that well-designed JIT strategies can contribute positively to market structure.

7 Discussion and Implications

Our results highlight the strategic advantage of JIT in CLMMs. Specifically, a JIT LP that enters just before a trade – when the pool price is aligned with the market price – can benefit from the resulting swap-induced price movement. In contrast, passive LPs are exposed to losses when price deviations from the market are corrected via arbitrage.



(a) Fees received by passive LPs.

(b) Trader surplus attributable to JIT liquidity.

Figure 5 Both plots compare the effects of higher JIT budget with a baseline where no JIT engage (x=0). (a) Effect of JIT budget on passive LP fees. Passive LP fee gains diminish as JIT liquidity increases. (b) Trader benefits via reduced slippage with higher JIT budgets. The Y-axis shows the slippage suffered by the trader under different budgets (Recall slippage from §1).

Importantly, JIT profitability arises only under specific conditions (Proposition 14), often requiring precise timing and positioning (§5.1). Given the competitive nature of bidding for inclusion (e.g., via Flashbot bundles), it is not surprising that real-world JIT activity remains limited.

JIT LPs complement the actions of arbitrageurs. While arbitrageurs restore price alignment after deviations, JIT providers preemptively mitigate such deviations by injecting liquidity in trades with negative price impact $\mathcal{C}$. In this sense, JIT acts as a frictional force – reducing the magnitude of price dislocations (Figure 5) that would otherwise create larger arbitrage opportunities. This increased liquidity is beneficial to traders, as it reduces slippage and improves execution quality. However, it also comes at the expense of passive LPs, who face reduced fee income.

7.1 Real dynamics of providers, arbitrageurs and traders

A common perception is that arbitrageurs profit primarily at the expense of passive LPs through the price impact. However, this is not always the case. Consider a model where the market price P_t evolves stochastically, and the pool price Q_t follows it with some lag due to noise. Suppose a trade moves the pool price from $Q_t = P_t$ to $Q_{t+1} \neq P_{t+1}$, and an arbitrageur subsequently restores alignment at $Q_{t+2} = P_{t+2}$. If $P_t = P_{t+1} = P_{t+2}$, the passive LP's price impact $\mathcal{C}_n$ is zero, but fees $\mathcal{F}_n > 0$ are still earned from both trades.

In this case, the arbitrageur's profit comes not from the LP, but from the noisy trader who executed the initial swap at a worse price – e.g., buying at $\sqrt{Q_t Q_{t+1}} > Q_t = P_t$. The arbitrageur captures this overpayment at $\sqrt{Q_{t+1}Q_{t+2}}$. JIT LPs benefit even more from such trades. By sandwiching the swap, they can collect both price impact gains and fees. Moreover, their presence dampens the price displacement, thereby reducing the arbitrageur's profit and the passive LP's fee share from the second trade.

To illustrate this, suppose a noisy trade pushes the price to $Q_{t+1} > Q_t$. A JIT LP that enters before the trade adjusts the effective spot price to Q^*_{t+1}, such that $Q_{t+1} > Q^*_{t+1} > Q_t$. The arbitrageur, now facing a smaller price deviation, captures less profit: $\sqrt{Q^*_{t+1}Q_{t+2}} < \sqrt{Q_{t+1}Q_{t+2}}$.

Assuming all arbitrage opportunities are rapidly corrected, the pool price trajectory is driven by market price movements plus correctable noise. This noise does not generate price impact directly; rather, it creates temporary dislocations that can be monetized by arbitrageurs and JIT LPs. In fact, this type of exploitable noise is the basis of Loss-versus-Rebalancing (LVR) [20], a cost borne by passive LPs even when price impact vanishes.

JIT LPs exploit this noise by positioning liquidity around anticipated dislocations. Their interventions reduce price volatility, preempt arbitrage, and extract part of the fees that would otherwise accrue to passive LPs or arbitrageurs. If arbitrage profits stem from noisy trades rather than passive LPs, JIT LPs can be seen as first movers that mitigate price deviations while capturing a portion of this surplus.

8 Related Work

Despite a growing body of work analyzing LP incentives in AMMs, including recent theoretical models of JIT LPs [11, 26], prior efforts either address market-level dynamics under informational asymmetry [11] or empirically characterize JIT activity [26]. To our knowledge, no existing work models the transaction-level, optimization-based behavior of JIT LPs in CLMMs such as Uniswap V3. We fill this gap by formally modeling the per-transaction utility of JIT LPs based on price impact (§2.3) and fee accrual (§2.2.1), analyzing how optimal strategies vary with trade direction and liquidity depth, and comparing predictions with observed on-chain behavior.

Complementing our work, Cartea et al. [12] develop a continuous-time model for strategic liquidity provision in CLMMs. They derive optimal dynamic strategies for LPs adjusting liquidity ranges over time, accounting for fee income, predictable loss, and concentration risk. Their results show how exchange rate drift and rebalancing costs shape behavior. Unlike our discrete, transaction-level model where JIT LPs react to individual swaps, their focus is on long-term LPs managing ongoing exposure. Still, their analysis highlights the need to model fee income and adverse selection in CLMMs.

Tang et al. [25] present a game-theoretic model of liquidity provision in CLMMs, focusing on interactions between passive LPs. Their static model captures how budget constraints and price tick granularity affect equilibrium behavior, identifying a unique Nash equilibrium with a water-filling strategy. By comparing theory to on-chain behavior, they show LPs in risky pools deviate more from optimality than those in stable pools. However, they do not model or consider the effects of JIT LPs.

In this work, we analyze price impact as a key metric. Price impact and LVR coincide for individual trades [3, 4]. While LVR matters for passive LPs over time, JIT LPs primarily care about price impact.

Recent studies explore how informational asymmetries and adversarial timing affect CLMM dynamics [11, 13, 15, 23]. Capponi et al. [11] model adverse selection and value extraction by informed traders. Qin et al. [23] and Daian et al. [13] examine MEV, showing how latency arbitrage and frontrunning harm LPs. While these works address systemic risks, we focus on microstructure-level decisions by JIT LPs under fee competition and price impact. Our formal framework captures their transaction-specific optimization, and shows how misjudging price impact can erode profits – even for strategically-timed liquidity.

9 Conclusion

Emerging tools such as CLMMs, Flashbots, and Uniswap V4 Hooks are reshaping DeFi by enabling JIT liquidity providers to improve capital efficiency, manage risk, and align strategies with real-time on-chain dynamics. Yet, empirical evidence shows that most JIT agents fail to exploit profitable opportunities and often incur losses due to simplistic strategies.

This work demonstrates that significant gains are achievable via optimization-based approaches that explicitly weigh fee accrual against price impact. Our transaction-level framework guides JIT agents toward utility-maximizing strategies that allocate capital more effectively and reduce adverse outcomes such as slippage or loss.

More broadly, the results support a shift toward reactive, transaction-driven market models. By analyzing individual trades, rather than long-term aggregates, we obtain sharper insights into the microstructure of AMMs and the role of JIT behavior.

Future research should explore extensions of AMM models that account for JIT strategies and protocols that safeguard fairness for passive LPs. Closing the loop between protocol design and agent strategy is key to building more adaptive and sustainable DeFi ecosystems.

References

1 Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core. `https://uniswap.org/whitepaper-v3.pdf`, 2021. Accessed: 2025-08-07.

2 Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v4 core. `https://uniswap.org/whitepaper-v4.pdf`, 2024. Accessed: 2025-08-07.

3 Abe Alexander and Lars Fritz. Impermanent loss and loss-vs-rebalancing I: some statistical properties, 2024. `arXiv:2410.00854`.

4 Abe Alexander, Guillaume Lambert, and Lars Fritz. Impermanent loss and Loss-vs-Rebalancing II, 2025. `arXiv:2502.04097`.

5 Allium. Allium – Enterprise blockchain data platform. `https://www.allium.so/`. (Accessed on 10/04/2024).

6 Guillermo Angeris and Tarun Chitra. Improved price oracles: Constant function market makers. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, AFT '20, pages 80–91, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3419614.3423251`.

7 Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, and Tarun Chitra. An analysis of Uniswap markets, 2021. `arXiv:1911.03380`.

8 Alif Aqsha, Philippe Bergault, and Leandro Sánchez-Betancourt. Equilibrium reward for liquidity providers in automated market makers, 2025. `arXiv:2503.22502`.

9 Jan Arvid Berg, Robin Fritsch, Lioba Heimbach, and Roger Wattenhofer. An Empirical Study of Market Inefficiencies in Uniswap and Sushiswap. In *International Conference on Financial Cryptography and Data Security*, pages 238–249. Springer, 2022. `doi:10.1007/978-3-031-32415-4_16`.

10 Andrea Canidio and Robin Fritsch. Arbitrageurs' profits, lvr, and sandwich attacks: batch trading as an amm design response, 2025. `arXiv:2307.02074`.

11 Agostino Capponi, Ruizhe Jia, and Brian Zhu. The Paradox Of Just-in-Time Liquidity in Decentralized Exchanges: More Providers Can Sometimes Mean Less Liquidity. *arXiv preprint arXiv:2311.18164*, 2023. `doi:10.48550/arXiv.2311.18164`.

12 Álvaro Cartea, Fayçal Drissi, and Marcello Monga. Decentralized finance and automated market making: Predictable loss and optimal liquidity provision. *SIAM Journal on Financial Mathematics*, 15(3):931–959, 2024. `doi:10.1137/23M1602103`.

13 Philip Daian, Steven Goldfeder, et al. Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. In *IEEE S&P*, 2020.

14 Robin Fritsch. Concentrated liquidity in automated market makers. In *Proceedings of the 2021 ACM CCS Workshop on Decentralized Finance and Security*, DeFi '21, pages 15–20, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3464967.3488590.

15 Lioba Heimbach, Eric Schertenleib, and Roger Wattenhofer. Risks and returns of Uniswap V3 liquidity providers. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies (AFT)*, pages 89–101, 2022. doi:10.1145/3558535.3559772.

16 Alfred Lehar, Christine Parlour, and Marius Zoican. Fragmentation and optimal liquidity supply on decentralized exchanges, 2024. arXiv:2307.13772.

17 Zihao Li, Jianfeng Li, Zheyuan He, Xiapu Luo, Ting Wang, Xiaoze Ni, Wenwu Yang, Xi Chen, and Ting Chen. Demystifying defi mev activities in flashbots bundle. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, pages 165–179, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3576915.3616590.

18 Bruno Llacer Trotti, Weizhao Tang, Rachid El-Azouzi, Giulia Fanti, and Daniel Menasché. Strategic analysis of just-in-time liquidity provision in concentrated liquidity market makers (extended version). *arXiv e-prints*, 2025. URL: https://arxiv.org/abs/2509.16157.

19 Annika Masrani. Pancakeswap hits record $310b trading volume in 2024. *Nasdaq*, 2024. URL: https://www.nasdaq.com/articles/pancakeswap-hits-record-310b-trading-volume-2024.

20 Jason Milionis, Ciamac C. Moallemi, Tim Roughgarden, and Anthony Lee Zhang. Automated market making and loss-versus-rebalancing, 2024. arXiv:2208.06046.

21 Deborah Miori and Mihai Cucuringu. DeFi: Modeling and Forecasting Trading Volume on Uniswap v3 Liquidity Pools. *SSRN Electronic Journal*, May 2023. Available at SSRN: https://ssrn.com/abstract=4445351 or http://dx.doi.org/10.2139/ssrn.4445351. doi:10.2139/ssrn.4445351.

22 PancakeSwap. PancakeSwap: AMM on Binance Smart Chain. https://docs.pancakeswap.finance/, 2021. Accessed: 2025-05-25.

23 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 198–214. IEEE, 2022. doi:10.1109/SP46214.2022.9833734.

24 SushiSwap. SushiSwap System. https://docs.sushi.com/, 2020. Accessed: 2025-05-25.

25 Weizhao Tang, Rachid El-Azouzi, Cheng Han Lee, Ethan Chan, and Giulia Fanti. Game Theoretic Liquidity Provisioning in Concentrated Liquidity Market Makers. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 9(1):1–45, 2025. doi:10.1145/3711700.

26 Xin Wan and Austin Adams. Just-in-Time Liquidity on the Uniswap Protocol. *SSRN 4382303 Electron. J.*, 2023.

27 Shuzheng Wang, Yue Huang, Wenqin Zhang, Yuming Huang, Xuechao Wang, and Jing Tang. Private order flows and builder bidding dynamics: The road to monopoly in ethereum's block building market, 2024. doi:10.48550/arXiv.2410.12352.

28 Xihan Xiong, Zhipeng Wang, William Knottenbelt, and Michael Huth. Demystifying just-in-time (JIT) liquidity attacks on uniswap v3. Cryptology ePrint Archive, Paper 2023/973, 2023. URL: https://eprint.iacr.org/2023/973.

29 Deniz Yüksel. A Retrospective Analysis of Public and Private Order Flow on the Ethereum Blockchain. Master's thesis in informatics, Technische Universität München, 2024.

30 M. Yuksel et al. Automated market makers and decentralized exchanges: a defi primer. *Financial Innovation*, 7(1):1–27, 2021. URL: https://jfin-swufe.springeropen.com/articles/10.1186/s40854-021-00314-5.

Money in Motion: Micro-Velocity and Usage of Ethereum's Liquid Staking Tokens

Benjamin Kraner
Blockchain & Distributed Ledger Technologies Group, University of Zurich, Switzerland

Luca Pennella
Department of Economics, Business, Mathematics and Statistics, University of Trieste, Italy

Nicolò Vallarano
UZH Blockchain Center, University of Zurich, Switzerland

Claudio J. Tessone
UZH Blockchain Center, University of Zurich, Switzerland

Abstract

We introduce a micro-velocity framework for analysing the on-chain circulation of Lido's liquid-staking tokens, stETH, and its wrapped ERC-20 form, wstETH. By reconstructing full transfer and share-based accounting histories, we compute address-level velocities and decompose them into behavioural components. Despite their growing importance, the micro-level monetary dynamics of LSTs remain largely unexplored. Our data reveal persistently high velocity for both tokens, reflecting intensive reuse within DeFi. Yet activity is highly concentrated: a small cohort of large addresses, likely institutional accounts, are responsible for most turnover, while the rest of the users remain largely passive. We also observe a gradual transition in user behavior, characterized by a shift toward wstETH, the non-rebasing variant of stETH. This shift appears to align with DeFi composability trends, as wstETH is more frequently deployed across protocols such as AAVE, Spark, Balancer, and SkyMoney.

To make the study fully reproducible, we release (i) an open-source pipeline that indexes event logs and historical contract state, and (ii) two public datasets containing every Transfer and TransferShares record for stETH and wstETH through 2024-11-08. This is the first large-scale empirical characterisation of liquid-staking token circulation. Our approach offers a scalable template for monitoring staking asset flows and provides new, open-access resources to the research community.

2012 ACM Subject Classification Social and professional topics → Information system economics; Computer systems organization → Distributed architectures

Keywords and phrases DeFi, Ethereum, Proof-of-Stake, Liquid Staking, Money Velocity, Inflation

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.9

Supplementary Material *Dataset (Token transfer and velocity data from Ethereum)*: `https://github.com/LucaPennella/money-in-motion-lsts` [13]
archived at `swh:1:dir:632752b04383cb2b9d6c2a3f57103b3a1d0ca213`

Funding This research was supported by the SwissDiamondCoin Foundation. The research has been produced with co-funding from the European Union - Next Generation EU.

1 Introduction

The transition to Proof-of-Stake (PoS) in Ethereum [2] has resulted in both expected and unexpected effects on the network. One notable outcome is the rise of third-party liquid staking providers [17, 12]. Staking in PoS protocols involves locking a specific amount of tokens (the stake) to participate in the consensus process. In Ethereum, stakers lock at least 32 ETH, enabling them to issue attestations in consensus epochs and participate in block proposal lotteries [2, 21]. This stake acts as collateral to ensure honest behaviour; any

© Benjamin Kraner, Luca Pennella, Nicolò Vallarano, and Claudio J. Tessone;
licensed under Creative Commons License CC-BY 4.0

dishonest activity may result in a penalty known as slashing. Additionally, staking requires participants to accept an opportunity cost. During the lock-up period, the staked tokens become illiquid, meaning they cannot be used in transactions or for other purposes. Liquid staking services address this limitation [5].

Lido, the dominant provider of liquid staking services, allows users to convert ETH into *stETH*, a liquid token. Holders of *stETH* continue to earn staking rewards, minus a fee paid to the service provider. This fee compensates providers like Lido, the biggest LST protocol, for managing staking responsibilities, including tasks that, if mishandled, could result in penalties or fund losses [8]. In addition to *stETH*, Lido also provides *wstETH*, a wrapped version of the token designed to maintain a stable ratio relative to the underlying staked ETH. *wstETH* is particularly suited for integration with decentralized finance (DeFi) protocols, as it is ERC-20 compliant and reflects staking rewards through an increasing exchange rate rather than token balance growth.

Liquid staking's rapid adoption has raised concerns about possible unintended impacts on the PoS ecosystem. Critics argue that widespread use of liquid staking could lead to outcomes such as block cartelization, ambiguity in protocol governance, and increased coupling of capital risk with protocol risk [10]. Currently, PoS systems offer limited ways to mitigate the growth of liquid staking, with existing approaches focusing primarily on encouraging voluntary self-regulation by service providers [10].

Liquid staking introduces a novel concept in the realm of digital assets, one that lacks a direct real-world comparison. It effectively allows money to multiply within users' digital wallets, a phenomenon largely absent from traditional financial systems[1]. This unique characteristic raises important questions about the nature and classification of liquid staking tokens (LSTs) such as *stETH* and its wrapped counterpart, *wstETH*.

LSTs can be viewed as debt instruments: the staked tokens resemble the principal of a bond, the staking yield corresponds to interest payments, and the eventual return of staked tokens parallels principal repayment. This perspective aligns with their role as inflation-resistant stores of value. However, LSTs also exhibit unique characteristics, such as their use in DeFi applications as collateral or for yield farming, positioning them closer to programmable money. Additionally, LSTs can be seen as derivative instruments, with their value tied to the underlying staked tokens. This interpretation underscores the complexity of LSTs and the challenges of fitting them into existing financial categories. The future trajectory of Ethereum's PoS ecosystem may depend on how these assets are perceived and used. If LSTs are primarily viewed as inflation-resistant stores of value, they may function like digital bonds. However, their liquidity and utility in transactions and DeFi applications could allow them to evolve into a new form of programmable money, which is secured against inflation from monetary expansion.

Liquid staking has also introduced new risks to the Ethereum ecosystem. For example, the adoption of liquid staking services has raised concerns about centralisation. As of August 2022, Lido controlled a 30.1 % market share of total staked ETH, highlighting the significant influence these services can have on the network. Yet today, as of May 2025, the centralisation is only slightly attenuated, as Lido still has a 27 % market share of total staked ETH.

[1] A useful comparison can be made with so-called *"helicopter money"*, a monetary policy tool in which central banks distribute funds directly to households. However, in most advanced economies, such transfers are not the sole mechanism of money creation, nor are they directly tied to individual wealth. Another relevant analogy is a system of central bank digital currency (CBDC), where money exists solely in interest-bearing accounts managed by the central bank, unlike physical cash, which bears no yield.

Moreover, the introduction of LSTs has increased the overall leverage in the crypto market. By enabling staked assets to be used as collateral, LSTs create new opportunities for yield farming but also introduce risk sources. The *stETH* crisis in 2022 demonstrated how the perceived 1:1 peg between *stETH* and ETH can break under market stress, leading to liquidity issues for overleveraged participants [17]. As the Ethereum ecosystem continues to evolve post-merge, the role and impact of liquid staking will likely remain a critical area of focus. The balance between providing liquidity and maintaining network security, as well as the potential for new financial products and services built on LSTs, will shape the future of Ethereum's PoS system and the broader DeFi landscape.

In this work, we contribute to this debate by offering the first comprehensive analysis of LST usage and circulation dynamics through the lens of money micro velocity [20, 6, 4], a granular measure of monetary activity at the individual account level. This approach enables us to detect patterns of financial behaviour that aggregate velocity metrics overlook, such as the concentration of activity among high net worth actors and the emergence of high-frequency intermediaries in an ostensibly decentralized ecosystem.

Our contributions are fourfold:

- Methodological innovation: We adapt the micro velocity framework to rebasing tokens by reconstructing transfer histories in share denominated units, overcoming limitations in event log availability due to the late introduction of the TransferShares event in the Lido protocol.
- Empirical insights: Using on-chain data from December 2020 to November 2024, we compute and analyze both global and disaggregated micro velocities of *stETH* and *wstETH*, revealing a strong concentration of transactional activity in a small subset of large accounts.
- Comparative analysis: We contrast the behavior of rebasing *stETH* and non-rebasing *wstETH* LSTs, highlighting how ERC-20 compliance and integration with DeFi influence token usage and reutilization patterns.
- Open science and reproducibility: To support transparency and enable future research, we release all tools and datasets used in this study at `https://github.com/LucaPennella/money-in-motion-lsts`. This includes two open-source tools: one for extracting and indexing on-chain event logs, and another for querying historical contract state variables. We also provide two curated datasets: one containing `Transfer` events for *wstETH*, and another with `TransferShares` records for *stETH*. In addition, we publish the raw event logs and contract state variables used in our analysis.

The remainder of the paper is structured as follows. In the *Related Work* section, we discuss prior research most relevant to our study. The *Methods* section introduces the foundations of the micro velocity framework, provides an overview of the Lido platform and its on-chain architecture, the *Data* section details the data collection process, and describes the preprocessing steps used to adapt micro velocity analysis to both *stETH* and *wstETH*. In the *Results* section, we present the global micro velocity trends and their decomposition by user categories, followed by an analysis of balance dynamics and a characterization of the Lido ecosystem. Finally, the *Discussion* section interprets the empirical findings and outlines directions for future research.

2 Related Work

The concept of *micro velocity*, the turnover rate of money measured at the level of individual addresses rather than in aggregate, has recently enriched the economics of digital currencies [20, 6, 4, 7]. Whereas traditional velocity treats all holders symmetrically, the micro perspective exposes behavioural diversity that is especially pronounced on blockchains. Empirical

work shows large cross-agent dispersion: wealthier accounts transact far more frequently and high-velocity *intermediary entities* channel a disproportionate share of flows, hinting at latent centralisation in seemingly decentralised networks [4, 6].

Complementary research documents macro-structural asymmetries. Makarov and Schoar [14] reveal the dominance of a handful of players in Bitcoin, while network studies trace persistent hierarchical patterns and accumulative advantage across multiple chains [9, 3]. Wealth inequality metrics such as the Gini coefficient broadly echo these findings [16]. Governance-oriented work further shows how power concentrates through on-chain accountability mechanisms [15], and similar network tests have begun to quantify decentralisation in lending protocols such as Aave [1].

A third strand examines *liquid staking protocols* (LSPs). Gogol et al. provide the first systematic taxonomy and risk map of liquid-staking designs [12, 11]; Scharnowski [17] analyzes basis spreads in price discovery; and Tzinas & Zindros [18] model the principal–agent tension between delegators and node operators.

Despite extensive work on native token circulation and the design of liquidity staking protocols, the intersection of staking, liquidity provision, and transactional dynamics at the agent level remains underexplored. Researchers have not yet systematically applied micro-velocity frameworks beyond native tokens, while LSP research rarely incorporates address-level behavioral analysis. In this paper, we bridge these gaps by extending the micro-velocity methodology to Lido's rebasing (*stETH*) and wrapped (*wstETH*) tokens, uncovering distinctive patterns of liquidity concentration and transformation unique to liquid staking tokens (LSTs). In addition to velocity metrics, we incorporate address-level balance analysis, providing a deep dive into the temporal behavior of key actors. To support transparency and reproducibility, we also release curated open datasets and modular software tools that enable the community to replicate and extend our results. Together, these contributions offer a comprehensive empirical foundation for the study of liquid staking economies at scale.

3 Methodology

To analyze the circulation dynamics of LSTs, we adopt a micro-velocity framework that measures transactional activity at the level of individual accounts. This method allows us to move beyond aggregate metrics and identify heterogeneous behaviour among different user groups, capturing both active and passive usage patterns.

We first introduce the theoretical basis of micro velocity and how it can be adapted to account-based blockchains. Then, we outline its implementation for rebasing tokens like *stETH*, including the handling of share-based accounting. Finally, we describe how this framework is extended to the wrapped, non-rebasing token *wstETH*, enabling a comparative analysis of LST behavior within the Lido ecosystem.

3.1 Micro Velocity

Consider an ERC-20 token account i endowed with a certain amount of tokens. Following [6] we define the probability distribution of tokens holding times of agent i at time t as $P_i^t(\tau)$:

$$P_i^t(\tau) = \frac{w_i^t(\tau)}{M_i(t)}$$

where:

- $w_i^t(\tau)$ is the total amount of tokens held by i at time t with holding times τ and,
- $M_i(t) = \sum_\tau w_i^t(\tau)$ is the total amount of tokens held by i at time t.

The Micro Velocity of agent i is then defined as:

$$V_i(t) = \sum_{\tau} \frac{1}{\tau} P_i^t(\tau) = \sum_{\tau} \frac{1}{\tau} \frac{w_i^t(\tau)}{M_i(t)} \tag{1}$$

$V_i(t)$ dimension is $block^{-1}$ (i.e., measured per block). From V_i we can derive the total velocity MV of the token as:

$$M(t)V(t) = \sum_i M_i V_i \tag{2}$$

In order to compute $P_i^t(\tau)$ for an account-based blockchain (e.g., Ethereum), we adopt a Last-In-First-Out (LIFO) policy for token spending, where users first spend the most recently received funds; if those are insufficient, older balances with different ages are considered. This approach is economically meaningful, as it naturally distinguishes "liquid" money akin to M0 or M1 in traditional macroeconomics, which sits at the top of the wallet, from "illiquid" money, typically held as longer-term savings or investment. This LIFO assumption aligns with prior empirical work [6], which also tested First-in-First-Out (FIFO) and random mixing policies and found negligible differences in velocity estimates. Intuitively, this reflects typical user behavior: everyday spending tends to use the most recently received or most accessible funds. For the numerical results, we relied on the Python package `MicroVelocityAnalyzer`[2].

3.2 Lido Platform Data Overview

Lido is a liquid staking platform that enables users to stake their ETH in a liquid and fractionalized manner. Through Lido, users can acquire *stETH* tokens by depositing ETH, which Lido stakes on their behalf. In exchange, *stETH* or its wrapped version, *wstETH* tokens are transferred to the user's account, representing the staked ETH. Unlike standard staked ETH, which remains locked as collateral, these tokens retain liquidity, allowing users to transfer them freely. This feature enables users to invest amounts below the 32 ETH required to initiate a validator, without the operational responsibilities typically associated with validator duties.

As the staked ETH accumulates rewards, Lido distributes these rewards to *stETH* holders through a process known as rebasing [12]. During rebasing, the *stETH* balance in each user's account is recalculated daily to reflect the earned rewards, effectively increasing the user's *stETH* balance over time[3]. In contrast, *wstETH* does not rebase. Instead, it reflects staking rewards through a continuously increasing exchange rate relative to ETH. As a result, *wstETH* are fully interoperable with DeFi platforms where predictable token balances and ERC-20 compatibility are essential and can be bridged across multiple blockchains [12].

The data collected for this study includes the transfer records related to both *stETH* and *wstETH* tokens, capturing user transactions and the evolution of staking positions over time. Additionally, we extract the internal state of the *stETH* smart contract, which is necessary to complement transfer information, particularly for accurately interpreting reward distributions, as will be discussed in the following sections.

[2] `https://github.com/fdecollibus/MicroVelocityAnalyzer/tree/parallelised_plus_bilance`

[3] Of consideration for the interested reader: the core Lido repository (current implementation): `https://github.com/lidofinance/lido-dao/tree/master` and a primer on Lido: `https://lido.fi/static/Lido:Ethereum-Liquid-Staking.pdf`.

3.2.1 stETH Shares

The *stETH* contract employs an internal share-based accounting mechanism to track each user's stake in the Lido-controlled ETH pool. When a user acquires *stETH* tokens, they effectively purchase shares in the Lido staking pool, representing the ETH they have staked. The conversion rate between *stETH* and shares is dynamic and depends on the total ETH staked with Lido.

Two functions facilitate the conversion between *stETH* tokens and shares:

- `getSharesByPooledEth`: This function calculates the number of shares corresponding to a specified amount of *stETH*, based on the formula:

$$\text{shares} = \frac{\text{stETH amount} \times \text{total shares}}{\text{total pooled ETH}} \tag{3}$$

- `getPooledEthByShares`: This function determines the amount of *stETH* corresponding to a specific number of shares, following the inverse formula:

$$\text{stETH amount} = \frac{\text{shares} \times \text{total pooled ETH}}{\text{total shares}} \tag{4}$$

These functions ensure that the protocol can dynamically adjust share-to-token ratios, accurately reflecting changes in the total pooled ETH and thus ensuring that each user's stake aligns with the current total pool distribution [4].

3.2.2 Rebasing

As rewards accrue from staked ETH, the *stETH* balance for each user changes to reflect these earned rewards, without affecting the user's underlying shares. This process, known as *rebasing*, recalculates each user's *stETH* balance based on their share of the total pooled ETH. The calculation follows the formula from Equation (4)[5]:

$$\text{balanceOf(account)} = \frac{\text{shares[account]} \times \text{totalPooledEther}}{\text{totalShares}}$$

where:

- **shares**: A mapping of each user's share count, updated with every Ether deposit, representing their fractional ownership in the staking pool.
- **totalShares**: The sum of all shares across accounts, used to proportionally distribute the pooled ETH.
- **totalPooledEther**: The total ETH held by the protocol, defined as the sum of `bufferedBalance`, `beaconBalance`, and `transientBalance`, where:
  - **Buffered balance**: ETH stored in the contract, yet to be deposited.
  - **Transient balance**: ETH submitted to the Deposit contract, pending visibility in the beacon chain state, defined as the difference between `DEPOSITED_VALIDATORS_POSITION` and `BEACON_VALIDATORS_POSITION` measured in ETH.
  - **Beacon balance**: ETH held in validator accounts, reported by oracles and serving as the primary contributor to *stETH* supply.

[4] Source: `https://github.com/lidofinance/lido-dao/blob/master/contracts/0.4.24/StETH.sol`
[5] Source of the formula and example: `https://docs.lido.fi/contracts/lido#rebase`

Rebasing events are triggered by the `handleOracleReport` function, which emits the `TokenRebased` event to reflect updated values for `totalPooledEther` and `totalShares`. It is important to note that rebasing only affects the value of these two variables, which in turn affects the balance calculation of each account in the *stETH* smart contract, but the rebasing doesn't iterate over all the accounts to update their balance, which would prove unfeasible and uneconomical.

Table 1 Summary of recorded events, highlighting that `TransferShares` events began later than `Transfer` events.

Token	Event	Number of Records	First Record Block	Last Record Block
stETH	Transfer	2,792,968	11,480,187	21,145,533
stETH	TransferShares	2,519,615	14,860,275	21,145,533
wstETH	Transfer	1,420,359	11,888,810	21,145,533

3.2.3 Minting Events and Initial Token Distribution

For the scope of the present work, we are not interested in the intricate web of smart contracts that manage the deposit of ETH and minting of *stETH*. What we care about is the trace (if any) that minting (and consequently new *stETH* users' adoption) leaves on the *stETH* token transfer records.

This process is observable through a standard on-chain event pattern: every token minting is recorded as a transfer from Ethereum zero address [6] to the minting user account, i.e. the user who's depositing ETH to buy *stETH*. This is consistent throughout all *stETH* implementations in time.

3.2.4 ETH Redemption and Shares Burning

For the scope of our research, which is computing the micro velocity of *stETH* shares, the burning of shares is not important, as burned shares are virtually motionless, meaning they are not to be transferred anymore, and as such their weight in the velocity computation becomes negligible as time passes. As of the Lido 2.0 implementation, deployed on May 12$^{\text{th}}$, 2023, following the Shappella upgrade, the burning address[7] has been introduced. Because *stETH* token burning became relevant with the implementation of stake withdrawal (as *stETH* tokens then became redeemable as well), it is reasonable to assume that we can ignore burning addresses previously set. Since the burning address only received tokens and never sent any (as expected for such an address), we do not consider it in the velocity calculation.

3.2.5 Lido Events

Events in Solidity serve as a bridge between smart contracts and external applications. When state changes occur on the blockchain, events provide a way to log and track these changes. The event data gets stored in transaction logs alongside the blocks, making them accessible to applications monitoring the blockchain.

[6] Ethereum zero address: `0x0000000000000000000000000000000000000000`
[7] Burning address: `0xD15a672319Cf0352560eE76d9e89eAB0889046D3`

In this study, we focused our attention on two particular events, which are crucial for tracking the flow of *stETH* and share allocations, as well as for understanding changes in pooled ETH over time, which directly impacts user balances and protocol dynamics:

- `Transfer`: This standard ERC-20 event logs token transfers, enabling tracking of *stETH* movements between accounts.
- `TransferShares`: This event logs share transfers specifically, complementing the ERC-20 standard by tracking the underlying share movements not captured by token-only events.

As of the current Lido implementation (version 2.0) these two events are emitted on the Ethereum chain together when a transaction in *stETH* tokens is emitted. This reflects the atomic relation between *stETH* shares and tokens.

3.2.6 Lido Variables

Constant state variables in Solidity serve as immutable values that are determined at compile time and stored directly in the contract's bytecode rather than in storage. This design choice makes them highly gas-efficient since reading these values doesn't require accessing blockchain storage. They're commonly used for fixed values like role identifiers, configuration parameters, or mathematical constants that won't change throughout the contract's lifetime.

In this study, we were interested in recovering the conversion rate between *stETH* shares and tokens, which, as described in Section 3.2.1, required the collection of the following constant state variables of Lido smart contracts:

- `lido.Lido.depositedValidators`
- `lido.Lido.beaconValidators`
- `lido.Lido.beaconBalance`
- `lido.Lido.bufferedEther`
- `lido.StETH.totalShares`

3.2.7 wstETH Mechanics and Tracking

wstETH is the wrapped, non-rebasing version of *stETH*, designed to improve compatibility with decentralized finance (DeFi) protocols. Unlike *stETH*, which periodically updates account balances to reflect accrued staking rewards (rebasing), *wstETH* maintains fixed balances per account. Instead, it encodes staking rewards via an increasing exchange rate relative to *ETH*, updated daily in sync with *stETH*'s rebase events.

This structure makes *wstETH* fully compliant with the ERC-20 standard and particularly suitable for smart contract integration, where deterministic token behavior is essential. As a result, tracking *wstETH* on-chain is straightforward: the token emits standard `Transfer` events, and no share-based accounting is required.

In our analysis, we leverage the complete transfer history of *wstETH* to compute micro velocity without needing to reconstruct internal state variables or account for rebasing. While this limits granularity compared to *stETH*, it reflects realistic usage conditions and supports a comparative evaluation across LST designs. Figure 1 illustrates the proportion of total stETH supply wrapped into wstETH over time.

We now turn to the data sources and processing steps required to apply this framework to the Lido ecosystem.

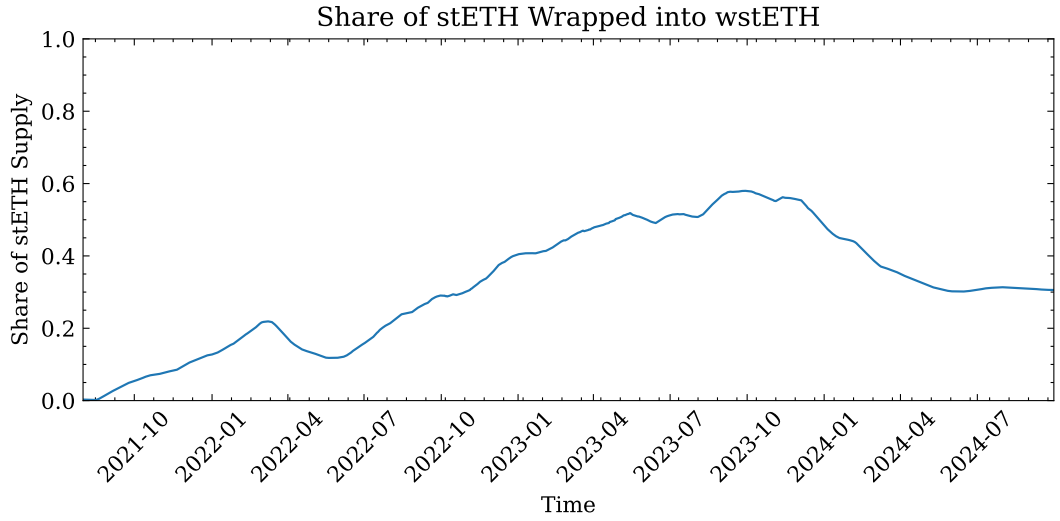


Figure 1 Evolution of the share of stETH wrapped into wstETH over time. The line represents the 30-day moving average of the proportion of total stETH supply held by the wstETH contract. This metric captures the relative demand for tokenized, non-rebasing stETH across the analyzed period.

4 Data

To implement the micro-velocity framework described in the previous section, we constructed a dataset that captures both the transactional and structural characteristics of Lido's liquid staking tokens. This required overcoming several challenges stemming from the rebasing nature of *stETH*, the evolving protocol design, and the need for consistent longitudinal tracking of token behavior at the address level.

In what follows, we describe the data sources, tools, and procedures used to collect, reconstruct, and process the relevant on-chain information. This includes the development of custom software to extract event logs and smart contract state variables, as well as the reconstruction of share-denominated transfer histories necessary for accurate velocity computation.

4.1 Software Tools

Many different tools exist to collect and analyze Ethereum token data, see [22]. Given the rebase nature of *stETH*, the token is not fully ERC-20 compliant, which implies specific tools have to be developed. After a careful analysis of Lido smart contracts, we developed two software tools, the `ethereum-event-tracker` and the `ethereum-variable-tracker`, which were integrated in the data pipeline to produce the data we use in the present work.

The `ethereum-event-tracker` repository[8] was used to retrieve event records from Ethereum logs used in this study. Table 1 provides a summary of the recorded events, including record counts and the range of blocks in which these events were recorded. The `ethereum-variable-tracker` was used to retrieve the state of the constant state variables listed in the previous section for each block of Lido deployment. It is immediate to observe in

[8] `https://gitlab.uzh.ch/bdlt/ethereum-event-tracker`

Table 1, that the number of `TransferShares` events is fewer than the number of `Transfer` events, which does not sound consistent with Lido implementation. This follows the fact that the `TransferShares` event was introduced to the protocol as part of LIP-11[9], an update designed to support the protocol's adjustments in anticipation of the Ethereum Merge[10]. As described in Section 3.2.7, *wstETH* emits standard ERC-20 `Transfer` events, enabling straightforward tracking

4.2 Data Processing

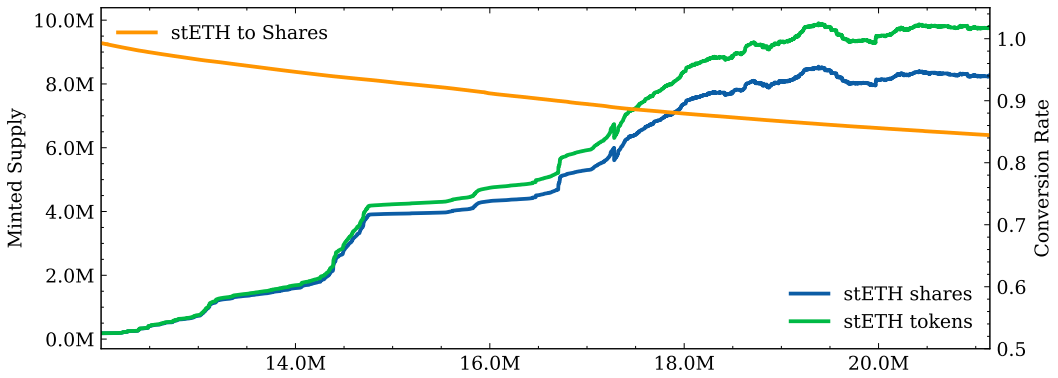


Figure 2 The left axis measures the the total amount of *stETH* tokens (green line) and shares (blue line) in millions. The right axis measures the conversion rate from *stETH* token to shares (orange line). On the x axis time is measured in Ethereum blocks.

In order to compute the micro velocity of *stETH* tokens, we had to account for the token's rebasing nature. An external observer who could only record transfers in *stETH* tokens would not be able to infer the real balances of an account, due to the daily change in account balances denominated in *stETH* tokens. Current methods to compute micro-velocity on Ethereum tokens rely on the collection of token transfers, and especially on the consistency of account balances denominated in the reference token. This makes it fundamental for our research to have access to the full record of *stETH* transfers expressed in shares.

Because of the late introduction of the `TransferShares` event, the collection of share-denominated transfers is incomplete for the early period of *stETH* existence. The same is not true for the `Transfer` events: as ERC-20 compatible events, these events were instantiated from the first deployment of the *stETH* contract. To fill the gap, we recovered the state of the *stETH* smart contract internal constants necessary to compute the tokens to share conversion rate and reconstruct the value in shares of each transfer originally recorded in *stETH* tokens, using the formula described in Section 3.2.1. This was done for each block of Lido deployment. By doing so, we were able to reconstruct a clean database of all *stETH* transfers denominated in shares.

In Figure 2 we can observe the total supply of *stETH* tokens and shares, as recovered from the constant state variable `lido.StETH.totalShares` from the *stETH* smart contract. As the supply grows, the conversion rate diminishes, meaning that the value of a single share is increasing, as the total Ethereum pooled in Lido validators increases and, as such, the

[9] `https://github.com/lidofinance/lido-improvement-proposals/blob/develop/LIPS/lip-11.md`
[10] `https://ethereum.org/en/roadmap/merge/`

rewards they bear. While this result is not novel per se, it stands to testify to the consistency of the data collection methodology. We plot the supply alongside the *stETH* to Shares conversion rate, which is used to convert token transfers into shares transfers and complete the dataset.

We also compute micro velocity separately for *wstETH*, the non-rebasing and ERC-20 compliant counterpart. Unlike *stETH*, *wstETH* maintains fixed balances and encodes staking rewards via an increasing exchange rate relative to ETH, thereby eliminating the need for share-based accounting. As such, the standard set of `Transfer` events is sufficient for computing micro velocity. While *wstETH* does not offer the granularity of share-level tracking, its simplicity and compatibility with DeFi protocols make it an important asset for comparative analysis. The inclusion of *wstETH* velocity allows us to capture a broader spectrum of user behaviour across Lido's liquid staking products.

5 Results

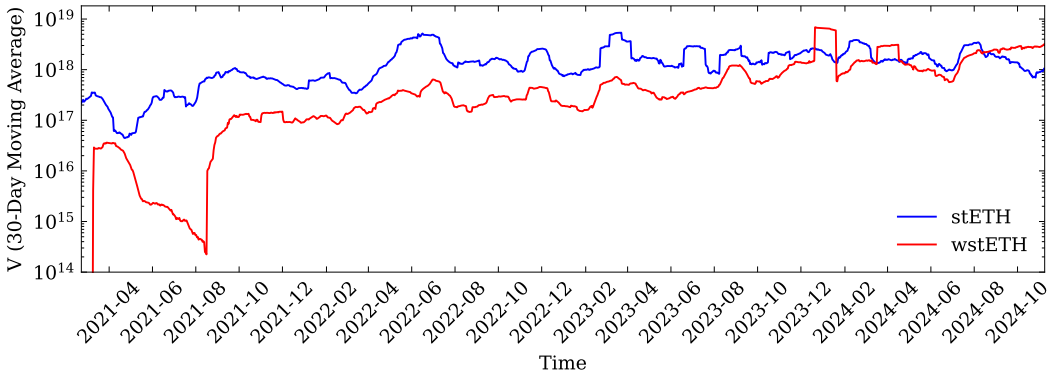


Figure 3 The global velocity $V(t)$ of *stETH* and *wstETH* tokens (see Equation (2), the dimension is $block^{-1}$, averaged weekly.)

Following the methodology described in Section 3.1, we calculate the individual *stETH* and *wstETH* velocities of all token accounts during the period under consideration, from block 11,480,187 (December 18, 2020) to block 21,145,533 (November 18, 2024). Using Equation (2), we derive the global velocity $V(t)$ for both assets, computed as the aggregate of individual micro velocities. The results are depicted in Figure 3, revealing three key features: consistently high values, a two-phase dynamic, and several distinct spikes.

By comparing the velocity we observe to the velocities reported on Proof-of-Work (PoW) and ERC-20 tokens in the literature [4, 6], we find that the global *stETH* and *wstETH* velocity is strikingly large. We find that both tokens exhibit significantly higher turnover. From a usage standpoint, we interpret this as evidence of Lido's success in achieving its mission: facilitating economic activity by unlocking the financial potential of staked tokens while maintaining Ethereum's Proof-of-Stake consensus and system security. Additionally, as interest-bearing tokens, *stETH* and *wstETH* provide an attractive alternative to ETH or other tokens as a store of value. They not only hedge against the effects of monetary expansion, thus appreciating in nominal terms, but also enable economic transactions.

From Figure 3, we observe a two-phase trajectory: an initial growth phase followed by a mature phase. The growth phase occurred between the end of 2020 and the first half of 2022, during which velocity increased, reflecting greater adoption and the maturation of Lido's

Table 2 Summary of address categories by received *stETH*. The *Thresholds* column defines the lower and upper bounds used to assign accounts to categories.

Category	Thresholds (*stETH*)	Total Received (*stETH*)	Count
Whale	$x \geq 10000$	141703559.627	1011
Orca	$3000 \leq x < 10000$	6981034.142	1278
Dolphin	$1000 \leq x < 3000$	4074250.128	2378
Fish	$100 \leq x < 1000$	4547189.463	14441
Shrimp	$10 \leq x < 100$	1636947.564	49982
Krill	$1 \leq x < 10$	416861.963	113749
Plankton	$x < 1$	68469.071	307188

economic role. Notably, Lido's merge-ready protocol upgrade took place in May 2022[11], coinciding with the Ethereum Merge, and introduced a more sophisticated version of the protocol. While this observation aligns with the natural growth in attention surrounding Lido and Ethereum's transition from PoW to PoS, it also validates our methodology, demonstrating its ability to capture significant on-chain economic events.

A significant spike in *stETH* velocity is observed in March 2023. This aligns with user anticipation of the Shappella upgrade (April 2023), which introduced staking reward withdrawals, and the release of Lido 2.0 (June 2023). These events likely stimulated economic activity by unlocking previously illiquid capital.

Despite being a newer asset with a smaller user base, *wstETH* shows a rapidly increasing velocity, converging with *stETH* from early 2023 onward. This is particularly notable given the lower number of total addresses involved (Table 3). We interpret this as a sign of higher per-token turnover, likely reflecting *wstETH*'s enhanced suitability for DeFi applications due to its fixed-balance, ERC-20-compatible design. The asset's structure encourages use in smart contracts and collateralized protocols, leading to intensive re-utilization. Notably, the global velocity of *wstETH* increased above that of its base token *stETH*. If this trend continues, differences in velocity may indicate a preferred role for staked Ether: *wstETH* functioning as an inflation-resistant form of smart money, and *stETH* serving more as a passive savings instrument.

To further investigate the source of the large velocity, we utilized the micro foundation of velocity and decomposed the global metric by contributor categories. Following the user taxonomy introduced by Lido[12], we classified all *stETH* accounts into seven categories based on the amount of *stETH* received during the study period (December 2020 to November 2024). Table 2 and Table 3 summarize these categories and their statistics for *stETH* and *wstETH*.

For *stETH*, whale accounts, that received at least 10,000 tokens, represent just 0.25 % of all addresses (1,011 out of more than 480,000) yet collectively received approximately 141.7 million *stETH*, a dominant share of the total distribution. This confirms a well-documented trend in blockchain economies: a small number of large accounts command a disproportionate amount of monetary flow [9, 19, 3]. Mid-tier categories (e.g., Orcas and Dolphins) also hold substantial quantities, while the vast majority of accounts (Plankton, Krill, Shrimp) received comparatively minor volumes.

[11] `https://research.lido.fi/t/announcement-merge-ready-protocol-service-pack/2184`
[12] `https://blog.lido.fi/analysis-of-steth-user-behaviour-patterns/`

Table 3 Summary of address categories by received *wstETH*. The *Thresholds* column defines the lower and upper bounds used to assign accounts to categories.

Category	Thresholds (*wstETH*)	Total Received (*wstETH*)	Count
Whale	$x \geq 10000$	149285851.223	780
Orca	$3000 \leq x < 10000$	4618320.295	850
Dolphin	$1000 \leq x < 3000$	2457660.110	1401
Fish	$100 \leq x < 1000$	2084328.725	6224
Shrimp	$10 \leq x < 100$	413353.129	11285
Krill	$1 \leq x < 10$	54465.652	14063
Plankton	$x < 1$	8030.266	56518

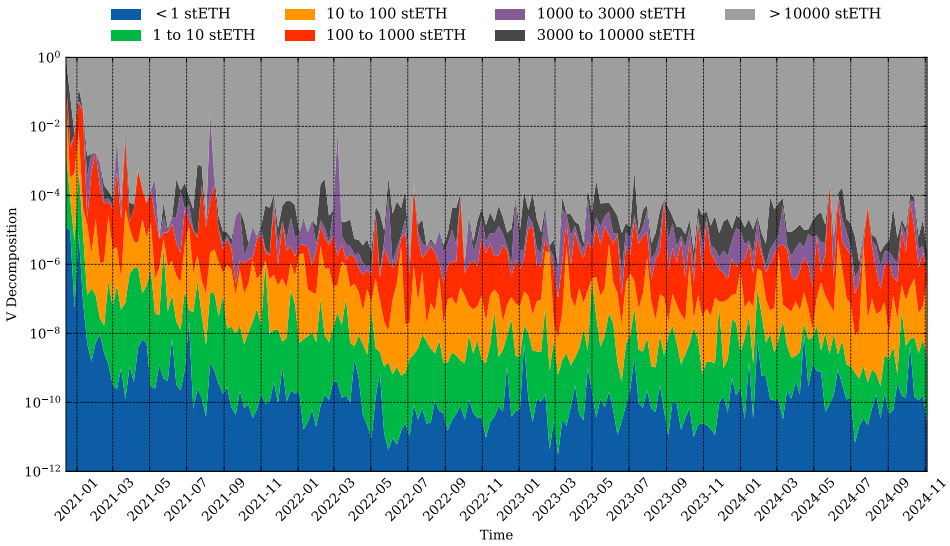
A similar concentration is observed for *wstETH*. Although whales are even fewer in number (780 addresses), they received nearly 149.3 million tokens, again constituting the majority of the total supply. Notably, the distribution for *wstETH* is even more top-heavy: small holders (Plankton, Krill, Shrimp) are fewer in number than in the *stETH* dataset and received a smaller aggregate amount (~476k tokens vs. over 2.1 million for *stETH*). This suggests a more targeted adoption of *wstETH* among sophisticated or institutional users, possibly driven by its composability and compatibility with DeFi infrastructure.

Taken together, the distributions reported in Table 2 and Table 3 indicate that the usage of liquid staking tokens exhibits a marked degree of asymmetry. In both *stETH* and *wstETH*, a small number of high volume addresses, classified as whales, receive the majority of tokens, while the vast majority of accounts hold relatively modest amounts. This pattern suggests the coexistence of two distinct user profiles: on one hand, large entities that actively transact substantial volumes of LSTs, likely for integration within DeFi protocols; on the other, a broader base of smaller holders whose participation is limited to token acquisition, likely to passively benefit from staking rewards.

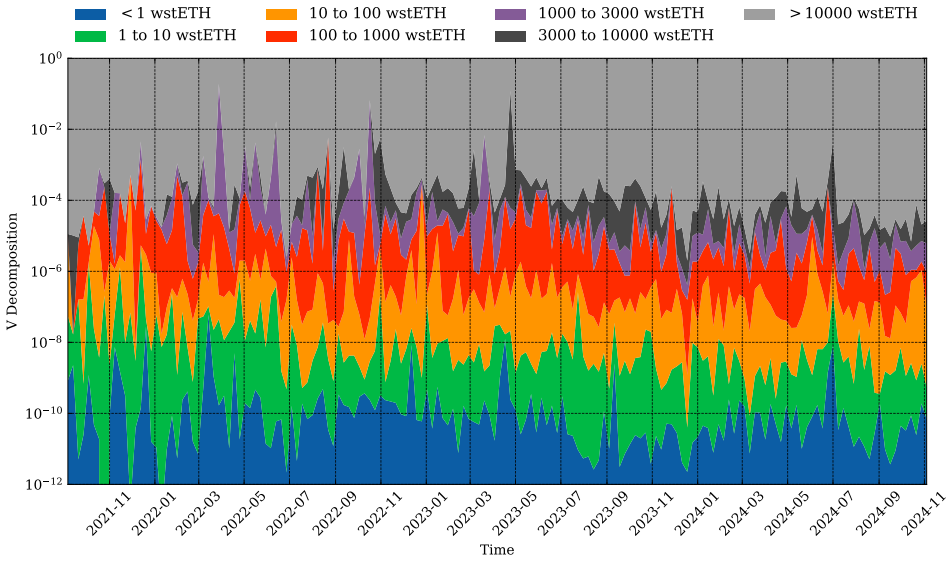
When examining the decomposition of micro velocity by account category for both *stETH* and *wstETH* (Figure 4a and Figure 4b), a consistent concentration pattern is observed. In each case, whale accounts representing less than 1 % of the total account for the overwhelming majority of total velocity. For *stETH*, whales contribute approximately 99 % of the aggregate velocity over the entire observation period. A comparable concentration is found for *wstETH*, where high-volume accounts similarly dominate the transactional dynamics.

Outside the whale category, the remaining velocity contributions are substantially lower and display a more uniform distribution across mid and small volume groups. This stratification is more clearly visible in the log-scale representation of velocity shares, which reveals the limited role of lower-tier accounts in driving token circulation. These findings suggest that, while liquid staking tokens are widely distributed in terms of address count, the effective circulation is predominantly sustained by a narrow subset of large participants.

The observed concentration of velocity among whale accounts is likely influenced by the integration of liquid staking tokens into decentralized finance applications. Large accounts may correspond to actors such as DeFi protocols, liquidity providers, or institutional agents that engage in high-frequency and high-volume transactions, thereby contributing disproportionately to total velocity. This mechanism appears consistent across both *stETH* and *wstETH*, although the fixed-balance structure and DeFi-oriented design of *wstETH* may further incentivize its use in automated financial operations. This analysis allows us to identify two distinct user typologies for Lido (and, by extension, for LSTs):



(a) Micro velocity shares decomposition for *stETH*, by categories defined in Table 2. Expressed in log scale.



(b) Micro velocity shares decomposition for *wstETH*, by categories defined in Table 2. Expressed in log scale.

Figure 4 Comparison of micro velocity decomposition across address categories for *stETH* and *wstETH*. Both distributions are highly concentrated in Whale addresses.

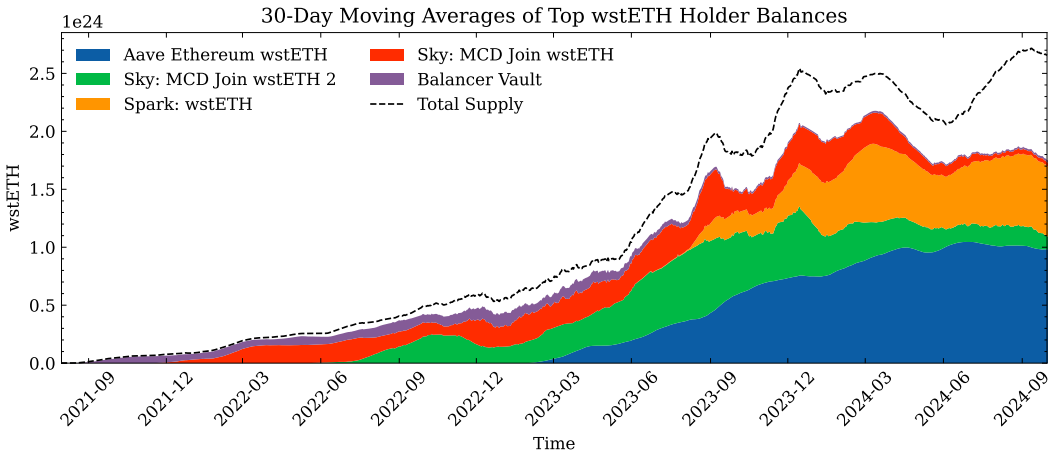


Figure 5 30-day moving averages of *wstETH* balances for the top five holder addresses. These include major DeFi protocols such as AAVE, Spark, Balancer, and SkyMoney.

1. A limited set of high-capacity accounts that receive and transact large volumes of liquid staking tokens, likely reflecting their integration into decentralized finance infrastructures.
2. A broad base of smaller accounts that primarily hold LSTs in a passive manner, likely to access staking rewards without running a validator, and that exhibit low transactional activity.

Table 4 Identified Ethereum Addresses and Associated Protocol Entities.

Address	Protocol Entities
`0x0b925ed163218f6662a35e0f0371ac234f9e9371`	Aave Ethereum wstETH
`0x12b54025c112aa61face2cdb7118740875a566e9`	Spark: wstETH
`0x248ccbf4864221fc0e840f29bb042ad5bfc89b5c`	SkyMoney: MCD Join wstETH 2
`0x10cd5fbe1b404b7e19ef964b63939907bdaf42e2`	SkyMoney: MCD Join wstETH
`0xba12222222228d8ba445958a75a0704d566bf2c8`	Balancer Vault

Beyond aggregate measures of micro-velocity, we further explore the behaviour of the largest token holders through account-level balance dynamics. Figure 5 plots the 30-day moving averages of *wstETH* balances for the five largest non-exchange holders, prominent DeFi actors such as Aave, Balancer, Spark and SkyMoney.

Table 4 lists these contracts. The Balancer V2 Vault is the upgradeable contract that escrow-holds every Balancer pool's assets and executes swaps, joins, exits and external asset-management hooks; whenever a pool contains *wstETH*, the Vault's inventory rises accordingly, explaining its position as the single largest holder outside centralised exchanges. The Aave V3 *wstETH* aToken and the homologous Spark Lend receipt token serve as interest-bearing ERC-20 wrappers. SkyMoney accepts *wstETH* as collateral through two join adapters[13], whose dual-adapter architecture enables differentiated risk parameters. Collectively, these five contracts account for the overwhelming majority of *wstETH* held outside centralised exchanges, illustrating the token's deep integration into liquidity-provision, yield-generation and collateral-management workflows across the contemporary DeFi stack.

[13] `https://developers.sky.money/protocol/core/join/`

The time series reveals stable yet distinct trajectories, with entities like AAVE and Spark consistently maintaining large holdings, suggesting long-term integration of *wstETH* into lending and collateral strategies.

It is noteworthy that an examination of the top five accounts already reveals a significant concentration of wstETH holdings, consistent with the broader concentration patterns observed in the velocity decomposition.

6 Discussion & Conclusion

In this work, we provided a detailed analysis of the *stETH* and *wstETH* token transfer protocols, offering novel insights into the behaviour of Lido users by computing and decomposing the micro velocity of both liquid staking tokens (LSTs). By leveraging on-chain data, we captured the transactional activity of accounts across the entire lifespan of these assets, revealing both structural patterns and temporal dynamics.

Our findings show that both *stETH* and *wstETH* exhibit remarkably high global velocity, especially compared to previously studied Proof-of-Work assets and standard ERC-20 tokens. This supports the hypothesis that LSTs effectively unlock the liquidity of staked assets without undermining the security guarantees of Ethereum's Proof-of-Stake consensus. As a result, these tokens circulate actively within the ecosystem while simultaneously accruing staking rewards. The increasing global velocity of *wstETH* may serve as a proxy for its depth of integration and functional usability within DeFi applications, especially when compared to the velocity of its base token.

We also observe a growing proportion of *stETH* that is wrapped into *wstETH*, underscoring a demand for non-rebasing tokens, likely driven by their ERC-20 compliance and deterministic behaviour in smart contract environments.

A key outcome of our decomposition is the identification of a highly asymmetric distribution of transactional activity. A small subset of accounts, classified as whales, is responsible for the vast majority of observed velocity in both *stETH* and *wstETH*, despite constituting less than 1 % of all users. These high-wealth actors likely represent institutional agents, DeFi protocols, and liquidity providers who integrate LSTs into automated or high-frequency financial strategies. Conversely, a large number of smaller accounts appear to use LSTs in a passive manner, primarily as a means of earning staking rewards without running a validator.

Further insights emerge when focusing on top holders' balance trajectories (Figure 5). We observe persistent large-scale holdings by DeFi-native actors where a strong concentration is already evident among the top five addresses. This suggests a deliberate integration of the wrapped token into lending, collateralization, and composable DeFi protocols.

Taken together, these findings illustrate the emergence of a dual user base: on one hand, high-frequency DeFi participants actively leveraging LSTs within financial protocols; on the other, a broad population of low-activity holders passively engaging with staking infrastructure. This duality underscores the financial maturity and evolving specialization of LSTs in Ethereum's ecosystem.

To place our findings in proper context, the same micro-velocity lens must now be applied to other LSTs on Ethereum (e.g., *cbETH*, *rETH*) and to liquid-staking assets on external PoS chains. A cross-token panel will calibrate what "high" or "low" velocity really means and let us classify assets by behavioural archetype: passive store-of-value, actively rehypothecated collateral, bridge-hopping fuel, and so forth. Because micro-velocity reacts at the transaction level, it can surface shifts in user behaviour long before coarse indicators such as raw volume or TVL respond, making it a candidate early-warning metric for researchers, protocol teams, and on-chain risk monitors.

For future work, it would be worthwhile to expand the analysis to *yield farming* behaviours, whereby LSTs are recursively used as collateral to borrow and reinvest in additional staking positions. Such strategies, implemented through yield optimizers and money market protocols, may amplify both token velocity and systemic leverage. Identifying these patterns on-chain could help quantify endogenous risk and clarify their impact on the monetary dynamics of liquid staking ecosystems.

These results ultimately serve to reinforce the pivotal role of Lido within the context of Ethereum's staking and DeFi landscape. By providing liquid staking infrastructure that balances accessibility, composability, and economic utility, Lido has not only become the dominant LST provider but also a structural pillar of post-Merge Ethereum. It is imperative to comprehend the dynamics that are facilitated by metrics such as micro velocity in order to grasp the evolving monetary and systemic properties of Proof-of-Stake ecosystems.

Data Availability

The datasets and tools used in this study are publicly available at `https://github.com/LucaPennella/money-in-motion-lsts`

References

1 Ziqiao Ao, Lin William Cong, Gergely Horvath, and Luyao Zhang. Is decentralized finance actually decentralized? a social network analysis of the aave protocol on the ethereum blockchain. *CoRR*, abs/2206.08401, 2022. doi:10.48550/arXiv.2206.08401.

2 Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X. Zhang. Combining GHOST and casper. *CoRR*, abs/2003.03052, 2020. doi:10.48550/arXiv.2003.03052.

3 Carlo Campajola, Raffaele Cristodaro, Francesco Maria De Collibus, Tao Yan, Nicolò Vallarano, and Claudio J. Tessone. The evolution of centralisation on cryptocurrency platforms. *CoRR*, abs/2206.05081, 2022. doi:10.48550/arXiv.2206.05081.

4 Carlo Campajola, Marco D'Errico, and Claudio J. Tessone. Microvelocity: rethinking the velocity of money for digital currencies, 2023. arXiv:2201.13416.

5 Tarun Chitra and Alex Evans. Why stake when you can borrow? *CoRR*, abs/2006.11156, 2020. doi:10.48550/arXiv.2006.11156.

6 Francesco Maria De Collibus. *The ethereum ecosystem from a transaction network perspective*. PhD thesis, University of Zurich, Zürich, April 2024. doi:10.5167/uzh-260608.

7 Francesco Maria De Collibus, Carlo Campajola, and Claudio J. Tessone. The microvelocity of money in ethereum. *EPJ Data Sci.*, 14(1):11, 2025. doi:10.1140/epjds/s13688-024-00518-6.

8 Mikel Cortes-Goicoechea, Tarun Mohandas-Daryanani, Jose Luis Muñoz-Tapia, and Leonardo Bautista-Gomez. Autopsy of ethereum's post-merge reward system. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2023, Dubai, United Arab Emirates, May 1-5, 2023*, pages 1–9. IEEE, 2023. doi:10.1109/ICBC56567.2023.10174942.

9 Francesco Maria De Collibus, Carlo Campajola, Guido Caldarelli, and Claudio J Tessone. Patterns and centralisation in ethereum-based token transaction networks. *Frontiers in Physics*, 12, 2024. doi:10.3389/fphy.2024.1305167.

10 Ethereum Foundation. The risks of lsd. https://notes.ethereum.org/@djrtwo/risks-of-lsd, 2022. Accessed: 2025-08-06. URL: https://notes.ethereum.org/@djrtwo/risks-of-lsd.

11 Krzysztof Gogol, Benjamin Kraner, Malte Schlosser, Tao Yan, Claudio J. Tessone, and Burkhard Stiller. Empirical and theoretical analysis of liquid staking protocols. *CoRR*, abs/2401.16353, 2024. doi:10.48550/arXiv.2401.16353.

12 Krzysztof Gogol, Yaron Velner, Benjamin Kraner, and Claudio J. Tessone. Sok: Liquid staking tokens (lsts) and emerging trends in restaking, 2024. doi:10.48550/arXiv.2404.00644.

13 Benjamin Kraner, Luca Pennella, Nicolò Vallarano, and Claudio J. Tessone. LucaPennella/money-in-motion-lsts. Dataset, Swiss Diamond Coin Foundation, European Union - Next Generation EU (visited on 2025-09-17). URL: https://github.com/LucaPennella/money-in-motion-lsts, doi:10.4230/artifacts.24344.

14 Igor Makarov and Antoinette Schoar. Blockchain analysis of the bitcoin market. *SSRN Electronic Journal*, October 2021. doi:10.2139/ssrn.3942181.

15 Kelsie Nabben and Primavera De Filippi. Accountability protocols? on-chain dynamics in blockchain governance. *Internet Policy Rev.*, 13(4):1–22, 2024. doi:10.14763/2024.4.1807.

16 Ashish Rajendra Sai, Jim Buckley, and Andrew Le Gear. Characterizing wealth inequality in cryptocurrencies. *Frontiers Blockchain*, 4:730122, 2021. doi:10.3389/fbloc.2021.730122.

17 Stefan Scharnowski and Hossein Jahanshahloo. The economics of liquid staking derivatives: Basis determinants and price discovery. *Journal of Futures Markets*, 45(2):91–117, 2025. doi:10.1002/fut.22556.

18 Apostolos Tzinas and Dionysis Zindros. The principal-agent problem in liquid staking. In Aleksander Essex, Shin'ichiro Matsuo, Oksana Kulyk, Lewis Gudgeon, Ariah Klages-Mundt, Daniel Perez, Sam Werner, Andrea Bracciali, and Geoff Goodell, editors, *Financial Cryptography and Data Security. FC 2023 International Workshops - Voting, CoDecFin, DeFi, WTSC, Bol, Brač, Croatia, May 5, 2023, Revised Selected Papers*, volume 13953 of *Lecture Notes in Computer Science*, pages 456–469, Cham, 2023. Springer. doi:10.1007/978-3-031-48806-1_29.

19 Nicolò Vallarano, Claudio J. Tessone, and Tiziano Squartini. Bitcoin transaction networks: An overview of recent results. *Frontiers in Physics*, 8:286, 2020. Published: December 3, 2020. Section: Physics of Networks. doi:10.3389/fphy.2020.00286.

20 Yougui Wang, Ning Ding, and Li Zhang. The circulation of money and holding time distribution. *Physica A: Statistical Mechanics and its Applications*, 324(3):665–677, 2003. doi:10.1016/S0378-4371(03)00074-8.

21 Tao Yan, Shengnan Li, Benjamin Kraner, Luyao Zhang, and Claudio J. Tessone. A data engineering framework for ethereum beacon chain rewards: From data collection to decentralization metrics. *Scientific Data*, 12(1):519, 2025. doi:10.1038/s41597-025-04623-7.

22 Peilin Zheng, Zibin Zheng, Jiajing Wu, and Hong-Ning Dai. Xblock-eth: Extracting and exploring blockchain data from ethereum. *IEEE Open J. Comput. Soc.*, 1:95–106, 2020. doi:10.1109/OJCS.2020.2990458.

Do Inflation Expectations Drive Cryptocurrency Investments?

Will Cong
Cornell University, Ithaca, NY, USA

Pulak Ghosh
Indian Institute of Management, Bangalore, India

Jiasun Li[1]
George Mason University, Fairfax, VA, USA

Qihong Ruan
Cornell University, Ithaca, NY, USA

Artem Streltsov
SUNY Buffalo, Buffalo, NY, USA

Abstract

This extended abstract summarizes the essence of [1], which uses proprietary data from a predominant cryptocurrency exchange in India and the country's Household Inflation Expectations Survey to document a significantly positive association between inflation expectations and individual cryptocurrency purchases. The authors find that the effect is concentrated in Bitcoin (BTC) and Tether (USDT) trading, and is robust after controlling for speculative demand captured by surveys of investors' expected cryptocurrency returns. They also find that higher inflation expectations are associated with more new cryptocurrency investors. These results have causal interpretations as confirmed by instrumental variables. The findings of [1] provide direct evidence that households already adopt cryptocurrencies for inflation hedging, which in turn rationalizes their high adoption in developing countries without a globally dominant currency.

2012 ACM Subject Classification Social and professional topics → User characteristics

Keywords and phrases Bitcoin, Cryptocurrency, Household Finance, Inflation, Stablecoin

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.10

Category Extended Abstract

Related Version
Full Version: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4549475

Funding *Jiasun Li*: NSF 2337338

1 Motivation

Understanding why people buy and hold cryptocurrencies is a central topic in both blockchain and finance research, as many tokens lack fiat backing or cash flow yet still command substantial value. While explanations abound – from speculative trading to utility in digital ecosystems – inflation hedging is among the most frequently proposed. This is especially true for cryptocurrencies with capped supplies or pegged values, which in principle protect purchasing power against local currency depreciation. The authors of [1] provide direct, systematic, and real-world evidence for this argument in a large emerging market context.

[1] Please send correspondence to Li; Authors are listed alphabetically.

© Will Cong, Pulak Ghosh, Jiasun Li, Qihong Ruan, and Artem Streltsov;
licensed under Creative Commons License CC-BY 4.0

2 Setting and Data

India offers an ideal testing ground: historically high inflation, limited access to foreign currencies due to strict capital control, and high rates of cryptocurrency adoption. The empirical analysis links two sources:

- **Exchange transactions:** pseudonymized trades from India's largest cryptocurrency exchange, which contain transaction and investor demographics details.
- **Inflation expectations:** Repeated cross-sections from the Inflation Expectations Survey of Households held by the Reserve Bank of India (India's central bank), with projections for general price changes three and twelve months ahead.

The study matches exchange activity to local survey measures via postal code and survey period.

3 Empirical Approach

The authors of [1] assess whether high local inflation expectations precede more net cryptocurrency purchases, as well as more new investors entering the exchange. To establish a causal relationship, they also use contemporaneous perceived inflation as an instrument variable.

4 Key Findings

Several consistent patterns emerge:

1. **Positive association:** Higher expected inflation is followed by larger net purchases of cryptocurrencies.
2. **Asset concentration:** The effect is concentrated in Bitcoin and Tether; other tokens show little response.
3. **New participation:** Local rises in expected inflation is followed by more entry of first-time investors onto the exchange.
4. **Robustness:** Results hold when controlling for speculative motives (measured by surveyed cryptocurrency return expectations), and disappear for trades not tied to the local currency (Indian Rupee).

5 Contributions and Implications

This work delivers rare micro-level evidence connecting inflation expectations to actual cryptocurrency investment. The findings:

- Demonstrate that some cryptocurrencies are already perceived and used as inflation hedges.
- Highlight differences across asset types, geographies, and investor demographics.
- Provide direct evidence to suggest that macroeconomic factors may shape cryptocurrency adoption.

For policymakers, the results underline the potential for cryptocurrencies to absorb household demand for value preservation in inflation-prone economies. For researchers, they open avenues for studying macro-financial drivers of digital asset markets beyond speculative trading.

6 Conclusion

The authors of [1] show that inflation expectations influence both the scale and incidence of cryptocurrency investment in India, particularly in assets aligned with hedging narratives. This result underscores the role of macroeconomic perceptions in driving adoption and hints at potentially similar patterns in other emerging markets facing persistent inflationary pressures.

References

1 Lin William Cong, Pulak Ghosh, Jiasun Li, Qihong Ruan, and Artem Streltsov. Inflation expectation and cryptocurrency investment. Technical report, National Bureau of Economic Research, 2024. URL: https://www.nber.org/papers/w32945.

Cryptocurrency Exchange Listings

Jiasun Li[1]
George Mason University, Fairfax, VA, USA

Mei Luo
Tsinghua University, Beijing, China

Muzhi Wang
Central University of Finance and Economics, Beijing, China

Zhe Wei
Shantou University, China

Abstract

This extended abstract summarizes the essence of [1], which studies the listing decisions as well as listing performances on cryptocurrency exchanges, and investigates potential strategic incentives and their interactions with regulatory exposures. The authors first present a case study comparing cryptocurrency listings on the largest and more regulated U.S. exchange, Coinbase, and the largest but less regulated global exchange, Binance. They find that: Regarding listing performance, while cryptocurrency listings on both exchanges see significantly positive short-term returns, the more regulated Coinbase sees significantly higher listing returns than the less regulated Binance; Regarding listing choices, while both exchanges tend to list cryptocurrencies with more GitHub development activities, conflicts of interests arise when exchanges list cryptocurrencies that their venture capital arms have previously invested in. Specifically, the authors find the less regulated Binance being more likely to list its self-invested coins with inferior fundamentals, and the apparent agency friction does not seem to be corrected by market forces. To obtain external validity of the lessons learned from the top two exchanges, the authors further construct an exchange regulation index on a larger sample of 80 qualified exchanges, and confirm the relation between stricter exchange regulations and higher short-term listing returns, controlling for cryptocurrency and exchange attributes.

2012 ACM Subject Classification Social and professional topics → Computing / technology policy

Keywords and phrases cryptocurrency, exchanges, listing, regulation

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.11

Category Extended Abstract

Related Version
Full Version: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4715718

1 Motivation

Listing a cryptocurrency on a major exchange can dramatically expand its market reach, improve liquidity, and influence valuation. The process is not only a technical task but also a strategic decision to the exchange, often made in the context of different regulatory regimes. While some exchanges face intensive scrutiny closer to traditional securities markets, others operate under lighter oversight, potentially allowing different stratetic incentives to shape listing choices. The authors of [1] explore these contrasts to understand how regulation and internal incentives jointly influence both what kind of cryptocurrencies are listed and how those listings perform.

[1] Please send correspondence to Li; Authors are listed alphabetically.

© Jiasun Li, Mei Luo, Muzhi Wang, and Zhe Wei;
licensed under Creative Commons License CC-BY 4.0

2 Comparative Case: Coinbase and Binance

The study first focuses on two global leaders: Coinbase, the largest U.S.-regulated exchange, and Binance, the largest exchange globally but operating outside of U.S. and arguably less regulated. The authors of [1] document that both exchanges typically list cryptocurrencies with more active development activities, measured by public GitHub records. However, in cases where an exchange's own venture capital arm holds prior investments, patterns diverge:

- On Binance, among listed tokens, the fundamentals of those that have received previous investment by the exchange's own venture arms tend to be weaker.
- On the more heavily-regulated Coinbase, such patterns do not appear.

In both venues, short-run abnormal returns following listing are positive, but they are significantly larger on Coinbase, suggesting either stronger screening or greater investor confidence in regulated platforms.

3 Extending to a Larger Sample

To assess whether patterns extend beyond the top two exchanges, the authors of [1] construct an *exchange regulation index* based on (1) the publicly listed status of the exchanges themselves, 2) financial licenses granted by local governments, 3) the general environment (such as investor protection, legal origin, and financial openness) of the country where the exchange is headquartered, and 4) status of non-BVI incorporation, given that BVI-areas (e.g. British Virgin Islands, Seychelles, Republic of Lithuania, Gibraltar, Dubai, South America, and The United Arab Emirates) allow entities with flexible organizational structure and limited financial reporting requirements. Applying this index to 80 exchanges, the authors confirm that listings on more regulated exchanges tend to yield higher initial returns.

4 Implications

The findings suggest that regulatory designs may influence market quality in the cryptocurrency space, not only through investor protection but also by shaping the incentives of exchanges and the performance of token listings. Where oversight is weaker, exchanges may prioritize internal investment returns over asset quality, potentially undermining long-run market integrity. Conversely, stronger regulation appears to enhance screening and boost investor trust, leading to better short-term performance at listing.

5 Concluding Remarks

By combining micro-level event studies with cross-exchange analysis, the authors of [1] provide new insights into the interplay between regulation, exchange listing decisions, and asset listing performance. The results have implications for policymakers considering exchange oversight frameworks and for market participants evaluating where to list or trade digital assets.

References

1 Jiasun Li, Mei Luo, Muzhi Wang, and Zhe Wei. Cryptocurrency exchange listings. Technical report, George Mason University, 2025. doi:10.2139/ssrn.4715718.

How Much Public Randomness Do Modern Consensus Protocols Need?

Joseph Bonneau
New York University, NY, USA

Benedikt Bünz
New York University, NY, USA

Miranda Christ
Columbia University, New York, NY, USA

Yuval Efron
Columbia University, New York, NY, USA

Abstract

Modern blockchain-based consensus protocols aim for efficiency (i.e., low communication and round complexity) while maintaining security against adaptive adversaries. These goals are usually achieved using a public randomness beacon to select roles for each participant. We examine to what extent this randomness is necessary. Specifically, we provide tight bounds on the amount of entropy a Byzantine Agreement protocol must consume from a beacon in order to enjoy efficiency and adaptive security. We first establish that no consensus protocol can simultaneously be efficient, be adaptively secure, and use $O(\log n)$ bits of beacon entropy. We then show this bound is tight and, in fact, a trilemma by presenting three consensus protocols that achieve any two of these three properties.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Consensus, Randomness Beacon

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.12

1 Introduction

Consensus is a cornerstone of distributed computing, with research spanning over four decades [33, 28]. In the consensus problem, n players (or nodes) with respective inputs must engage in communication in order to reach *agreement* on a value. The challenge stems from the presence of at most f *corrupted* (eq. *Byzantine*) players, which may deviate from any given protocol in arbitrary ways.

The consensus problem has been considered under a myriad of constraints and allowances across various axes, including network synchrony, setup assumptions, the use of randomness by honest players, and the capabilities of the adversary (Garay and Kiayias [20] provide a detailed survey). With the renewed interest in this problem due to its foundational role in blockchains, two properties of particular importance from a practical point of view have been *efficiency* and *adaptive security*.

1. **Efficiency.** We consider an efficiency notion motivated by modern blockchain-related consensus protocols. We say that a protocol has *low communication* (or in other words, is *laconic*) if in every round few of the players (i.e., $o(n) << n$) send messages. A protocol has *low latency* if it requires at most $o(n)$ rounds. We refer to a protocol as *efficient* if it has both low communication and low latency. Our efficiency notion is essential for consensus protocols to scale to thousands of players and finalize transactions quickly. Indeed, most modern blockchain systems (e.g., Bitcoin and Ethereum) are efficient in this sense.

© Joseph Bonneau, Benedikt Bünz, Miranda Christ, and Yuval Efron;
licensed under Creative Commons License CC-BY 4.0

2. **Adaptive Security.** We say that a consensus protocol is *secure against an adaptive adversary* if it solves consensus even when the adversary may choose to corrupt players "on the fly," based on its observations thus far. We consider both a basic *adaptive adversary* which can corrupt up to f players at any point based on the current transcript of the protocol, but cannot un-corrupt them, and a *mobile blocking adversary* which can choose potentially new players to target in each round but can only block their messages, and not cause arbitrary behavior.

Observe that either of these properties is straightforward to achieve without the other: if efficiency is not important, classical Byzantine fault-tolerant consensus protocols [10] are adaptively secure. On the other hand, Nakamoto-style consensus [32] with round-robin leader election is efficient against a non-adaptive adversary, but has linear latency against an adaptive adversary.

In practice, the vast majority of blockchains, including Ethereum and Bitcoin, aim to achieve both of the above properties. They do so by relying on a source of shared randomness for the players, for tasks such as leader election and committee sampling [5, 32, 14]. Such a primitive that provides access to fresh randomness at every point in time is referred to in the literature as a *randomness beacon* [34]. Implementing such a beacon, however, is an expensive task, with current constructions either employing delay functions [29, 7], requiring many rounds or much communication [37, 12, 35, 24], or achieving a sub-optimal version of the ideal functionality subject to manipulation [22, 5].

On the other hand, it turns out, that the use of *unpredictable* randomness is essential to the design of efficient consensus protocols [17, 6, 1]. Dolev and Reischuk [17] show that any deterministic protocol must have $\Omega(n^2)$ communication complexity. Bar-Joseph and Ben-Or [6] show that against a computationally unbounded adaptive adversary that can view the local state of all players, any (even randomized) synchronous BA protocol requires $\tilde{\Omega}(\sqrt{n})$ rounds. Abraham et al. [1] show that without unpredictable randomness, any consensus protocol must use $\Omega(n^2)$ communication.

1.1 Our Setting

The goal of this paper is to quantify the precise amount of randomness that must be drawn from such a beacon to allow for efficient and adaptively secure consensus protocols. We make minimal auxiliary assumptions, focusing on the information-theoretic (i.e., a computationally unbounded[1] adversary and no PKI), synchronous setting. Specifically, we consider the following ideal functionality for a beacon: At each round t, a fresh, uniformly random string is revealed to all players.

Such an assumption, referred to as an *idealized common coin* or *randomness beacon* is a common assumption in the BA protocols for asynchronous networks [15, 31]. We say that a protocol has *low beacon entropy* if it can be implemented using a randomness beacon that generates $O(\log n)$ bits in total during the protocol. We stress that it is important that the random string revealed to all players at round t is not only uniform, but also is *unpredictable* prior to round t. In other words, no adversary can *predict* the contents of the string of round t prior to round t. Abraham et al. [1] show that any protocol secure against an adversary that can predict the content of the beacon ahead of time, must use $\Omega(n^2)$ communication. We emphasize that our lower bound applies even when the protocol has access to our beacon with stronger unpredictability.

[1] Under computational assumptions, e.g., using a delay function, a beacon can be constructed. The unbounded adversary setting enables us to isolate the utility of the beacon.

1.2 Our Results

We provide a complete and tight characterization of efficiency, adaptive security, and low beacon randomness:

Impossibility (Section 4, Theorem 7 and Corollary 8): We show that no consensus protocol can simultaneously achieve all three properties. That is, any efficient, adaptively secure consensus protocol must use a randomness beacon that outputs $\omega(\log n)$ bits. Our impossibility result holds against a computationally unbounded adversary in a broadcast communication model.

Possibility (Section 5, Lemmas 9, 15, and 17): We show that the above impossibility is tight by presenting three consensus protocols, each realizing exactly two of the above three properties. Our protocols work against a computationally unbounded adversary in a peer-to-peer communication model. Together, they demonstrate a *trilemma*: consensus protocols can achieve any two of efficiency, adaptive security and low-entropy, but not all three.

2 Related Work

A long line of work considers a *computationally bounded* adversary [13, 2, 18, 21]. Protocols in this setting employ cryptographic tools to enable adaptive secure consensus protocols with as few as $O(1)$ rounds (in expectation). This research culminates in the work of Ghinea, Goyal and Liu-Zhang [21], which matches the decades-old lower bound Chor, Merritt, and Shmoys [13] which states that any r round consensus protocol has error probability at least $\Omega(\frac{1}{r^r})$.

Coming back to an unbounded adversary, other works [26, 36, 8, 1, 16] consider relaxed versions of the adaptive adversary, and design consensus protocols that circumvent the aforementioned round and communication lower bounds of [1, 6].

Adaptively secure consensus. By Abraham et al. [1], any protocol that is secure against a strongly adaptive adversary must use $\Omega(n^2)$ bits of communication. This bound is known to be tight in the setting of a computationally bounded adversary by the work of Abraham et al. [2]. The protocol of this work also achieves optimal $O(1)$ round complexity, in expectation.

Adversary Relaxations. On the road to circumventing the lower bounds of [1, 6], various relaxations to the strongly rushing adversary have been considered.

- A typical relaxation is that if the adversary chooses to corrupt a player p at round r, player p still performs the honest behaviour of round r prior to the adversary taking control of p (at which point the adversary can send additional messages, still in round r). Protocols achieving $o(n^2)$ communication complexity under this assumption include the work of King and Saia [26], and the follow up work by King, Lonargan, Saia and Trehan [25] in which they show protocols with $O(n^{1.5})$ communication complexity. Additional cryptographic assumptions allow the design of protocols with nearly linear communication complexity [8, 11, 2, 9]. Another class of protocol circumventing the communication lower bound of [1] are motivated by blockchain applications, and make use of proof-of-work or proof-of-stake assumptions [16, 32].
- Another line of work considers a *late* adaptive adversary. Roughly, this is an adaptive adversary with an outdated view of the state of the protocol. At each round the adversary may choose players to corrupt based on all information available so far, however the actual corruption occurs several rounds later. Most of these works consider relaxed variants of the consensus problem, e.g., almost-everywhere consensus [36, 4, 3, 27].

- A mobile blocking adversary has been considered previously in [36], and is generally related to the well studied model of omission failures [30, 23].

Randomness beacon entropy. As mentioned, the idealized randomness beacon assumption, also known as a common coin, is extensively used in consensus protocol design in asynchronous networks (See [31] and references within). A recent work [23] studies a similar question to ours. Specifically, they consider protocols secure against an adaptive *omission failure* adversary, and characterize the trade-off between the required number of oracle calls to a random beacon and the round complexity.

3 Preliminaries

Notation. We let λ denote the security parameter. If a function $f(\lambda)$ is $O(1/\mathsf{poly}(\lambda))$ for every polynomial in λ, we say f is *negligible* in λ. We write $f = \mathsf{negl}(\lambda)$. We let $\log(x)$ denote the logarithm base 2 of x. If X is a random variable over domain Ω, and $\mathrm{Supp}(X) = \{x \in \Omega \mid \Pr[X = x] > 0\}$, we denote the min-entropy of X by $H_\infty(X) = \min_{x \in \mathrm{Supp}(X)} \frac{1}{\Pr[X=x]}$.

Model. We consider a network of n players (eq. nodes or participants). We focus here on the permissioned setting, in which the set of n players is fixed and known to all players in advance. At most f players can be corrupt. We further assume that communication takes place over a synchronous network with maximum message delay of Δ, i.e. a player p at round t has received all messages sent to it up to round $t - \Delta$. For simplicity of exposition, we assume that $\Delta = 1$, but all our results extend naturally to the general case of delay parameter Δ. Throughout the paper we consider two models for a communication network. In the *peer-to-peer* model, players may send different messages to different players during each round. In the *broadcast* model, players (including corrupted players) may only broadcast a message, which is received by all other players.

Specifically, the latter model does not allow corrupt players to equivocate. Our lower bound holds even in the more generous broadcast model, and our upper bounds hold even in the more general peer-to-peer model of communication.

Adversary model. All of the adversaries considered in the paper are computationally unbounded, and in particular we assume no PKI (though we do assume authenticated channels between participants). Specifically, we consider three types of adversaries, *static*, *adaptive*, and *mobile blocking*. Let f be the adversary's corruption budget. It is useful for us to consider an adversary as a *pair* of algorithms $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, where $\mathcal{A}_0$ is responsible for choosing the identities of up to f corrupted players at every round t, and $\mathcal{A}_1$ is the adversary participating in the protocol. A bit more formally, at each round t, $\mathcal{A}_0$ may observe (Π, t, Tr_t) and output a set $\mathcal{F}_t$ of at most f players. Here, Π is a description a protocol and Tr_t refers to the transcript of a protocol up to round t (see Definition 2). Each adversary type we consider imposes different restrictions on either $\mathcal{A}_0$ or $\mathcal{A}_1$.[2]

- *Static.* The static adversary chooses the identities of f corrupt players at round $t = 0$, and this choice is fixed throughout the execution of the protocol. The adversary has complete control over the behaviour of the corrupted players. In other words $\mathcal{F}_t = \mathcal{F}_0$ for all t. There are no restrictions on $\mathcal{A}_1$.

[2] One might ask about the possibility of a mobile adversary capable of corruption, but this is tricky to reason about as it requires a notion of "un-corrupting" players.

- *Adaptive.* At the beginning of each round t, the adversary chooses the identities of $f_t < f$ players to corrupt in that round, based on its observations of the transcript of the protocol and the players it has corrupted thus far. These players stay corrupted for the rest of the execution. In other words, $\mathcal{F}_t \subseteq \mathcal{F}_{t+1}$ for all t. There are no restrictions on $\mathcal{A}_1$.
- *Mobile blocking.* A mobile blocking adversary can, at every round, observe (Π, t, Tr_t), where t is a round, and Tr_t is the transcript of Π up to round t, and output a set $\mathcal{F}_t$ of at most f players that are prohibited from sending messages at round t. We emphasize that unlike the adaptive adversary which is constrained by the total number of corruptions, the mobile adversary may corrupt up to f players in each round regardless of how many players it has corrupted in the past. A mobile blocking adversary is also captured by the $(\mathcal{A}_0, \mathcal{A}_1)$ notation via $\mathcal{A}_0(\Pi, t, \mathrm{Tr}_t) = S$, and $\mathcal{A}_1$ be an adversary behaviour in which corrupted players send no messages.

In any adversary variant, we say that an adversary corrupting at most $f = \rho n$ players is *ρ-bounded*.

Randomness. The protocols we consider in this work have two main sources of randomness. For both of them, we assume an ideal functionality as our goal is to reason about the nature and amount of randomness required to design consensus protocols.

1. **Common Random String (CRS).** We assume that at round $t = 0$, an arbitrarily long (as per the protocol's specification) uniformly random string CRS is revealed to all players. A *static* adversary must choose the identities of corrupt players *prior* to the CRS being revealed. An *adaptive* or *mobile* adversary can choose the identities of corrupt players after CRS is revealed.
2. **Randomness Beacon.** A randomness beacon is an oracle that at every round t, produces a fresh uniformly random string seed_t of an arbitrary length (as per the protocol's specification). In particular, for each round t, an *adaptive* or a *mobile* adversary must choose the identities of corrupted players prior to the revelation of seed_t.

Clearly, in the presence of an adaptive adversary the latter source of randomness is significantly more powerful than the former. It is precisely the latter source of randomness that is the main focus of this paper.

Authenticated Channels. Similarly to prior work [26, 8], we assume that communication channels in our network are *authenticated*. Intuitively, this means that each player knows the identity of the sender for any message received. This can be formalized by instantiating communication channels with a map that maps a message m to the tuple (m, p), where p is the sender of m. This mapping is fixed and can not be tampered with by the adversary.

Executions. An *execution* is a tuple $E = (\Pi, \mathcal{A}, r)$ where Π is the protocol run by honest players. $\mathcal{A}$ is a particular adversary, i.e. $\mathcal{A}$ is an algorithm that chooses corrupt players (only at $t = 0$ if static, otherwise $\mathcal{A}_0$ chooses at every round t) based on the transcript of the protocol and internal state of corrupt players, and instructions for the behaviour of the environment. r denotes the random coins of all players. Given a protocol Π we say that E is an execution of Π if the first component of E is Π.

We say that an execution is *admissible* in the adaptive model if for all rounds $t \geq 0$ it holds that $f_t < \frac{1}{3}n$. In the static model we further demand that the identities of corrupt players remain fixed throughout the execution of the protocol. At times we abuse notation and refer to an execution also as the random variable $(\Pi, \mathcal{A})$ which is a distribution over executions in which Π is the protocol run by honest players, and $\mathcal{A}$ is the adversary (corrupt players in each round, and actions taken by them). We denote this random variable by $E_{\mathcal{A}}$.

The specification of the consensus problem we consider takes the form of the well known Byzantine Agreement problem [33, 28].

Byzantine Agreement (BA). In the BA task, n players must come to an agreement on a bit under some constraints. Each player P_i has an input $b_i \in \{0,1\}$, and produces an output $o_i \in \{0,1\}$. We say that a protocol Π solves the BA task if the following three properties are guaranteed by Π, except for with $\mathsf{negl}(\lambda)$ probability.

1. *Termination.* There exists t such that all honest players have produced an output by round t.
2. *Agreement.* For any pair of honest players P_i, P_j, the probability (over the randomness of the adversary and the protocol) that these players output different values is at most $\mathsf{negl}(\lambda)$.
3. *Validity.* If all honest players have the same input bit b, then all honest players always output b.

An additional critical notion relating to a BA protocol is *latency*, which is defined as follows. Intuitively, latency captures the expected number of rounds it takes for Π to terminate.

▶ **Definition 1** (Latency). *Let Π be a protocol solving BA, and let $T_{i,\mathcal{A}}$ denote the random variable indicating the round in which player i terminates given an adversary $\mathcal{A}$ corrupting some subset of the players. Given an execution $E = (\Pi, \mathcal{A}, r)$, let H_E denote the set of players that remain honest throughout the entire execution.* Π *has* expected *latency ℓ given f corruptions if for all adversaries $\mathcal{A}$ corrupting at most f players,*

$$\max_{\substack{(\Pi,\mathcal{A}) \\ \mathcal{A} \text{ corrupts at most } f \text{ players}}} \mathop{\mathbb{E}}_{E=(\Pi,\mathcal{A},r)} \left[\max_{p \in H_E} T_{p,\mathcal{A}} \right] \leq \ell.$$

Randomness beacon. In this paper, we consider an ℓ-bit randomness beacon to be an n-party protocol $\Pi^{\mathcal{O}}$ *with access to a random oracle* $\mathcal{O}$ that satisfies termination and agreement and, on input 1^λ, outputs a string s of length $\ell(\lambda)$. Furthermore, the value output by the honest parties must be computationally indistinguishable from random, *in the random oracle model*. In other words, the adversary responsible with distinguishing between the uniform distribution and the output of the beacon is only bounded in its number of queries to the random oracle, but not in any other manner. In particular, it can perform arbitrary computation. More precisely, let $\mathcal{A}$ be any (computationally unbounded) adversary corrupting f out of n players. Let $\mathcal{B}^{\mathcal{O}}$ by any computationally bounded distinguisher, making $poly(\lambda)$ queries to the random oracle $\mathcal{O}$. Denote by $v \leftarrow \Pi^{\mathcal{O}}_{\mathcal{A}}$ the output of the honest parties from an execution of Π with the adversary $\mathcal{A}$. For any such $\mathcal{A}, \mathcal{B}^{\mathcal{O}}$ it must hold that:

$$\left| \Pr[\mathcal{B}^{\mathcal{O}}(1^\lambda, v) = 1 : \ v \leftarrow \Pi^{\mathcal{O}}_{\mathcal{A}}] - \Pr[\mathcal{B}^{\mathcal{O}}(1^\lambda, r) = 0 : \ r \leftarrow \{0,1\}^{\ell(\lambda)} \ ] \right| \leq \mathsf{negl}(\lambda).$$

We say that $\Pi^{\mathcal{O}}$ is secure against a static/adaptive/mobile blocking adversary if the above holds and $\mathcal{A}$ is a static/adaptive/mobile blocking adversary respectively. We note that the definition of a randomness beacon varies throughout the literature; here, we consider a weak notion where the adversary corrupting parties during the protocol execution is *different* from the adversary attempting to distinguish the beacon output from random. This notion is still nontrivial.

4 Impossibility Result

In the following, we define formal notions in order to rigorously discuss the *amount* of common randomness consumed by a BA protocol with adaptive security and low communication. Recall that in this section, we assume to be in the broadcast model of communication. We begin with defining the transcript of a protocol.

▶ **Definition 2** (Transcript). *Consider a* BA *protocol* Π *for a network of* n *players, and let* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *denote some adversary. Denote by* $E_{\mathcal{A}}$ *the random variable of executions under* $(\Pi, \mathcal{A})$*. We define the transcript* $Tr_{E_{\mathcal{A}}}$ *to be the random variable that indicates the identities and contents of the players speaking in each round. Formally, we have*

- $\mathrm{Tr}_{E_{\mathcal{A}}} = \left\{\mathrm{Tr}^t_{E_{\mathcal{A}}}\right\}_{t\in\mathbb{N}}$
- $\mathrm{Tr}^t_{E_{\mathcal{A}}} = (I^t_{E_{\mathcal{A}}}, C^t_{E_{\mathcal{A}_1}}, A^t_{\mathcal{A}}, \mathsf{seed}_t, \mathsf{CRS})$*, where* $I^t_{E_{\mathcal{A}_0}} \subseteq [n]$ *is a subset of honest players at round* t*, and* $C^t_{E_{\mathcal{A}}} \in \left(\{0,1\}^*\right)^{I^t_{E_{\mathcal{A}}}}$ *denotes the messages of those players. Furthermore, an honest player* p_i *speaks in round* t *iff* $i \in I^t_{E_{\mathcal{A}}}$*, and the message it sends is consistent with* $C^t_{E_{\mathcal{A}}}$*.* $A^t_{\mathcal{A}}$ *contains the identities of the adversarial parties speaking at round* t *and the contents of their messages*

We consider the following notion of unpredictability, that intuitively says that the probability that an adaptive adversary can guess correctly the set of speaking parties in every round, given the transcript of the protocol up to that round is negligible.

▶ **Definition 3** (Adversary's guess of speaking parties). *Let* Π *be a consensus protocol, and let* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *be an adversary. We define the adversary's* guess *to be a series* $(\hat{I^0_{\mathcal{A}}}, \hat{I^1_{\mathcal{A}}}, \hat{I^2_{\mathcal{A}}}, ...)$ *given by an algorithm* $\mathcal{A}_2$ *of the adversary's choice where for each* $t \in \mathbb{N}$*,*

$$\hat{I}^t_{\mathcal{A},\mathcal{A}_2} \leftarrow \mathcal{A}_2(\Pi, \left\{\mathrm{Tr}^i_{E_{\mathcal{A}}} \mid 0 \le i \le t-1\right\}, t)$$

Given the above definition, we can now define global leader unpredictability:

▶ **Definition 4** (Global leader unpredictability). *Let* Π *be a* BA *protocol. We say that* Π *has global leader unpredictability if for any adversaries* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *and* $\mathcal{A}_2$*,*

$$\Pr\left[\forall t, \hat{I}^t_{\mathcal{A},\mathcal{A}_2} = I^t_{E_{\mathcal{A}_0}} \mid \mathsf{CRS}\right] = \prod_{t=0}^{\infty} \Pr\left[\hat{I}^t_{\mathcal{A},\mathcal{A}_2} = I^t_{E_{\mathcal{A}_0}} \mid \left\{I^i_{E_{\mathcal{A}_0}} \mid 0 \le i < t\right\}, \mathsf{CRS}\right] \le \mathsf{negl}(\lambda)$$

We now prove that if a protocol Π satisfies global leader unpredictability, then its transcript contains a significant amount of min-entropy.

▶ **Lemma 5.** *Let* Π *be a* BA *protocol that satisfies global leader unpredictability. Then for any constant* $c > 0$ *and any adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*,*

$$\sum_{t=0}^{\infty} H_\infty(\mathrm{Tr}^t_{E_{\mathcal{A}}} \mid \left\{\mathrm{Tr}^i_{E_{\mathcal{A}}} \mid 0 \le i < t\right\}, \mathsf{CRS}) \ge c \log n.$$

Proof. Assume towards a contradiction that there exist a constant $c > 0$ and an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ such that

$$\sum_{t=0}^{\infty} H_\infty(\mathrm{Tr}^t_{E_{\mathcal{A}}} \mid \left\{\mathrm{Tr}^i_{E_{\mathcal{A}}} \mid 0 \le i < t\right\}, \mathsf{CRS}) < c \log n.$$

Now for each $t \in \mathbb{N}$, denote by k_t the value $H_\infty(\mathrm{Tr}^t_{E_\mathcal{A}} \mid \{\mathrm{Tr}^i_{E_\mathcal{A}} \mid 0 \leq i < t\}, \mathsf{CRS})$. Applying $f(x) = \frac{1}{2^x}$ to both sides of the inequality above, we get that

$$\frac{1}{2^{\sum_{t=0}^{\infty} k_t}} > \frac{1}{n^c}$$

Thus, by definition of min-entropy, we have that there exist strings $S_t, t \in \mathbb{N}$ s.t.

$$\frac{1}{2^{k_t}} = \Pr[\mathrm{Tr}^t_{E_\mathcal{A}} = S_t \mid \{\mathrm{Tr}^i_{E_\mathcal{A}} \mid 0 \leq i < t\}, \mathsf{CRS}]$$

We thus have that

$$\Pr[\forall t \in \mathbb{N}, \mathrm{Tr}^t_{E_\mathcal{A}} = S_t \mid \mathsf{CRS}] = \prod_{t=0}^{\infty} \Pr[\mathrm{Tr}^t_{E_\mathcal{A}} = S_t \mid \{\mathrm{Tr}^i_{E_\mathcal{A}} \mid 0 \leq i < t\}, \mathsf{CRS}] > \frac{1}{n^c}$$

where the first transition is by definition of the probability of event intersection. Since $I^t_{E_\mathcal{A}}$ is a part of $\mathrm{Tr}^t_{E_\mathcal{A}}$, we thus get that there exists set $\hat{I}_t, t \in \mathbb{N}$ s.t.

$$\Pr[\forall t \in \mathbb{N}, I^t_{E_\mathcal{A}} = \hat{I}_t \mid \mathsf{CRS}] > \frac{1}{n^c}$$

Now note that an adversary $\mathcal{A}_2$ that predicts at round t the set $\hat{I}_t$ can realize this success probability, thus violating the global leader unpredictability condition, as required. Such an adversary is realizable both in the adaptive and mobile blocking models since the adversary at round t has definitive knowledge of the transcript up to and including round $t-1$, including CRS. ◀

We would now like to prove that the lack of the global unpredictability condition, combined with low communication, implies susceptibility to attacks by adaptive adversaries. With the notion of a transcript in hand, we can also now formally define the relevant notion for this paper of *low* communication.

▶ **Definition 6.** *Let Π be a consensus protocol, and let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be an adversary. We define the communication complexity of Π to be the random variable $C = \sum_{t \in \mathbb{N}} |I^t_{E_\mathcal{A}}|$. We say that the communication is uniform if there exists T such that except for with $\mathsf{negl}(\lambda)$ probability it holds that $|I^t_{E_\mathcal{A}}| = O(\frac{C}{T})$ for all $t \leq T$ and $|I^t_{E_\mathcal{A}}| = 0$ for all $t > T$. We refer to $\frac{C}{T}$ as the uniform communication complexity of Π. We say that a protocol has* low communication *if it has uniform communication of $o(n)$.*

The goal of defining uniform communication, as opposed to just considering the general number of bits exchanged between players in the protocol, is to speak rigorously about protocols in which only a few players speak in each round. We now aim to prove the following theorem, which says that any BA protocol in which few players speak in every round, uses low beacon entropy, and is adaptively secure, requires many rounds. In other words, no BA protocol can simultaneously be efficient, adaptively secure, and use low beacon entropy.

▶ **Theorem 7.** *Let Π be a protocol that has the following properties:*

- *Π does not satisfy global leader unpredictability.*
- *Except for $O(1)$ initial rounds, Π has uniform communication complexity k*

Then w.p. $\frac{1}{p(n)}$ for some polynomial p, Π requires $\Omega(\frac{\rho n}{k})$ rounds in the presence of a ρ-bounded adaptive adversary, for any $\rho \geq \frac{k}{n}$. Furthermore, if a mobile blocking adversary is considered, then Π does not exist for any $\rho \geq \frac{k}{n}$.

Proof. By the assumption that Π does not satisfy global leader unpredictability, we deduce that there exist a polynomial $p(n)$ and adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ and $\mathcal{A}_2$ such that

$$\Pr[\forall t, \hat{I}^t_{\mathcal{A},\mathcal{A}_2} = I^t_{E_{\mathcal{A}_0}} \mid \mathsf{CRS}] \geq \frac{1}{p(n)}.$$

Now consider the following adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$, acting as follows. Let $r = O(1)$ be the number of initial rounds which may have high communication complexity. In the following, we employ a well-known result [13], which states that any BA protocol with r rounds has error probability of at least $\Omega(\frac{1}{n^r})$, with the adversary needing to corrupt at most r players [13]. In our case, $\Omega(\frac{1}{n^r}) = 1/n^{O(1)}$.

1. For the first r rounds, employ the adversary strategy [13], corrupting at most r players in the process.
2. $\mathcal{B}_0$ acts as follows: At the beginning of any round $t > r$, use $\mathcal{A}_2(\Pi, \{\mathrm{Tr}^i_{E_{\mathcal{A}}} \mid 0 \leq i < t\})$ to obtain $\hat{I}^t_{\mathcal{A},\mathcal{A}_2}$. In the case of the adaptive adversary, corrupt all the players in $\hat{I}^t_{\mathcal{A},\mathcal{A}_2}$ until the total number of corrupted parties has reached the corruption budget ρn. In the case of a mobile blocking adversary, corrupt them for round t.
3. $\mathcal{B}_1$ sends no messages by the corrupted players; that is, the players in $\hat{I}^t_{\mathcal{A},\mathcal{A}_2}$. Note that this is compatible behaviour with the mobile blocking adversary, in addition to the adaptive adversary.

From here on in, the analysis is conditioned on the event that $\mathcal{A}_2$ has guessed correctly the identity of speakers at all rounds, which occurs w.p. at least $\frac{1}{p(n)}$, by assumption. Due to the strategy of [13], we have that with at least inverse polynomial probability, Π does not solve BA up to round r of the execution. For the following $\Omega(\frac{\rho n}{k})$ rounds, no progress is made since the adversary corrupts all parties participating in the protocol for those rounds. Thus at least with inverse polynomial probability, Π requires $\Omega(\frac{\rho n}{k})$ rounds, as required. In the case of a mobile blocking adversary, no progress is made at all, indefinitely, and so in particular the protocol either has no termination, or has error probability at least $\Omega(\frac{1}{n^r \cdot p(n)})$, which is a contradiction. ◀

We now show that global leader unpredictability implies a randomness beacon. This randomness beacon simply runs Π and computes the hash(i.e. queries the random oracle) of the identities that spoke in each round. Since these identities are unpredictable, given enough parties there is sufficient randomness to construct a beacon.

▶ **Corollary 8.** *Let Π be an n-party protocol satisfying global leader unpredictability. If $n \geq \lambda$, there exists a randomness beacon $\Pi'^{\mathcal{O}}$ such that:*

- *$\Pi'^{\mathcal{O}}$ has the same communication complexity and latency as Π.*
- *$\Pi'^{\mathcal{O}}$ outputs λ bits that are indistinguishable from random by any efficient ($poly(\lambda)$ query) adversary in the random oracle model.*

Proof. Let $\mathcal{O} : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function which we model as a random oracle. Let $\Pi'^{\mathcal{O}}$ be the protocol obtained by running Π and outputting the hash(i.e. querying the oracle $\mathcal{O}$) of the identities of the parties that speak in each round of Π. Observe now that $\Pi'^{\mathcal{O}}$ trivially satisfies the first bullet point, since it involves only running Π and applying a function to its transcript. Since we operate in the broadcast setting with authenticated channels, all parties know these identities of speaking parties. Recall that global leader unpredictability states that this tuple of speaking parties has at least $c \log n$ min-entropy for any constant c. Therefore, for any fixed string s, the probability that this tuple of identities equals s is at most $1/\lambda^c$ for any constant c. Let $\mathcal{B}^{\mathcal{O}}$ be a computationally bounded adversary

making a polynomial number of queries to the random oracle $\mathcal{O}$. The probability that $\mathcal{B}$ queried the tuple of speaking parties to $\mathcal{O}$ is smaller than the inverse of any polynomial, which is negligible. Therefore, except with negligible probability the hash of the tuple of speaking parties is a freshly random value from the perspective of $\mathcal{B}^{\mathcal{O}}$. ◀

5 Possibility Results

Having established the above trade-off, we turn to showing that it exactly captures the role of randomness in efficient and adaptively secure BA. Specifically, we present three protocols solving BA, all simplified versions of known protocols from the literature, and prove that each of them satisfies two of the three properties we described above. To make our results as strong as possible, throughout this section, we consider a model where players are deterministic and all players are given access to an ideal randomness beacon and a CRS. Recall (see Section 3) that an adaptive adversary must make corruption choices for round r *prior* to seeing the beacon output of round r. While for the lower bound result we assumed a broadcast model of communication, we assume a peer-to-peer network for the upper bounds, to make our results as strong as possible. In particular, corrupt players can equivocate. We formally describe our framework and general BA protocol in the following section, and then showcase how this framework is instantiated in three ways to obtain protocols:

1. **Efficiency and adaptive security.** We show that when entropy from a beacon is not limited, there exists a protocol that is adaptively secure and efficient. The details, along with our general framework, are in Section 5.1.
2. **Low beacon entropy and adaptive security.** If one is willing to forgo low communication, we show that there is an adaptively secure, low beacon entropy protocol that also has $O(1)$ expected round complexity. The details are in Section 5.2.
3. **Low beacon entropy and efficiency.** If one wishes to forgo adaptive security, then there exists an efficient protocol that uses low beacon entropy that is secure against a static adversary. The details are in Section 5.3.

Mobile blocking adversary. All the proofs relating to security against an adaptive adversary in this section can easily be modified to work for a mobile blocking adversary, with the *same* protocols. The main observation is that against an adversary that can only silence players, none of our proofs use the property that *the same* parties are corrupted between rounds, and also that in our protocols, the actions of an honest player depend only on the messages received from the previous round, and not any other round in the past.

5.1 Efficiency and Adaptive Security

In this section we describe our general framework for BA protocol design, along with one instantiation of it to obtain a BA protocol that has low communication(i.e. $O(\lambda)$ parties talk in each round) and is secure against an adaptive adversary, as long as $3f_t + 1 < (1-\epsilon)n$ for all $t \in \mathbb{N}$ and constants ϵ. Formally, we prove the following in this section:

▶ **Lemma 9.** *For all $\epsilon > 0$, assuming a randomness beacon, there exists a protocol Π that except for with $exp(-\lambda)$ probability, solves the* BA *task in the presence of an adaptive adversary that satisfies $3f_t + 1 < (1-\epsilon)n$ for all $t \in \mathbb{N}$.*
Furthermore, the protocol has $O(1)$ expected latency, and the protocol has uniform $O(\lambda)$ communication complexity, in expectation.

Our general framework, with which all three of our protocols are designed, resembles that of Gafni and Losa [19] of interlacing executions of the commit-adopt task and leader election. The main differences are the use of the randomness beacon to ensure both low communication and security against an adaptive adversary for some of our considered settings, and the consideration of a stronger adversary (Losa and Gafni assume a non equivocating adversary). With this in mind, we describe in this section the general framework and its concrete implementation to obtain Lemma 9. The following sections explain how to modify the implementation to obtain the other two protocols.

In all our protocols, the set of parties tasked with speaking in each round is denoted by com_t for round t. We refer to com_t as the *committee* of round t. Furthermore, each round t has a designated leader, denoted by ℓ_t. Despite the fact that *every round* has a leader, in most rounds the leader does not have any special role. The identity of com_t and ℓ_t in each of our considered settings is derived both from the beacon and the given CRS in different ways, as follows.

1. **Efficiency and adaptive security.** To achieve Lemma 9, we use the randomness beacon as follows. Denote the string distributed at round t to the players by the randomness beacon by seed_t. We choose $|\mathsf{seed}_t| = O(\lambda \log n)$, where λ is the security parameter, and we treat seed_t as $\mathsf{seed}_t = (\ell_t, com_t)$ where $\ell_t \in [n]$ is the identity of a player, which is referred to as the *leader* of round t, and $com_t \subseteq [n]$ is a subset of size $O(\lambda)$ of players, indicating the *committee* of round t. The protocol presented in this section makes no use of the given CRS.
2. **Adaptive security and low beacon entropy.** This protocol also makes no use of the CRS. In this protocol, $com_t = [n]$ for all t, and the leader ℓ_t is elected via the randomness beacon string seed_t as above. See Section 5.2 for further details.
3. **Efficiency and low beacon entropy.** This protocol is the only protocol that makes use of the CRS. We interpret the CRS as $(\mathsf{seed}_1, \mathsf{seed}_2, \ldots)$ where $\mathsf{seed}_t = (\ell_t, com_t)$ with $\ell_t \in [n]$ and $com_t \subseteq [n]$ of size $O(\lambda)$. See Section 5.3 for further details.

Our framework is comprised from the interlacing of two components.

- Commit-Adopt (CA). The CA protocol consists of 2-rounds. In each of these rounds, only the *committee* com_t speaks. If the CA step fails to achieve consensus, players proceed to the second phase of the protocol.
- Conciliator (CO). Intuitively, the goal of the Conciliator task is to bring back the honest players into a consistent view after the previous CA failed. This is done by running an additional CA and a leader election, and then outputting a value to continue with for the subsequent CA.

We now formally define the two procedures CA, and CO.

▶ **Definition 10** (Commit-Adopt). *In the commit-adopt task (CA), each player receives an input value z, and must produce an output of the form commit(z') or adopt(z') for some value z', with the following guarantees.*

1. Agreement. *If an honest player outputs commit(z) for some value z, then all honest players must output either commit(z) or adopt(z).*
2. Validity. *If all honest players input the same value z, then all honest players must output commit(z).*
3. Termination. *There exists a round $r \in \mathbb{N}$ such that by round r, all awake honest players have submitted an output.*

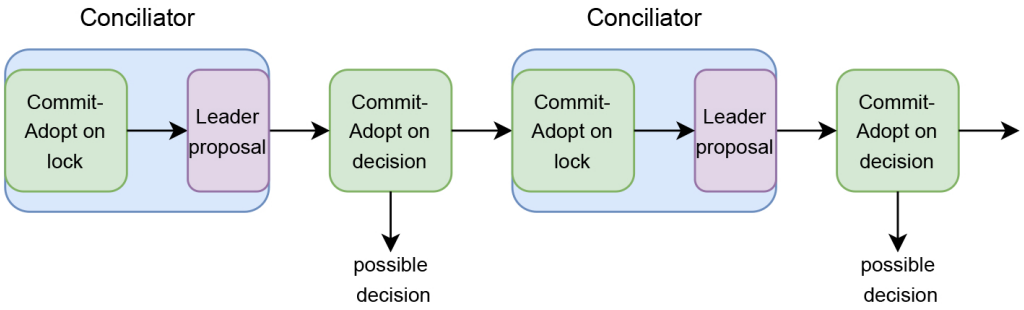


Figure 1 Illustration of the structure of all three of our protocols. CA and CO procedures are interlaced, with each CO procedure ensuring safety on a potentially locked value, followed by letting a leader propose a value to the decision CA. In each round t, only the players in com_t or ℓ_t send messages.

▶ **Definition 11** (Conciliator). *In the conciliator task (CO), each player has an input value z and must produce an output with the following guarantees.*

1. Validity. *If all honest players input the same value z, then all honest players output z.*
2. Termination. *There is a round $r \in \mathbb{N}$ such that all honest players output by round r.*
3. Probabilistic Agreement. *With probability at least $\frac{2}{3}$, all players output the same value z inputted by some honest player.*

Our generic protocol alternates between executions of CO and CA until BA is solved. See Figure 1 for an illustration. We denote the i-th execution of CA and CO by CA$[i]$ and CO$[i]$, respectively. We now provide formal descriptions of the protocols for CA and CO using ℓ_t, com_t as explained above to obtain Lemma 9. The following sections explain how these implementations are modified to obtain our other protocols.

Algorithm 1 CA.

We employ the following adopt-commit protocol executed by all honest players p with input z_p.

1. At round $t = 0$, if $p \in com_0$, p broadcasts its input z_p. Otherwise, do nothing.
2. At round $t = 1$, if $p \notin com_1$, do nothing. Otherwise, if there exists a value z that p has received z from more than $\frac{2|com_0|}{3}$ of the players in com_0[3], p broadcasts $vote(z)$. Otherwise, do nothing.
3. At round $t \geq 2$, p decides on its output as follows.
   a. If there exists value z such that p has received $vote(z)$ from at least $\frac{2|com_1|}{3}$ of the players in com_1, output $commit(z)$.
   b. Else if there exists a value z for which p received more $vote(z)$ from players in com_1 than for any other value, output $adopt(z)$.
   c. Else, output $adopt(z_p)$.

We now prove that the above procedure solves the CA problem in the presence of an adaptive adversary, as long as there is a constant $\epsilon > 0$ s.t. $3f_t + 1 < (1 - \epsilon)n$ for all $t \in \mathbb{N}$. In general, the above procedure solves the CA problem as long as com_t contains a greater than $\frac{2}{3}$-majority of honest players for all rounds t. In the context of Lemma 9, we make the following observation, which is used liberally throughout the section. We denote the above protocol by Π.

▶ **Observation 12.** *Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be an adaptive adversary. Then for all t it holds that*

$$\Pr\left[|\mathcal{A}_0(\Pi, t, \mathrm{Tr}_t) \cap com_t| \geq \frac{|com_t|}{3}\right] = \mathsf{negl}(\lambda)$$

Proof. The proof follows from standard concentration bounds. Specifically we get that the expected number of corrupted players in com_t is upper bounded by $\frac{|com_t|}{3}(1-\epsilon)$, and thus by a Hoeffding bound we get the probability that the number of corrupted players in com_t exceeds $\frac{|com_t|}{3}$ is at most $e^{-\Omega(\epsilon^2|com_t|)} = e^{-\Omega(\epsilon^2\lambda)}$. For this we use the facts that $|com_t| = \Omega(\lambda)$, ϵ is a constant, the choice of each player in com_t being independent, and the independence of $\mathcal{A}_0(\Pi, t, \mathrm{Tr}_t)$ on com_t. ◀

We thus assume from here on in for the remainder of the analysis that an honest supermajority assumption holds at all rounds amongst the committee members, i.e. $|\mathcal{A}_0(\Pi, t, \mathrm{Tr}_t) \cap com_t| < \frac{|com_t|}{3}$ holds for all t. With that in mind all that is left is to prove that the above protocol solves the CA problem.

▶ **Lemma 13.** *For any constant $\epsilon > 0$, except for with $\mathsf{negl}(\lambda)$ probability, Algorithm 1 solves the CA task whenever $3f_t + 1 < (1-\epsilon)n$.*

Proof. Termination is clear from the behavior of the protocol. We move on to Validity, assume that all honest parties start the protocol with input z, and let com_0, com_1 be as in the protocol. Then in particular, all honest players in com_0 broadcast z. The above observation implies that any honest player in com_1 observes the value z from more than $\frac{2}{3}$ of the players in com_0. Thus all honest players in com_1 broadcast $vote(z)$ message at round 1. Which in turn causes all honest players to output $commit(z)$ at $t = 2$, as required. For Agreement, let $commit(z)$ be the value output by some honest player p. Which means that p observed $vote(z)$ from more than $\frac{2}{3}$ of the members of com_1. Note first that no other honest member of com_1 sends $vote(z')$ for a different value, as this implies that that two honest players q_1, q_2 in com_1, respectively received multicasts of z, z' from more than $\frac{2}{3}$ of the members of com_0, and a quorum intersection argument implies the existence of an honest player in com_0 that broadcast two different inputs, which can not occur. Thus for any other value z' an honest player can only observe strictly less than $\frac{|com_1|}{3}$ $vote(z')$ messages from players in com_1. On the other hand, if p observed $vote(z)$ from more than $\frac{2}{3}$ of the members of com_1, this implies, together with Observation 12, that more than $\frac{1}{3}$ of the honest members of com_1 broadcast $vote(z)$, which means *all honest players* have seen $vote(z)$ messages from more than $\frac{1}{3}$ of the members of com_1. Combined, this means that all honest players have observed more $vote(z)$ messages from players in com_1 than for any other value, and thus output either $adopt(z)$ or $commit(z)$, as required. ◀

We now proceed to designing a protocol for the CO task.

Algorithm 2 CO.

We employ the following conciliator protocol executed by all honest players p with input z_p.

1. If $t \in [0, 2]$, run according to CA protocol with input value z.
2. If $r = 3$, if $p = \ell_3$ or $p \in com_3$, broadcast CA output.
3. if $r \geq 4$, then:
   a. If p received $commit(z)$ from more than $\frac{1}{3}$ of the players in com_3 for some value z, then p outputs z.
   b. Else, if p received $adopt(z)$ or $commit(z)$ for some value z from ℓ_3, then output z
   c. Else, output z_p.

We now proceed to prove the correctness of the above procedure.

▶ **Lemma 14.** *For any constant $\epsilon > 0$, except for with* $\mathsf{negl}(\lambda)$ *probability, Algorithm 2 solves the* CO *task whenever* $3f_t + 1 < (1-\epsilon)n$.

Proof. Termination is clear from the behavior of the protocol. For Validity, if z is the input of all honest players, then by the Validity of CA, we get that by round 3, all honest players output $commit(z)$. For probabilistic agreement, note that if no honest player outputs according to item (a), then the property holds as the leader is honest w.p. at least $\frac{2}{3}$. Otherwise, Let p be an honest player that output according to item (a), which implies that p observed more than $\frac{1}{3}$ fraction of $commit(z)$ messages for the same value z from the players in com_3. In particular this implies that at least one of those players is honest. Thus by the Agreement property CA, we have that all honest players output either $commit(z)$ or $adopt(z)$. Thus no player in com_3 sends $commit(z')$ for any value $z \neq z'$. In particular this means that no honest player views more than a $\frac{1}{3}$ fraction of $commit(z')$ for any $z' = z$. Thus, all honest players that output according to item (a) agree. Denote that value by z. Now note that w.p. at least $\frac{2}{3}$, ℓ_3 is honest, and if this is the case, then ℓ_3 also output either $commit(z)$ or $adopt(z)$, by the agreement property of CA, and all other honest players output z according to item b in that case. Thus, w.p. at least $\frac{2}{3}$, all honest players agree on the output, as required. ◀

We denote by L_c, L_{ca} the number of rounds required to run the CO, CA tasks, respectively. We can now describe the generic BA protocol we employ. The following protocol is executed by every honest player p with input z.

Algorithm 3 BA protocol framework.

1. For $i = 0, ..., D$:
   - **a.** If $t \in [(L_c + L_{ca})i, (L_c + L_{ca})i + L_c]$, run as in $\mathsf{CO}[i]$ with input being the output of $\mathsf{CA}[i-1]$ or z if $i = 0$.
   - **b.** If $r \in [(L_c + L_{ca})i + (L_c + 1), (L_c + L_{ca} + 1)i]$ run as in $\mathsf{CA}[i]$ with input being p's output in $\mathsf{CO}[i]$.
   - **c.** let o be the output of $\mathsf{CA}[i]$.
     - **i.** If $o = commit(v)$ for some value, then output v as BA decision. Participate in the protocol for one more iteration with inputs to all subroutines being fixed to v.
     - **ii.** Else, move on to $i + 1$.

With the above instantiations of CA and CO in mind, we now prove that the above protocol proves Lemma 9.

Proof. For Termination, consider an iteration i of the protocol above. By probabilistic agreement of CO, we have that w.p. at least $\frac{2}{3}$, all honest players agree on the output of $\mathsf{CO}[i]$. Denote it by z. Conditioned on agreement, Validity of CA guarantees that all honest players output $commit(z)$, and thus they all terminate at the end of $\mathsf{CA}[i]$. Thus for every iteration i, w.p. at least $\frac{2}{3}$, all honest players terminate at the end of iteration $i + 1$. Thus, w.h.p. all honest players terminate after $O(\log n)$ iterations. For Validity, consider the case where all players have the same input z. The Validity properties of both CO and CA imply that at end of iteration 1, all honest players output $commit(z)$ from $\mathsf{CA}[1]$ and output z, as required. For Agreement, Let p be the first honest player to output, with output z. I.e. there exists an i such that p output $commit(z)$ from $\mathsf{CA}[z]$, and no honest player has output $commit(z')$ for any z at any iteration $i^* < i$. In particular, this implies, by the consistency

of CA, that all other honest players output either $commit(z)$ or $adopt(z)$ from $\mathsf{CA}[i]$. Which implies that all honest players enter iteration $i+1$ with input z. The Validity of an iteration, proven above, implies thus that by the end of iteration $i+1$, all honest players output z.

Combining the fact that for every iteration i, w.p. at least $\frac{2}{3}$, all honest players terminate at the end of iteration $i+1$ with the small size of com_t for all t, we obtain that the protocol halts in $O(1)$ rounds in expectation, and has $O(\lambda)$ uniform communication complexity, in expectation, as required. ◀

Note that we have essentially proved that Algorithm 3 solves BA whenever the number of honest participants in each round exceeds a $\frac{2}{3}$ fraction. Furthermore, it has expected latency of $O(1)$.

Nakamoto Consensus. Note that Nakamoto consensus [32], also gives an efficient protocol with adaptive security. In each round, a leader is elected from a beacon. The leader adds a block to a chain, and consensus is reached on a prefix of the current longest chain, i.e. on blocks that are k-deep in the longest chain. This implies that in order to agree on a single block with $1 - exp(-\lambda)$ confidence, it must be $k = O(\lambda)$ deep in the longest chain. Thus, the beacon needs to emit $k \cdot \log(n)$ random bits to reach consensus. We note that the classic implementation of Nakamoto does not satisfy the validity condition of BA. This is easily remedied with a single invocation of CA prior to the initiation of the Nakamoto protocol. While CA requires $\Omega(n^2)$ communication, note that our lower bound (Theorem 7) applies even when the protocol has $O(1)$ initial rounds of high communication.

5.2 Adaptive security and low beacon entropy

Next, we showcase a protocol that is secure against an adaptive adversary and does not satisfy Definition 4 (i.e., has low beacon entropy). This, of course, as per Theorem 7, implies that this protocol has to have high communication. Specifically, all n parties send messages in every round. Specifically, we prove the following lemma.

▶ **Lemma 15.** *For every $\rho < \frac{1}{3}$, and assuming a randomness beacon there exists a ρ-secure consensus protocol against an adaptive adversary, with $O(1)$ expected latency. Furthermore, the protocol does not satisfy Definition 4.*

We once again in this protocol make no use of CRS, as we aim to be secure against an adaptive adversary. The protocol Π is going to follow the same structure of Algorithm 3, with the following modifications.

- All players participate in every round of Algorithm 3. In other words, $com_t = [n]$ for all rounds t. In particular, the randomness beacon is not used for committee sampling.
- The random beacon is still being used to sample a uniformly random leader during the CO task. That is the only use of the randomness beacon. In other words, the output of the randomness beacon at round t, denoted by seed_t, contains only the identity of a single player ℓ_t which is the chosen leader for round t.

Correctness of the protocol and its security against an adaptive adversary are immediate from the proof of correctness for Algorithm 3, and the adversary being ρ-bounded for $\rho < \frac{1}{3}$. The last item to prove is the following.

▷ Claim 16. Algorithm 3 when implemented without committees, satisfies that there exists a constant $c > 0$ such that

$$\sum_{t=0}^{\infty} H_\infty(\mathrm{Tr}^t_{E_\mathcal{A}} \mid \{\mathrm{Tr}^i_{E_\mathcal{A}} \mid 0 \le i < t\}) \le c \log n$$

Proof. Note that the only source of entropy in the modified protocol is the leader election in the CO subroutine. Besides that, all content of all messages is a deterministic function of the inputs of the players. Furthermore, we have that for each iteration i, w.p. at least $\frac{2}{3}$, all honest players terminate by the end of iteration $i+1$. Thus, We get that for any adversary $\mathcal{A}$ and any execution E, and for every iteration $i > 0$ w.p. at least $1 - \frac{1}{3^{i-1}}$, $I^t_{E_\mathcal{A}} = \emptyset$ where t is any round during iteration i. In particular we then get that the total min entropy of the transcript of the protocol Π during iteration i, for all $i > 1$ is upper bounded by $\log(\frac{3^{i-1}}{3^{i-1}-1})$. For $i = 0, 1$, the transcript is determined by the identity of the random leader, hence both of these iterations provide $\log n$ min entropy each. In total, we get that

$$\sum_{t=0}^{\infty} H_\infty(I^t_{E_\mathcal{A}} | \{Tr^i_{E_\mathcal{A}} \mid 0 \le i < t\}, \mathsf{CRS}) < 2\log n + \sum_{t=2}^{\infty} \log(\frac{3^{t-1}}{3^{t-1}-1}) < 2\log n + 1$$

as required. ◁

5.3 Efficiency and low beacon entropy

In this section, we showcase a protocol achieving both low communication and low Randomness beacon entropy usage, which in particular implies that it doesn't have global leader unpredictability (see Definition 4). This is the only protocol in which we make use of the CRS. Specifically, we run Algorithm 3 but instead of using seed_t to select committees and a leader for $\mathsf{CO}[i]$, all the information about the committees and leaders for each iteration are taken from CRS. I.e. the CRS is interpreted as $(\mathsf{seed}_1, \mathsf{seed}_2, \ldots)$, where $\mathsf{seed}_t = (com_t, \ell_t)$ with $\ell_t \in [n]$ and $com_t \subseteq [n]$ being a subset of players of size $O(\lambda)$. We assume that CRS is sufficiently long to encode such information. We recall that a static adversary makes its choice of corruption *before* observing the CRS. Formally, we prove the following.

▶ **Lemma 17.** *For every $\epsilon > 0$ and $\rho < \frac{1-\epsilon}{3}$, and assuming a* CRS*, there exists a ρ-secure consensus protocol against a static adversary with $O(1)$ expected latency and $O(\lambda)$ uniform communication complexity, in expectation.*

For simplicity, we abuse notation and refer to ℓ_t, com_t for the purposes of this lemma also as the leader and the committee of round t as described in the CRS.

▶ **Observation 18.** *Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be a static adversary. Then for all t it holds that*

$$\Pr\left[|\mathcal{A}_0(\Pi, t) \cap com_t| \ge \frac{|com_t|}{3}\right] = \mathsf{negl}(\lambda)$$

when com_t is taken from the CRS*.*

The observation follows from the same arguments as Observation 12.

Proof. The proof of the lemma follows from the correctness of Algorithm 3 whenever the honest fraction of participating parties in every round exceeds $\frac{2}{3}$. Algorithm 3 was shown to have $O(1)$ expected latency and by the size of com_t for all t we get that the communication complexity of the protocol is uniform $O(\lambda)$ in expectation, as required. ◀

6 Conclusion

Our motivation in this paper was to characterize the space of what is possible in modern consensus protocols. Although prior work has examined many desirable properties of consensus, the relationship between randomness, communication efficiency, and adaptive

security lacked a rigorous understanding. These properties have become especially desirable in modern-day consensus protocols, yet their theoretical foundations have lagged behind. We further our understanding of the consensus landscape by showing a trilemma: *Efficient* consensus with *adaptive security* requires $\Omega(\log n)$ bits of *beacon entropy*. On the positive side, we provide protocols that achieve any pair of these properties.

Implications for practice. In modern blockchains, adaptive security is especially desirable to prevent bribery attacks or other attacks where nodes are temporarily compromised. In practice, modern blockchains often use randomness beacons to unpredictably elect leaders or committees. However, these randomness beacons come at a cost and add design and implementation complexity. Our work shows that efficient and adaptively secure protocols *require* beacons; this cost is in some sense unavoidable.

A natural next question is exactly how much randomness is needed from the beacon. Our possibility results show that the amount of entropy required is relatively low, growing only logarithmically in the number of parties. Therefore, using a beacon is unavoidable, yet this beacon can be relatively lightweight.

Extensions to other settings. In this work, we have considered a computationally unbounded adversary and an information-theoretic setting. This choice mainly served to maintain the simplicity and clarity of the exposition. A natural alternative model is the PKI setting, with computationally bounded adversaries. Our positive results trivially extend to this setting, as information-theoretic security implies security against a computationally bounded adversary. Extending our lower bound is slightly more subtle, though we expect it to hold for a notion of pseudorandomness rather than statistical beacon entropy. We leave formalizing this analogous bound as an interesting direction for future work.

In this work, we restricted our attention to a special variety of low-communication protocols, specifically ones where the set of parties speaking in each round is small. This notion of communication efficiency is desirable in practice, and several protocols use a committee subsampling approach to this end (e.g., Algorand, Ethereum). However, one might also be interested in other notions of communication efficiency, for example the setting where communication is simply measured by the total number of bits exchanged by honest parties. An open question is whether our results stand in this alternate setting.

References

1 Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. *Distributed Comput.*, 36(1):3–28, 2023. doi:10.1007/S00446-022-00428-8.

2 Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous Byzantine Agreement with Expected O(1) Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience. In *Financial Crypto*, 2019.

3 Mohamad Ahmadi, Abdolhamid Ghodselahi, Fabian Kuhn, and Anisur Rahaman Molla. The cost of global broadcast in dynamic radio networks. *Theor. Comput. Sci.*, 806:363–387, 2020. doi:10.1016/J.TCS.2019.07.013.

4 Mohamad Ahmadi and Fabian Kuhn. Multi-message broadcast in dynamic radio networks. In *ALGOSENSORS*, 2016.

5 Kaya Alpturer and S. Matthew Weinberg. Optimal RANDAO manipulation in ethereum. In *AFT*, 2024. doi:10.4230/LIPIcs.AFT.2024.10.

6 Ziv Bar-Joseph and Michael Ben-Or. A tight lower bound for randomized synchronous consensus. In *PODC*. ACM, 1998. doi:10.1145/277697.277733.

7 Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable Delay Functions. In *CRYPTO*, 2018.

8 Elette Boyle, Ran Cohen, and Aarushi Goel. Breaking the O(n)-bit Barrier: Byzantine Agreement with Polylog Bits Per Party. In *PODC*. ACM, 2021. doi:10.1145/3465084.3467897.

9 Nicolas Braud-Santoni, Rachid Guerraoui, and Florian Huc. Fast byzantine agreement. In *PODC*. ACM, 2013. doi:10.1145/2484239.2484243.

10 Miguel Castro, Barbara Liskov, et al. Practical Byzantine Fault Tolerance. In *OSDI*, 1999.

11 Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. doi:10.1016/J.TCS.2019.02.001.

12 Kevin Choi, Aathira Manoj, and Joseph Bonneau. Sok: Distributed randomness beacons. In *IEEE Security & Privacy*, 2023.

13 Benny Chor, Michael Merritt, and David B. Shmoys. Simple constant-time consensus protocols in realistic failure models. *J. ACM*, 36(3):591–614, 1989. doi:10.1145/65950.65956.

14 Francesco D'Amato and Luca Zanolini. A simple single slot finality protocol for ethereum. In *ESORICS*, 2023.

15 Sourav Das, Sisi Duan, Shengqi Liu, Atsuki Momose, Ling Ren, and Victor Shoup. Asynchronous Consensus without Trusted Setup or Public-Key Cryptography. In *ACM CCS*, 2024.

16 Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *Eurocrypt*, 2018.

17 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, 1985. doi:10.1145/2455.214112.

18 Matthias Fitzi, Chen-Da Liu-Zhang, and Julian Loss. A new way to achieve round-efficient byzantine agreement. In *PODC*. ACM, 2021. doi:10.1145/3465084.3467907.

19 Eli Gafni and Giuliano Losa. Brief Announcement: Byzantine Consensus Under Dynamic Participation with a Well-Behaved Majority. In *DISC*, 2023. doi:10.4230/LIPIcs.DISC.2023.41.

20 Juan A. Garay and Aggelos Kiayias. SoK: A Consensus Taxonomy in the Blockchain Era. In *CT-RSA*, 2020.

21 Diana Ghinea, Vipul Goyal, and Chen-Da Liu-Zhang. Round-optimal byzantine agreement. In *Eurocrypt*, 2022.

22 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, 2017.

23 Mohammad Hajiaghayi, Dariusz R. Kowalski, and Jan Olkowski. Nearly-optimal consensus tolerating adaptive omissions: Why a lot of randomness is needed? In *PODC*. ACM, 2024. doi:10.1145/3662158.3662826.

24 Alireza Kavousi, Zhipeng Wang, and Philipp Jovanovic. SoK: Public Randomness. In *Euro S&P*, 2024.

25 Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In *ICDCN*, 2011.

26 Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. *J. ACM*, 58(4):18:1–18:24, 2011. doi:10.1145/1989727.1989732.

27 Marek Klonowski, Dariusz R. Kowalski, and Jaroslaw Mirek. Ordered and delayed adversaries and how to work against them on a shared channel. *Distributed Comput.*, 32(5):379–403, 2019. doi:10.1007/S00446-018-0341-7.

28 Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

29 Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Paper 2015/366, 2015. URL: https://eprint.iacr.org/2015/366.

30 Julian Loss and Gilad Stern. Zombies and ghosts: Optimal byzantine agreement in the presence of omission faults. In *TCC (4)*, volume 14372 of *Lecture Notes in Computer Science*, pages 395–421. Springer, 2023. doi:10.1007/978-3-031-48624-1_15.

31 Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with t < n/3, o(n2) messages, and O(1) expected time. *J. ACM*, 62(4):31:1–31:21, 2015. doi:10.1145/2785953.

32 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

33 Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980. doi:10.1145/322186.322188.

34 Michael O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 1983.

35 Mayank Raikwar and Danilo Gligoroski. SoK: Decentralized randomness beacon protocols. In *Australasian Conference on Information Security and Privacy*, 2022.

36 Peter Robinson, Christian Scheideler, and Alexander Setzer. Breaking the $\tilde{\Omega}(\sqrt{n})$ barrier: Fast consensus under a late adversary. In *SPAA*. ACM, 2018.

37 Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *IEEE Security & Privacy*, 2017.

Composable Byzantine Agreements with Reorder Attacks

Jing Chen* ✉
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Jin Dong ✉
Beijing Academy of Blockchain and Edge Computing (BABEC), China

Jichen Li ✉
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Xuanzhi Xia ✉
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Wentao Zhou ✉
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Abstract

Byzantine agreement (BA) is a foundational building block in distributed systems that has been extensively studied for decades. With the growing demand for protocol composition in practice, the security analysis of BA protocols under multi-instance executions has attracted increasing attention. However, most existing adversary models focus solely on party corruption and neglect important threats posed by adversarial manipulations of communication channels in the network. Through channel attacks, messages can be reordered across multiple executions and lead to violations of the protocol's security guarantees, without the participating parties being corrupted.

In this work, we present the first adversary model that combines party corruption and channel attacks. Based on this model, we establish new security thresholds for Byzantine agreement under parallel and concurrent compositions, supported by complementary impossibility and possibility results that match each other to form a tight bound. For the impossibility result, we show that even authenticated Byzantine agreement protocols cannot be secure under parallel composition when $n \leq 3t$ or $n \leq 2c + 2t + 1$, where t and c denote the number of corrupted parties and communication channels, respectively. For the possibility result, we prove the existence of secure protocols for unauthenticated Byzantine agreement under parallel and concurrent composition, when $n > 3t$ and $n > 2c + 2t + 1$. More specifically, we provide a general black-box compiler that transforms any single-instance secure BA protocol into one that is secure under parallel executions, and we provide a non-black-box construction for concurrent compositions.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Security and privacy → Security protocols

Keywords and phrases Byzantine agreement, protocol composition, channel reorder attack, security threshold

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.13

Funding This work is partially supported by Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing.

Acknowledgements The authors would like to thank several anonymous reviewers for their valuable comments.

*Corresponding author.

© Jing Chen, Jin Dong, Jichen Li, Xuanzhi Xia, and Wentao Zhou;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

The Byzantine agreement (BA) problem [46] is a foundational challenge in distributed systems, concerned with achieving consensus among parties even in the presence of failures or malicious behavior. This problem has been extensively studied under various models, leading to significant theoretical breakthroughs, including both impossibility results [43, 26] and the development of innovative protocols [24, 9, 47, 39]. As a core mechanism for achieving distributed consistency and fault tolerance, Byzantine agreement underpins a wide range of applications, including blockchain technologies [12, 1, 48, 55, 41, 56], secure multiparty computation [54, 35, 4, 11, 17, 28, 33] and diverse distributed services [30, 45, 2].

Much of the early research on the Byzantine agreement problem was conducted under a classical model where a network of n parties, each holding an initial input value, communicates over reliable synchronous channels. In this model, faults are restricted to the parties themselves: up to t parties may be corrupted by an adversary, but the communication network is assumed to be secure – messages cannot be forged, altered, or dropped by the adversary once sent by an honest party. The standard goals of a BA protocol in this setting are: (1) *agreement* – all honest parties eventually terminate and output the same value; and (2) *validity* – if all honest parties start with the same input, that value is the output. Studies show that in unauthenticated settings, BA is achievable if and only if $t < \frac{n}{3}$ [43, 46], while in authenticated settings, where digital signatures prevent forgery [50, 36], BA can be achieved with any number of corruption $t < \frac{n}{2}$ [27]. The expected constant-round protocols with optimal resiliency in the two settings were then constructed in [23] and [39], respectively.

However, the aforementioned studies primarily focus on single-instance executions of Byzantine protocols, leaving the critical issue of protocol composition unaddressed. In modern distributed environments, such as sharded blockchains, multiparty computation frameworks, or cross-chain settings, multiple consensus protocols may be composed sequentially, in parallel, or concurrently. In sequential composition, each protocol instance begins only after the previous one has been completed. In parallel composition, all instances are initiated at the same time and proceed with their steps aligned with each other. The most general model, concurrent composition, grants the adversary full control over the start times and execution rates of different instances.

The shift from single-instance to multi-instance execution introduces new problems, demanding a re-examination of existing assumptions about fault models and protocol security. In particular, [44] showed that, without a unique and common session identifier for every execution of the protocol, no authenticated BA protocol can remain secure even under just two parallel executions when corruption exceeds $n/3$. In other words, authenticated BA performs as poorly as unauthenticated BA under parallel composition. Common session IDs can sometimes be achieved via a bootstrap phase, but in this work we would like to pursue the power of stateless composition without relying on such a bootstrap, so that the resulting BA protocols can be directly applied whenever compositional executions are needed.

The challenge with protocol composition is that the adversary's power is amplified, and in our study we will amplify it even further. Indeed, a key limitation of the classical model is its implicit assumption that adversarial power is confined to corrupting parties, while communication channels remain trustworthy in the sense that messages sent between honest parties will be received correctly. This abstraction overlooks more realistic adversarial behaviors, such as man-in-the-middle or message-reordering attacks, which target the network rather than the parties themselves, especially when protocol composition is concerned. By carefully disrupting or manipulating some communication channels between multiple protocol instances, an adversary may induce cross-protocol interference, leading to violations of agreement or validity that would not occur in isolated executions.

1.1 Adversary Model

To address the limitations of classical adversary models, we introduce an extended adversarial framework tailored to protocol composition environments. In this model, the adversary retains the classical ability to corrupt up to t parties, allowing them to deviate arbitrarily from the protocol. In addition, channels can also be corrupted, without sending and receiving parties being corrupted or even aware of the channel attack. This reflects realistic attack approaches in networks where multiple protocols share the underlying network infrastructure.

In the real world, the adversary may exploit man-in-the-middle attacks, such as ALPACA [7], to redirect the messages and bypass the security guarantees of TLS and application layer protocols, resulting in cross-protocol attacks that rearrange the messages from different protocols without breaking the signature scheme.

Such a setting captures the inherent interdependence of parallel or concurrently running consensus protocols, where messages from different instances may traverse overlapping physical or logical channels. Through reorder attacks, the adversary can introduce inter-instance inconsistencies that undermine global agreement guarantees, even if each protocol remains individually secure under classical assumptions.

This hybrid adversary model – combining party corruption with channel attacks – introduces new theoretical challenges. In particular, these two kinds of attacks are coupled together, thus determining the exact resilience thresholds under simultaneous control of t parties and c channels requires a fresh analytical approach that goes beyond traditional BA models.

1.2 Our Results

Our main results establish a new tight security threshold under the new adversary model. We first present the impossibility result showing that even in the authenticated setting, BA under parallel (and concurrent) composition is not achievable if $n \leq 3t$ or $n \leq 2c + 2t + 1$, where t is the number of corrupted parties and c is the number of channels that the adversary can manipulate.

Intuitively, even if a majority of parties are honest, agreement may still fail if the adversary can confuse one party, making it unable to tell which specific protocol instance it is participating in. For example, when two protocols are running in parallel, an attacker can target a particular party and swap the messages it is supposed to receive in the two protocols, causing its result to differ from that of the other honest parties. Therefore, it is not enough to simply bound the number of corrupted parties – one must also ensure that the honest parties maintain a sufficient number of communication with parties in the same protocol. This leads to a trade-off between party corruption and channel interference, and motivates the combined threshold $n > 2c + 2t + 1$ as the condition necessary to preserve the communication advantage of the honest majority.

We further prove possibility results under compositional executions, showing that the condition $n > 2c+2t+1$ along with the condition $n > 3t$ in classical setting (which is the tight bound when only party corruption is considered) is sufficient even in the unauthenticated setting. That is, when the combined adversarial power satisfies both conditions, it is possible to design a BA protocol that remains secure under arbitrary parallel and concurrent executions. These results complement our impossibility result, forming a tight security threshold: $n > \max\{2c + 2t + 1, 3t\}$ is both necessary and sufficient for protocol composition in our adversary model.

Noticeably, for parallel composition, we provide a general black-box compiler that transforms any single-instance secure BA protocol into one that is secure under parallel executions. A non-black-box construction for concurrent composition is provided, which is secure under asynchronous networks. By carefully structuring message sending and leveraging the honest majority's residual communication connectivity, our protocol prevents cross-instance interference and ensures that no single execution is compromised due to shared network vulnerabilities.

In conclusion, our work provides the first formal characterization of security for BA in the presence of combined party corruption and channel attacks under protocol composition, filling a critical gap in the theoretical understanding of consensus under adversarial network interference. By introducing a unified adversary model and establishing the tight security threshold, we extend the classical BA framework to more accurately reflect the complication of modern distributed systems.

2 Related Work

Composition of Byzantine Agreements. In a stateless model where protocols do not have distinguishing IDs, which is also considered by our work, [44] studied the sequential, parallel, and concurrent composition of authenticated BA protocols and presented the impossibility result as mentioned in the introduction. It also constructed secure randomized protocols under sequential composition. Without cryptographic primitives, [3] considered concurrently secure BAs with expected-constant round in both synchronous and asynchronous networks with optimal resiliency in the classical model (i.e., $n > 3t$). However, as pointed out by [16], its security proof had some subtle issues regarding, e.g., the use of oblivious leader election.

Both [3] and [16] are not stateless: they assume a unique and common session ID for every execution of the protocol. As pointed out by [44], concurrent composition of any number of executions in this case is possible. Indeed, the focus of [3] and [16] was to construct expected-constant round protocols that are concurrently composible. Similarly, [37] assumed unique session IDs but considered an adversary who can corrupt a player in some but not all parallel executions.

Moreover, [16] followed the Universal Composability (UC) framework [8], which means the protocol is secure under arbitrary composition with itself and other cryptographic protocols in an adversarially controlled manner. Focusing on the asynchronous setting, [16] considered concurrent BA and presented the first information-theoretic multi-valued oblivious common coin (OCC) protocol with optimal resiliency. It further provided a modularized protocol for round-preserving parallel composition of BA that was simpler than the construction in [3].

Computation and Communication Complexity. An important line of work considers the complexity of a single execution of a Byzantine agreement. A body of foundational studies, in particular [19, 20, 9], proposed Byzantine agreement protocols that required a quadratic number of messages. More specifically, the $O(n^2)$ communication complexity necessitates nearly all-to-all communication between parties. Subsequently, for *randomized Byzantine protocols*, a seminal work [40] proposed a protocol where each party speaks to only $\tilde{O}(1)$ other parties, though their protocol achieved almost-everywhere agreement [21] rather than agreement, where $1 - O(\log^{-1} n)$ fraction of the parties reach agreement. Subsequent studies have sought to bridge this gap by achieving agreement from almost-everywhere agreement. To the best of our knowledge, so far the best communication complexity achievable for agreement in authenticated setting is $\tilde{O}(1)$ rounds with $\tilde{O}(\sqrt{n})$ communication and computation [32]. With enhanced cryptographic primitives such as LWE, this can be improved to $\tilde{O}(1)$ rounds and $poly(\lambda, \log n)$ communication [5, 25], where λ is the security parameter.

However, for *deterministic Byzantine protocols*, [18] established that achieving agreement inherently requires at least $\Omega(t^2)$ communication. Furthermore, [14] extended this lower bound to other generalized validity definitions (e.g., weak validity), demonstrating that any well-defined Byzantine agreement variant necessitates $\Omega(t^2)$ communication. Thus, deterministic Byzantine agreements incur fundamentally more overhead than randomized ones.

Different from this line of works that focused on single-instance executions of BA protocols, our work investigates compositional executions of BA protocols. Although our protocols are not necessarily optimal in terms of their complexity, our black-box compiler protocol achieves polynomial communication and computation. Further improving the complexity of our protocols and proving complexity lower-bounds in our model are interesting open problems.

Network Attacks. Many prior works on network security have shown that channel attacks (e.g., man-in-the-middle attacks) without corrupting participating parties exist in a wide range of scenarios and pose a significant threat in real applications. By delaying, forging, dropping, or redirecting, attackers can break SSL/TLS certificate validation in many critical software applications [34, 51], cause security issues for HTTPS and its certificate trust model [15], inflict devastating damage on private and consortium blockchains [22], and so on.

Finally, it's worth noticing that for transaction messages in DeFi, attackers (miners) can gain Miner Extracted Value (MEV) through transaction reordering attacks such as those studied in [53], while [38] indicated that currently no mitigation schemes can fully solve this problem. For a fully connected large-scale network, [52] analyzed Simple Majority Protocol (SMP) under probabilistic message loss and proved that it can reach consensus in three rounds of communication with probability approaching 1. However, those are within the same consensus protocol and the role of channel attacks in the composition of Byzantine agreements has not been explored yet.

3 Our Model

3.1 Byzantine Agreement

We consider the problem of *Byzantine Agreement* where $P = \{P_1, \ldots, P_n\}$ is a set of n parties, and P is common knowledge among all parties. The protocol Π is executed m times in parallel (or concurrently), where we denote the k-th instance by Π_k. Each party P_i in each instance Π_k is formally modeled by an interactive (randomized) Turing machine ITM_k^i, with an input tape containing its initial value, an output tape for its final output, a random tape, and $n-1$ pairs of communication tapes (input/output) corresponding to the other $n-1$ parties in the same Π_k, denoted by $Input_{i,j}^k$ and $Output_{i,j}^k$ for each $P_j \neq P_i$. More specifically, $Output_{i,j}^k$ contains messages that i sends to j, and $Input_{i,j}^k$ contains messages that i receives from j. Under parallel or concurrent composition, a party locally runs m copies of such interactive Turing machines simultaneously, each corresponding to a particular protocol instance.

Network and Communication. In most parts of the paper except Section 6, the network is synchronous and point-to-point: communication proceeds in rounds, each consisting of a *send* phase followed by a *receive* phase. In the send phase, a party writes the sending message onto the corresponding communication output tape. Then, in the receive phase, the message is written onto the target party's input tape under the same protocol instance.

More specifically, the communication channel between parties P_i and P_j, denoted by C_{ij}, is the set of all input and output tapes between the two parties, for all m protocol instances. By symmetry, C_{ji} means the same thing as C_{ij} and an attack on one is also an attack on the other. In an uncompromised channel C_{ij}, messages originating from $Output^k_{i,j}$ are delivered exclusively to $Input^k_{j,i}$ for all $k \in \{1, ..., m\}$, and vice versa.

Stateless Setup. In stateless compositions of BA protocols, there is no common session identifier for the protocol instances. The indices of different protocol instances are for discussion purposes and are not accessible by the parties. Indeed, each Turing machine ITM^i_k has access to its tapes but doesn't know the global index k. Internally, each party may have individual numbering for its Turing machines, but those numbers may not be consistent among different parties.

For authenticated Byzantine Agreement (ABA), a *common setup* is shared across all protocol instances. In particular, all parties undergo a one-time trusted preprocessing phase that generates cryptographic primitives, which are then used consistently across all protocol instances. Formally, each interactive Turing machine has a read-only *setup tape,* and all Turing machines run by the same party share this setup tape. For example, this tape may contain the secret key of a signature scheme used by the party, as well as the public verification keys of other parties. For unauthenticated Byzantine Agreement, no cryptographic primitives are used and there is no setup tape or a setup phase.

Composition of BA Protocols. We consider two types of protocol composition: parallel composition and concurrent composition. In parallel composition, all protocols start simultaneously, and each round has one time step across all protocol instances. In concurrent composition, the adversary can independently control the start time and the time steps of each round of each protocol instance.

Adversarial Model. We consider an adversary $\mathcal{A}(t, c)$ operating in the *point-to-point full information* setting in a parallel or concurrent execution of protocol instances $\Pi_1, \Pi_2, ..., \Pi_m$. Its capabilities include:

- *Corruption:* The adversary can corrupt up to t parties and chooses which parties to corrupt before any protocol instance begins. This is called static corruption.
- *Channel Reorder Attack:* The adversary can attack up to c channels between honest parties. When C_{ij} is under attack, the adversary is allowed to redirect messages on the corresponding input tapes for different protocol instances and in both directions. See below for a formal definition, which is also illustrated in Figure 1.

▶ **Definition 1** (Channel Reorder Attack). *For any two honest parties P_i and P_j, a channel reorder attack on C_{ij} allows the adversary to arbitrarily redirect messages between different protocol instances and in both directions. Specifically, the adversary can deliver any message $msg \in Output^{k_1}_{x,y}$ to any $Input^{k_2}_{y,x}$, where $k_1, k_2 \in \{1, ..., m\}$ and $\{x, y\} = \{i, j\}$.*

- *Rushing:* The adversary is rushing, meaning it can observe all messages sent by honest parties in a round before deciding the messages to send from corrupted parties and message reordering in the same round.
- *Computation Power:* For our impossibility results, which apply to authenticated BA protocols, we assume that all parties (honest and adversarial) are probabilistic polynomial-time (PPT). In contrast, for our positive results, which apply to unauthenticated BA protocols, we allow computationally unbounded adversaries.

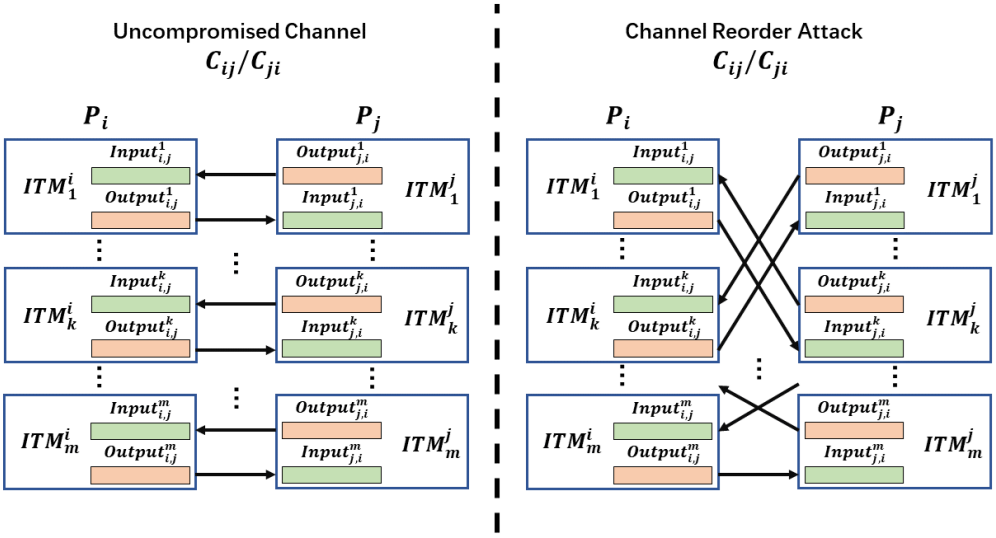


Figure 1 Communication diagrams for channel C_{ij}/C_{ji}, with and without channel reorder attack.

For this extended adversary model, a protocol Π is considered a solution to the BA problem if it satisfies the following properties:

▶ **Definition 2** (Byzantine Agreement). *Let $\mathcal{A}(t, c)$ be an adversary who can corrupt up to t parties and c channels. A protocol Π solves the Byzantine Agreement problem if for any adversary $\mathcal{A}(t, c)$, the following two properties hold:*

- Agreement*: All honest parties eventually terminate and output the same value.*
- Validity*: If all honest parties begin with the same initial value v, then their output values must all be v.*

3.2 Composition Security

Composition security plays a pivotal role in analyzing the robustness of cryptographic protocols executed within larger systems or concurrently with other protocol instances, rather than in isolation. Our work adopts the definition of composition security as presented in [44], with the notable distinction that we explicitly consider attacks on communication channels.

▶ **Definition 3** (Composition Security of ABA). *Let $P_1, ..., P_n$ be parties for an ABA protocol Π. We say that Π remains secure under m instances of parallel (or concurrent) executions, if for every PPT adversary $\mathcal{A}(t, c)$, the requirement of BA holds for every individual instance of Π within the following procedure:*

1. Setup Phase: *A single, global trusted setup phase is performed once. This phase generates setup strings $s_1, \ldots, s_n$ for parties $P_1, \ldots, P_n$, which are then consistently used across all m protocol instances.*
2. Static Corruption: *The adversary $\mathcal{A}(t, c)$ statically corrupts up to t parties, gaining full control over their actions. Additionally, the adversary can attack up to c communication channels C_{ij} between honest parties, enabling channel reorder attacks described in Definition 1.*
3. Parallel (Concurrent) Executions: *The following procedure is repeated in parallel (or concurrently) for each protocol instance, until the adversary halts:*
   - *The adversary specifies the initial input values for all parties.*
   - *Each party uniformly generates the content of its random tape for the execution in this protocol instance.*

- *All parties are invoked for the execution of* Π.
- *The adversary determines the messages sent by the corrupted parties, while honest parties strictly adhere to the protocol* Π.
- *Messages transmitted over attacked channels are subject to the adversary's reordering on corresponding input tapes, whereas messages on unattacked channels between honest parties are delivered reliably.*

For unauthenticated Byzantine Agreement protocols, the definition of composition security is analogous, with the difference that no setup phase or cryptographic primitives are needed.

4 Impossibility Result

Authentication allows protocols to achieve higher fault tolerance in the stateless execution model, tolerating an arbitrary number of corrupted parties. However, this intuition breaks down in composed scenarios. When multiple executions occur, a signature from a Party P on message x does not prove that P signed x in the "current" specific execution. An adversary can "borrow" signed messages from one execution and use them in another, rendering the public-key infrastructure "useless" in distinguishing execution contexts.

▶ **Theorem 4.** *There does not exist a protocol for stateless authenticated Byzantine Agreement in a synchronous network with n parties that remains secure under parallel composition (even for just two executions) against an adversary $\mathcal{A}(t, c)$, provided that (1) $n \leq 3t$, or (2) $n \leq 2c + 2t + 1$.*

Proof. For condition (1), the result follows directly from Theorem 1 in [44].

We prove the result for condition (2) by contradiction. Assume that there exists an authenticated Byzantine Agreement protocol Π that remains secure under two parallel executions against an adversary $\mathcal{A}(t, c)$ with $n \leq 2c + 2t + 1$. We assume $c > 0$, otherwise we have $2t + 1 \geq n$ and it can be reduced to condition (1).

We consider two independent executions of Π: Let $A_1, A_2, \cdots, A_n$ and $B_1, \cdots, B_n$ be independent copies of n parties participating in protocol Π. The independent copies mean that for each i, A_i and B_i are the same party that runs in two different parallel executions of Π. For the purpose of the proof below, we denote the set $T = \{A_2, \cdots, A_{\lfloor \frac{n+1}{2} \rfloor}\}$, $\widetilde{T} = \{B_2, \cdots, B_{\lfloor \frac{n+1}{2} \rfloor}\}$, $H = \{A_{\lfloor \frac{n+3}{2} \rfloor}, \cdots, A_n\}$, and $\widetilde{H} = \{B_{\lfloor \frac{n+3}{2} \rfloor}, \cdots, B_n\}$. Note that we have $n \leq 2c + 2t + 1$, so we have $c + t \geq |T| = |\widetilde{T}|$ and $c + t \geq |H| = |\widetilde{H}|$.

Let $rounds(\Pi)$ denote the maximum number of communication rounds required for protocol Π to achieve termination in any execution. By the termination guarantees of the agreement property of Byzantine Agreement protocols, $rounds(\Pi)$ is finite and well-defined. Furthermore, given the security assumption that Π remains secure under parallel composition, it follows that Π must terminate within $rounds(\Pi)$ rounds even when executed concurrently across multiple protocol instances.

We now introduce an important abstract concept and will instantiate it in different ways in the proof. A system X is a tuple:

- A set $P = \{P_1, \ldots, P_n\}$ of n parties, each party P_j runs two copies of ITMs, A_j and B_j, for protocol Π;
- An adversary $\mathcal{A}(t, c)$ controlling t corrupted parties and reordering c channels;
- Initial input values for all ITMs on their input tapes;
- A network topology governing inter-party connectivity.

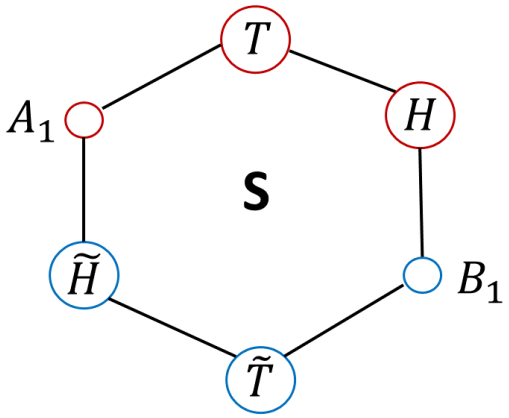


Figure 2 Network topology of System S.

Intuitively, if the system contains a non-trivial adversary who corrupts some parties and reorders some channels, then the network topology is consistent with the adversary's reordering of the channels. In some mental games, we may instantiate the system without an adversary, in which case the network topology dictates how the parties' communication channels are inter-connected crossing the two executions of Π.

For any ITM M run by a party in P, its *view* in system X, denoted by $\mathsf{view}_X(M)$, consists of the content of M's input tapes and random tape, where the former in particular includes the initial input value and the messages received by M from other parties during the execution of X.

When all parties (including $\mathcal{A}(t, c)$) behave deterministically, $\mathsf{view}_X(M)$ is uniquely determined by the initial configurations of the tapes and the adversarial behavior. Conversely, under randomized executions (where the content of the random tapes are sampled uniformly at random), $\mathsf{view}_X(M)$ constitutes a random variable over the corresponding probability space.

We now instantiate the system S through the following configuration:

- **Network Topology**: ITM A_1 establishes connectivity with the set $\widetilde{H}$ (replacing its original link to H), while ITM B_1 interfaces with H instead of $\widetilde{H}$. This structural reconfiguration is illustrated in Figure 2.
- **Initial Input Value**:
  - ITMs in $\{A_1\} \cup T \cup H$ receive input 0;
  - ITMs in $\{B_1\} \cup \widetilde{T} \cup \widetilde{H}$ receive input 1.
- **Adversary**: In this system, there is no adversary; formally, we have $t = 0$ and $c = 0$.
- **Protocol Execution**: All ITMs in S strictly adhere to the instruction of protocol Π, simulating an *authenticated Byzantine Agreement* environment. Specifically, each party:
  - Generates messages as prescribed by Π's honest execution semantics;
  - Sends these messages via the network topology defined above.

Now we first state the following lemmas and we will prove them afterwards.

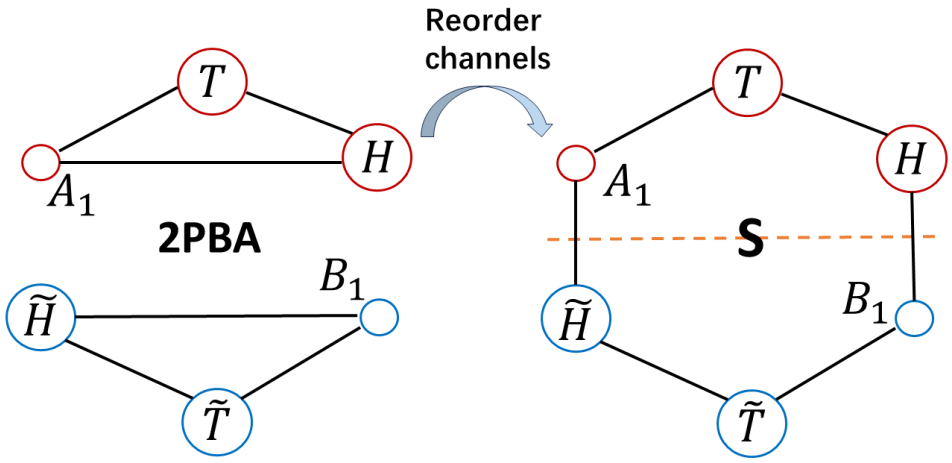


Figure 3 Party's view is identical between System 2PBA and System S.

▶ **Lemma 5.** *In system S, all ITMs in* $\{A_1\} \cup T$ *output 0, and there exists at least one ITM in* H *outputs 0. Symmetrically, all ITMs in* $\{B_1\} \cup \widetilde{T}$ *output 1, and there exists at least one ITM in* $\widetilde{H}$ *outputs 1.*

▶ **Lemma 6.** *In system S, all ITMs in* $\{A_1\} \cup \widetilde{H}$ *and at least one ITM in* $\widetilde{T}$ *output the same value; Symmetrically, all ITMs in* $\{B_1\} \cup H$ *and at least one ITM in* T *outputs the same value.*

By combining Lemma 5 with Lemma 6 we have $Output_S(\widetilde{H}) = Output_S(A_1) = 0$ and at least one ITM $\widetilde{h} \in \widetilde{H}$ satisfies $Output_S(\widetilde{h}) = Output_S(B_1) = 1$. Hence, we derive a contradiction on the output of $\widetilde{h}$, thereby completing the proof of the theorem. ◀

Next, we complete the proofs of Lemma 5 and Lemma 6.

Proof of Lemma 5. To prove the lemma, we define a new system 2PBA as follows and as shown in Figure 3:

- **Network Topology**: The system consists of two parallel executions of protocol Π, denoted Π_0 (consisted of A_1, T and H) and Π_1 (consisted of B_1, $\widetilde{T}$ and $\widetilde{H}$).
- **Initial Input Value**:
  - ITMs in $\{A_1\} \cup T \cup H$ receive input 0;
  - ITMs in $\{B_1\} \cup \widetilde{T} \cup \widetilde{H}$ receive input 1.
- **Adversary**: The adversary $\mathcal{A}(t, c)$ chooses an arbitrary subset of parties $\{P_{\lfloor \frac{n+3}{2} \rfloor}, \cdots, P_n\}$, denoted by C, which satisfies $|C| = \min\{c, n - \lfloor \frac{n+3}{2} \rfloor + 1\}$. Then the adversary reorders all the channels between C and P_1, and corrupts the parties in $\{P_{\lfloor \frac{n+3}{2} \rfloor}, \cdots, P_n\} \setminus C$. As we have $2c + 2t + 1 \geq n$, the adversary can indeed corrupt all these parties. In the following proof, we denote $ITM_0^C = \{ITM_0^j : j \in C\}$ as the set of ITMs run by C in the protocol Π_0, and as the same, we denote $ITM_1^C = \{ITM_1^j : j \in C\}$.
- **Protocol Execution**:
  - All honest parties in S strictly adhere to the instruction of protocol Π;
  - The corrupted parties reorder their channels between P_1, and besides this action, they adhere to the instruction of protocol Π.

We then prove $\mathsf{view}_{2PBA}(A_1) = \mathsf{view}_S(A_1)$, $\mathsf{view}_{2PBA}(T) = \mathsf{view}_S(T)$, and $\mathsf{view}_{2PBA}(H) = \mathsf{view}_S(\widetilde{H})$.

We proceed by induction on the round counter $j \geq 1$.

Base Case ($j = 1$).

- *Message Generation*: All parties in both 2PBA and S generate identical message sets due to protocol Π strictly.
- *Message Routing*:
  - In S: Messages propagate through the hexagonal graph.
  - In 2PBA: The adversary $\mathcal{A}(t, c)$ enforces the following redirections:

$$\begin{aligned} A_1 \to H \overset{\text{reorder}}{\mapsto} \widetilde{H}, \quad & \widetilde{H} \to B_1 \overset{\text{reorder}}{\mapsto} A_1, \\ A_1 \to T, \quad & T \to H \\ H \to T, \quad & T \to A_1 \\ H \to A_1 \overset{\text{reorder}}{\mapsto} B_1, \quad & B_1 \to \widetilde{H} \overset{\text{reorder}}{\mapsto} H. \end{aligned}$$

  After reordering, B_1 communicates with H rather than $\widetilde{H}$, A_1 communicates with $\widetilde{H}$ rather than H. Other than that, all the ITMs strictly adhere to the instruction of Π. It is easy to see that the actual message routing is the hexagonal graph the same as in S.
- *View Equivalence*: The above routing rules induce identical message distributions at all parties' interfaces. Thus, $\mathsf{view}_{2PBA}(A_1) = \mathsf{view}_S(A_1)$, $\mathsf{view}_{2PBA}(T) = \mathsf{view}_S(T)$, and $\mathsf{view}_{2PBA}(H) = \mathsf{view}_S(\widetilde{H})$.

Inductive Step. Assume equivalence holds through step j. For step $j + 1$:

- *Message Generation*: Identical message distributions emerge from equivalent historical views (by inductive hypothesis).
- *Message Routing*: The hexagonal routing in S and adversarial redirection in 2PBA preserve the same correspondence as $j = 1$.
- *View Update*: Therefore, $\mathsf{view}_{2PBA}^{(j+1)}(A_1) = \mathsf{view}_S^{(j+1)}(A_1)$ and $\mathsf{view}_{2PBA}^{(j+1)}(T) = \mathsf{view}_S^{(j+1)}(T)$ maintain for the $j + 1$ rounds.

Owing to the termination property of Byzantine Agreement, which mandates that the protocol Π halts within $rounds(\Pi)$, the aforementioned procedure completes in at most $rounds(\Pi)$ rounds. Consequently, we derive that $\mathsf{view}_{2PBA}(A_1) = \mathsf{view}_S(A_1)$, $\mathsf{view}_{2PBA}(T) = \mathsf{view}_S(T)$, and $\mathsf{view}_{2PBA}(H) = \mathsf{view}_S(\widetilde{H})$.

Given the equivalence of views, the parties A_1, T and H in both systems 2PBA and S must yield identical final outputs. By the validity property of Byzantine Agreement, since A_1, T, and H in Π_0 are initialized with input 0 and A_1, T, $ITM_0^C \subset H$ are honest, then $Output_{2PBA}(A_1) = Output_{2PBA}(T) = Output_{2PBA}(ITM_0^C) = 0$. Consequently, system S inherits this outcome, we have $Output_S(A_1) = Output_S(T) = Output_S(ITM_0^C) = 0$.

Analogously, through the system 2PBA, we can rigorously demonstrate $\mathsf{view}_{2PBA}(B_1) = \mathsf{view}_S(B_1)$, $\mathsf{view}_{2PBA}(\widetilde{T}) = \mathsf{view}_S(\widetilde{T})$ and $\mathsf{view}_{2PBA}(\widetilde{H}) = \mathsf{view}_S(\widetilde{H})$, thereby establishing $Output_S(B_1) = Output_S(\widetilde{T}) = Output_S(ITM_1^C) = 1$. ◀

Proof of Lemma 6. Similarly, we construct system 2PBA' as instantiated below, shown in Figure 4:

- **Network Topology**: The system consists of two parallel executions of protocol Π, denoted Π_0' (consisted of A_1, $\widetilde{H}$ and $\widetilde{T}$) and Π_1' (consisted of B_1, H and T).
- **Initial Input Value**:
  - ITM $\{A_1\}$ receives input 0 and ITMs in $\widetilde{H} \cup \widetilde{T}$ receive input 1;
  - ITM $\{B_1\}$ receives input 0 and ITMs in $H \cup T$ receive input 1.

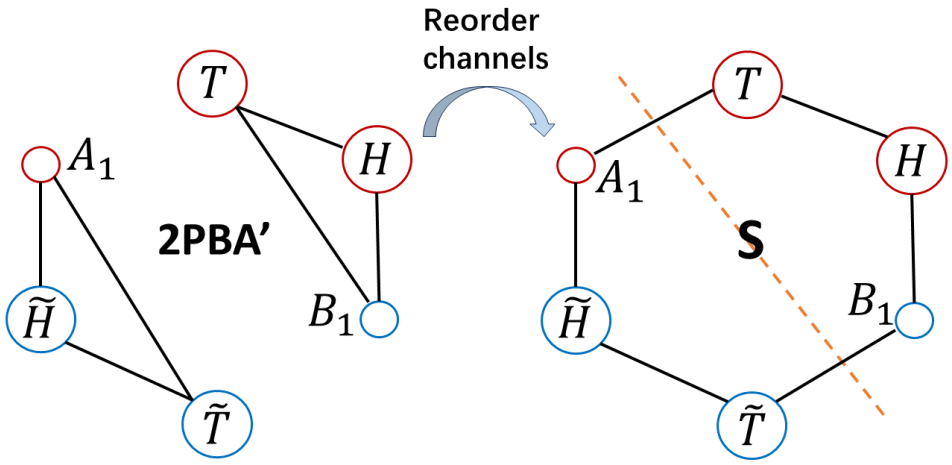


Figure 4 Party's view is identical between System 2PBA' and System S.

- **Adversary**: The adversary $\mathcal{A}(t, c)$ chooses an arbitrarily subset of parties $\{P_2, \cdots, P_{\lfloor \frac{n+1}{2} \rfloor}\}$, denoted by C', which satisfies $|C'| = \min\{c, \lfloor \frac{n+1}{2} \rfloor - 1\}$. Then the adversary reorders all the channels between C' and P_1, and corrupts the parties in $\{P_2, \cdots, P_{\lfloor \frac{n+1}{2} \rfloor}\} \backslash C'$. As we have $2c+2t+1 \geq n$, the adversary can indeed corrupt all these parties. In the following proof, we denote $ITM_0^{C'} = \{ITM_0^j : j \in C'\}$ as the set of ITMs run by C' in the protocol Π_0, and as the same, we denote $ITM_1^{C'} = \{ITM_1^j : j \in C'\}$.
- **Protocol Execution**:
  - All honest parties in S strictly adhere to the instruction of protocol Π;
  - The corrupted parties reorder their channels between P_1, and besides this action, they adhere to the instruction of protocol Π.

The subsequent proof by mathematical induction is analogous to the reasoning in system 2PBA of the preceding proof; therefore it is omitted here. It is noteworthy that after we get $\mathsf{view}_{2PBA'}(A_1) = \mathsf{view}_S(A_1)$, $\mathsf{view}_{2PBA'}(\widetilde{H}) = \mathsf{view}_S(\widetilde{H})$, and $\mathsf{view}_{2PBA'}(\widetilde{T}) = \mathsf{view}_S(\widetilde{T})$, we use the Agreement property instead of the Validity property of Byzantine agreement to deduce that $Output_S(A_1) = Output_S(\widetilde{H}) = Output_S(ITM_1^{C'})$. Similarly, we have $Output_S(B_1) = Output_S(H) = Output_S(ITM_0^{C'})$. ◀

5 Black-box Compiler for Parallel Composable BA

This section presents a novel black-box compiler that transforms any BA protocol into one that remains secure under parallel composition. The proposed approach addresses the challenges introduced by message reordering attacks through the integration of a new Reliable Message Transmission (RMT) protocol.

5.1 Reliable Message Transmission for Parallel Composition

To achieve security under parallel composition, our compiler replaces the standard point-to-point communication primitives in a given BA protocol with our newly designed Reliable Message Transmission protocol. This RMT primitive guarantees correct and unforgeable message delivery within the same protocol instance, even in the presence of adversarial message reordering. The formal specification of the RMT is presented in Protocol 1.[1]

Algorithm 1 Reliable Message Transmission.

Given that party P_i wishes to send message msg to P_j, the reliable message transmission protocol proceeds as follows:

- Round 0: Party P_i initiates this message transmission by sending the message $(deliver, msg, P_i, P_j)$ to all parties except P_j.
- Round 1: Party P_i sends the message $(deliver, msg, P_i, P_j)$ to P_j; and each other party $P_k \neq P_j$ that received message $(deliver, msg, P_i, P_j)$ from P_i in round 0 forwards the same message $(deliver, msg, P_i, P_j)$ to party P_j.
- Decision: Upon party P_j receiving strictly more than $\frac{n-1}{2}$ copies of the message $(deliver, msg, P_i, P_j)$ by the end of round 1, it accepts message msg from P_i.

When $n > 2t+2c+1$, this RMT protocol ensures message delivery through a redundancy-based approach. The principle is to use a majority vote to secure message transmission against potential attacks, a technique also found in prior works but in different formats; see, e.g., [49, 29, 13]. When executed by all participating parties, the RMT primitive is characterized by two essential properties for its security within the black-box compiler.

- *Correctness*: If both parties P_i and P_j are honest, and P_i used the RMT protocol to send a message msg to P_j in round r, then P_j will accept the message msg from P_i by the end of round $r+1$.
- *Unforgeability*: If both parties P_i and P_j are honest, and P_i did not use the RMT protocol to send message msg to P_j in round r, then P_j will not accept msg from P_i by the end of round $r+1$.

The following lemma establishes the security guarantees of the RMT protocol:

▶ **Lemma 7.** *Protocol 1 satisfies both* correctness *and* unforgeability *properties against any adversary $\mathcal{A}(t, c)$, provided that $n > 2c + 2t + 1$.*

Proof. *Correctness:* Since P_i is an honest party, it will send the message $(deliver, msg, P_i, P_j)$ to all other parties except P_j in round 0. In round 1, each party $P_k \neq P_j$, including P_i, forwards/sends the message to P_j, unless:

- P_k is corrupted by the adversary, or
- The communication channel C_{ik} or C_{kj} is attacked by the adversary.

Since each corruption or channel attack can prevent at most one forwarding, at most $t + c$ of these forwarded messages can be suppressed. Thus, P_j receives at least $n - 1 - t - c$ forwarded messages. Since $n > 2c + 2t + 1$, we have $\frac{n-1}{2} > t + c$, which is equivalent to $n - 1 - t - c > \frac{n-1}{2}$. Therefore, P_j receives strictly more than $\frac{n-1}{2}$ copies of the message $(deliver, msg, P_i, P_j)$, and accepts msg from P_i as valid by the end of round 1.

[1] If the values of t and c are publicly known, then the threshold $\frac{n-1}{2}$ in Protocol 1 can be replaced by $t + c$ and the protocol continues to work.

Unforgeability: If P_i did not initiate the RMT protocol for message msg to P_j, then the only way for P_j to receive $(deliver, msg, P_i, P_j)$ is via adversarial intervention. The adversary can inject at most $t + c$ such messages through corrupted parties or attacked channels. Since $t + c < \frac{n-1}{2}$, P_j receives fewer than the majority threshold of messages, and thus does not accept msg from P_i. ◀

5.2 Black-box Compiler

Using the RMT protocol, our black-box compiler applies a simple yet effective transformation: given an existing BA protocol Π_*, it constructs a new protocol Π by replacing all point-to-point message delivery operations with the RMT protocol. The internal logic, state transitions, and computation procedures of Π_* are completely preserved, ensuring that the transformation is non-intrusive and protocol-agnostic.

Formally, a party P_i's ITM^i in protocol Π uses its corresponding ITM^i_* in Π_* as a sub-routine. The outer one, ITM^i, handles message sending and receiving according to the RMT protocol, and passes accepted messages to the inner one, ITM^i_*. The inner one then computes its state transition and out-going messages according to Π_*, and passes the out-going messages to ITM^i to send out.

Notice that in Π, a message that originally takes one round to deliver in Π_* now takes two rounds. As such, honest parties will only initiate a message sending in odd rounds (wlog assuming the protocol starts with round 1) and will only accept a message in even rounds. More specifically, if in the execution of Π_* an honest party P_i sends a message to P_j in round r, then in the execution of Π, P_i initiates the corresponding message sending in round $2r - 1$ and P_j accepts the message by the end of round $2r$.

▶ **Theorem 8.** *Let Π_* be a BA protocol for n parties that tolerates up to t Byzantine corruptions. Let Π be the protocol obtained by replacing all point-to-point communication in Π_* with the RMT protocol. Then, Π is secure under parallel composition of any number of executions against any adversary $\mathcal{A}(t, c)$, provided that $n > 2t + 2c + 1$.*

Proof. Arbitrarily fixing an instance Π_j, we prove its security under parallel composition by induction. Consider a mental game with a single execution of the protocol Π_* as follows.

For each honest party P_i and the corresponding ITM^i_j in Π_j, let its inner Turing machine be ITM^i_{*j}. We construct an ITM^i_* for Π_*, such that ITM^i_* has the same initial input tape, random tape (and setup tape) as ITM^i_j, and thus also as ITM^i_{*j}.

Note that, for any adversary $\mathcal{A}(t, c)$ in Π, the only effects it can have on some party P_i in instance Π_j are the following:

- It can fully control ITM^i_j if the party P_i is corrupted.
- It can cause an honest party P_i's ITM^i_j to accept a message msg from a corrupted party P_j at some round $2r$. This is because, in Π, no honest ITM accepts a message in odd rounds, and due to the unforgeability property of RMT, the adversary cannot forge a message from other honest parties to make an honest P_i accept it.

Therefore, for any adversary $\mathcal{A}$ in Π_j, we simulate a corresponding adversary $\mathcal{A}_*$ in Π_* as follows:

- Whenever $\mathcal{A}$ corrupts a party P_i in Π_j, $\mathcal{A}_*$ corrupts the corresponding ITM^i_* in Π_*.
- Whenever $\mathcal{A}$ causes an honest ITM^i_j to accept a message msg from a corrupted party P_j at round $2r$ in Π_j, $\mathcal{A}_*$ sends message msg from ITM^j_* to ITM^i_* at round r in Π_*.

Since for every honest party P_i, both ITM^i_{*j} and ITM^i_* have the same input, random, and setup tapes at the beginning of round 1, they have the same state transition and out-going messages. Because ITM^i_j handles message sending and receiving according to the RMT

protocol, and by the construction of the adversary $\mathcal{A}_*$ in Π_*, at the end of round 2 in Π_j, ITM_j^i has accepted exactly the same messages as received by ITM_*^i at the end of round 1 in Π_*. Thus ITM_{*j}^i after round 2 of Π_j has the same view as ITM_*^i after round 1 of Π_*. That is, $\text{view}_{\Pi_j}^2(ITM_{*j}^i) = \text{view}_{\Pi_*}^1(ITM_*^i)$.

Let $\text{round}(\Pi_*)$ be the maximum finite number of rounds in protocol Π_*. For each $r = 1, \ldots, \text{round}(\Pi_*)$, we compare the view of ITM_{*j}^i after $2r$ rounds of Π_j with the view of ITM_*^i after r rounds of Π_*. By the inductive hypothesis, we have

$$\text{view}_{\Pi_j}^{2r-2}(ITM_{*j}^i) = \text{view}_{\Pi_*}^{r-1}(ITM_*^i).$$

Thus they again have the same state transition and out-going messages in the corresponding round $2r-1$ and round r.

Due to the correctness and the unforgeability properties of the RMT protocol, and by the construction of the adversary $\mathcal{A}_*$, again by the end of round $2r$ of Π_j, ITM_j^i accepts exactly the same messages as received by ITM_*^i at the end of round r in Π_*. Hence, $\text{view}_{\Pi_j}^{2r}(ITM_{*j}^i) = \text{view}_{\Pi_*}^{r}(ITM_*^i)$ for all r.

Accordingly, we conclude that the final output value of ITM_j^i in Π_j (which is that of ITM_{*j}^i) and the final output value of ITM_*^i in Π_* are identical. As this holds for all honest parties in Π_j, we have that Π_j inherits the same agreement and validity properties as Π_*. Thus Π is secure under parallel composition. ◀

By applying our compiler to any existing BA protocol, we can construct a protocol that remains secure under parallel composition. In particular, by transforming the protocol of Garay and Moses [31], which is secure in the unauthenticated, stateless setting whenever $n > 3t$, we obtain the following corollary:

▶ **Corollary 9.** *There exists a protocol for stateless, unauthenticated BA that is secure under parallel composition of any number of executions against any adversary $\mathcal{A}(t, c)$, provided that the total number of parties n satisfies $n > 3t$ and $n > 2c + 2t + 1$.*

While the black-box compiler significantly enhances BA protocol's security, it introduces some performance overheads. The round complexity of the compiled protocol Π is doubled compared with the original protocol Π_*. The communication overhead per message in the original Π_* is asymptotically $O(n)$. In particular, for a message from P_i to P_j, P_i sends to $n-2$ parties in Round 0 of RMT, and up to $n-1$ parties may send the message to P_j in round 1. These overheads are the trade-off for achieving security against adversarial message reordering in parallel compositions. Designing more efficient compilers that maintain similar security guarantees while reducing communication or round complexity remains an important direction for future research.

6 Concurrent BA Protocol under Asynchronous Network

The preceding sections focused on parallel composition under a synchronous network model, where message delivery is guaranteed within a fixed time bound (i.e., a round). In this section, we shift our attention to a more challenging setting, the asynchronous network model. Here, messages can be delayed arbitrarily, and there is no global clock to coordinate the actions of different parties. Moreover, we consider concurrent composition, where protocol instances may not start simultaneously, making it significantly harder for a party to determine whether a received message indeed belongs to a particular instance or has been reordered.

To address these challenges, we first construct a reliable broadcast protocol tailored for the asynchronous setting. Building upon this foundation, we then leverage the seminal work of [6] and construct an unauthenticated BA protocol that is secure under concurrent composition in an asynchronous network. Notably, security in the asynchronous model implies security in the synchronous setting. Therefore, our BA protocol is also secure under concurrent composition in synchronous networks.

6.1 Reliable Broadcast Under Concurrent Composition

Reliable Broadcast (RB) [10] is a fundamental communication primitive that ensures a message sent by an honest sender is reliably delivered to all parties. It guarantees the following properties:

- *Agreement*: If any honest party accepts a message m, then all honest parties eventually accept m.
- *Validity*: If the sender is honest and broadcasts a message m, then all honest parties eventually accept m.

Based on the work of [6] and utilizing the idea behind our RMT protocol in Section 5, we present Protocol 2 that achieves reliable broadcast under concurrent executions against any adversary $\mathcal{A}(t, c)$, under the conditions $n > 3t$ and $n > 2c + 2t + 1$. To our best knowledge, this is the first RB protocol that is concurrently secure in the stateless model, and may be of independent interest.

The core idea is to avoid accepting messages based solely on direct receipt. Instead, each party maintains a local view of all other parties' state. This view is only updated after a majority (i.e., more than $\frac{n-1}{2}$) of confirming messages have been received. Specifically, let the original sender in the RB protocol be $P_g = P_1$. The local state of a party P_j with $j = 1, \ldots, n$ at some party P_i, denoted by S_j^i, can be in one of the following three forms:

- $\{initial, \perp\}$: no value has been received by P_j yet;
- $\{prepare, v\}$: value v tentatively accepted by P_j;
- $\{commit, v\}$: value v confirmed and will not change at P_j.

Respectively, the messages sent by a party P_i during the protocol reflect its own state and take one of the following forms:

- $(receive, v, P_g)$: P_i has received value v sent by P_g;
- $(echo, v, P_i)$: P_i has tentatively accepted v;
- $(ready, v, P_i)$: value v has been confirmed by P_i.

By adapting the BA protocol Π^* from Figure 4 of [6] and replacing all broadcast procedures with our RB protocol, we have the following:

▶ **Theorem 10.** *There exists a protocol for stateless, unauthenticated BA that is secure under concurrent composition of any number of executions in an asynchronous network against any adversary $\mathcal{A}(t, c)$, where the number of parties n satisfies $n > 3t$ and $n > 2c + 2t + 1$.*

To prove Theorem 10, it suffices to prove the security of our RB protocol under concurrent composition in an asynchronous network; see Theorem 17 in Section 6.2. Theorem 10 then holds by the security of protocol Π^* and an inductive analysis similar to the proof of Theorem 8. We include the protocol Π^* in the appendix of this paper for reference purpose.

Algorithm 2 Reliable Broadcast.

Say party P_g wishes to broadcast value v. Each party P_i initializes its local view of each party P_j's state as $S_j^i = \{initial, \perp\}$. The protocol proceeds as follows:

- Step 0 (performed by P_g): send $(initial, v)$ to all parties including itself.

For each party P_i:

- Step 1: Upon receiving some message $(initial, v)$ from P_g, send $(receive, v, P_g)$ to all parties.
- Step 2: Wait until the receipt of strictly more than $\frac{n-1}{2}$ $(receive, v, P_g)$ messages for the same value v from different parties. If local state $S_i^i = \{initial, \perp\}$, set $S_i^i = \{prepare, v\}$ and send $(echo, v, P_i)$ to all parties.
- Step 3: Upon receiving some message $(echo, v, P_j)$ from party P_j, forward it to all parties.
- Step 4: Wait until the receipt of strictly more than $\frac{n-1}{2}$ $(echo, v, P_j)$ messages for the same v and P_j from different parties. If $S_j^i = \{initial, \perp\}$, set $S_j^i = \{prepare, v\}$.
- Step 5: Wait until strictly more than $\frac{n+t}{2}$ parties P_j have local state $S_j^i = \{prepare, v\}$ for the same v. If $S_i^i = \{initial, \perp\}$ or $S_i^i = \{prepare, *\}$, where $*$ can be any value, set $S_i^i = \{commit, v\}$ and send $(ready, v, P_i)$ to all parties.
- Step 6: Upon receiving some message $(ready, v, P_j)$ from party P_j, forward it to all parties.
- Step 7: Wait until the receipt of strictly more than $\frac{n-1}{2}$ $(ready, v, P_j)$ messages for the same v and P_j from different parties. If $S_j^i = \{initial, \perp\}$ or $S_j^i = \{prepare, *\}$, where $*$ can be any value, set $S_j^i = \{commit, v\}$.
- Step 8: Wait until $t + 1$ parties P_j have state $S_j^i = \{commit, v\}$ for the same v. If $S_i^i = \{initial, \perp\}$ or $S_i^i = \{prepare, *\}$, where $*$ can be any value, set $S_i^i = \{commit, v\}$ and send $(ready, v, P_i)$ to all parties.
- Step 9: Wait until $2t+1$ parties P_j have state $S_j^i = \{commit, v\}$ for the same v. Accept v.

6.2 Composition Security of Reliable Broadcast

We now prove that Protocol 2 satisfies the properties of reliable broadcast under concurrent composition. In particular, we consider m instances of Protocol 2 concurrently executed against adversary $\mathcal{A}(t, c)$, under conditions $n > 3t$ and $n > 2c + 2t + 1$.

▶ **Lemma 11.** *In any instance, for any two honest parties P_i and P_j, P_i eventually sets its local state $S_j^i = \{commit, v\}$ if and only if P_j broadcasts the message $(ready, v, P_j)$ during the execution of this instance.*

Proof. If an honest party P_j broadcasts $(ready, v, P_j)$, then in the execution of any other honest party P_i, this message will be forwarded by every party P_k unless:

- P_k is corrupted by the adversary, or
- the communication channel C_{jk} or C_{ki} is under attack.

Since each adversary action can block at most one message, the number of suppressed forwards is at most $t + c$. Therefore, P_i eventually receives more than $\frac{n-1}{2}$ such messages and sets its local state for P_j to $S_j^i = \{commit, v\}$.

Conversely, if P_j did not send the message in that instance, then the adversary can inject at most $t + c$ copies $(ready, v, P_j)$. Since $t + c < \frac{n-1}{2}$, an honest party will not receive enough messages to satisfy the threshold, and thus will not set P_j's local state to $S_j^i = \{commit, v\}$. ◀

▶ **Lemma 12.** *In any instance, for any two honest parties P_i and P_j, P_i eventually sets its local state $S_j^i = \{prepare, v\}$ only if P_j broadcasts the message $(echo, v, P_j)$ during the execution of this instance.*

Proof. If P_j did not send the message $(echo, v, P_j)$ in that instance, then the adversary can inject at most $t + c$ copies $(echo, v, P_j)$. Since $t + c < \frac{n-1}{2}$, an honest party will not receive enough messages to satisfy the threshold, and thus will not set P_j's local state to $S_j^i = \{prepare, v\}$. ◀

▶ **Lemma 13.** *In any instance, for any two honest parties P_i and P_j who respectively send messages $(ready, v, P_i)$ and $(ready, u, P_j)$ during the execution of this instance, we have $v = u$.*

Proof. Assume for contradiction. Let P_x and P_y be the first honest parties to send $(ready, v, P_x)$ and $(ready, u, P_y)$, respectively. Party P_x must have observed strictly more than $\frac{n+t}{2}$ party P_j with local state $S_j^x = \{prepare, v\}$. Party P_y must have observed strictly more than $\frac{n+t}{2}$ party P_j with local state $S_j^y = \{prepare, u\}$. This implies an overlap where an honest party P_k has $\{prepare, v\}$ in the view of P_x and $\{prepare, u\}$ in the view of P_y.

Therefore, by Lemma 12, it implies that P_k sent both $(echo, v)$ and $(echo, u)$ messages. However, an honest party sends only one type of *echo* message during a reliable broadcast protocol. Therefore, $v = u$. ◀

▶ **Lemma 14.** *In any instance, for any two honest parties P_i and P_j who accept values v and u respectively, we have $v = u$.*

Proof. If P_i accepts v, then it must have observed at least $2t + 1$ parties P_k with the state $S_k^i = \{commit, v\}$ in its local view, which includes at least $t + 1$ honest parties. Similarly, P_j must have observed at least $2t + 1$ parties P_k with state $S_k^j = \{commit, u\}$ in its local view, which includes at least $t + 1$ honest parties. By Lemma 11, there exists at least one honest party P_x that have send $(ready, v, P_x)$, and one honest party P_y that have send $(ready, u, P_y)$. Applying Lemma 13, it follows that $v = u$. ◀

▶ **Lemma 15.** *In any instance, if an honest party P_i accepts a value v, then every other honest party eventually accepts v in that instance.*

Proof. Suppose P_i accepts value v. Then its local view must include at least $2t + 1$ parties P_x with the state $S_x^i = \{commit, v\}$, including at least $t + 1$ honest parties. By Lemma 11, every honest party P_k in these $t + 1$ parties must have send $(ready, v, P_k)$ in that instance. Also by Lemma 11, eventually, all honest parties P_j will receive these messages and update their local views, setting P_k's state to $S_k^j = \{commit, v\}$. According to step 8 of the protocol, upon observing $t + 1$ parties in state $\{commit, v\}$, every honest party P_j updates its own state to $S_j^j = \{commit, v\}$, and send $(ready, v, P_j)$ to all parties. Again, by Lemma 11, all honest parties P_y will receive these messages and update their local views, setting P_j's state $S_j^y = \{commit, v\}$ eventually. Since $n > 3t$, the number of honest parties exceeds $2t + 1$. Thus, every honest party will eventually have at least $2t + 1$ $\{commit, v\}$ states in its local view and will accept v. ◀

▶ **Lemma 16.** *In any instance, if an honest party P_g broadcasts a value v, then all honest parties eventually accept v in that instance.*

Proof. Since P_g is honest, it sends $(initial, v, P_g)$ to all parties. At least $(n-t-c)$ honest parties receive this message and forward $(receive, v, P_g)$ to all other parties. Each honest party P_i eventually receives more than $\frac{n-1}{2}$ such $(receive, v, P_g)$ messages. It then sets its local state $S_i^i = \{prepare, v\}$ and sends $(echo, v, P_i)$ to all parties. As a result, each honest party P_i receives more than $\frac{n-1}{2}$ *echo* messages for value v and each honest party P_j, and thus sets the state $S_j^i = \{prepare, v\}$. This ensures that every honest party P_i observes enough *prepare* state in its local view to satisfy the condition for committing. Accordingly, each honest party updates its own state $S_i^i = \{commit, v\}$ and sends $(ready, v, P_i)$ to all parties.

In Step 7 of the protocol, each honest party P_i receives more than $\frac{n-1}{2}$ *ready* messages for value v and each honest party P_j, causing each party to update the state of P_j $S_j^i = \{commit, v\}$. Therefore, since $n > 3t$, every honest party P_i eventually has at least $2t+1$ $\{commit, v\}$ states in local view and accepts v. ◀

▶ **Theorem 17.** *Protocol 2 realizes reliable broadcast against any adversary $\mathcal{A}(t, c)$ under concurrent composition in an asynchronous network, provided that $n > 3t$ and $n > 2t+2c+1$.*

Proof. *Validity:* In any instance, if the sender P_g is honest and broadcast value v, then by Lemma 16, all honest parties eventually accept v in that instance.

Agreement: In any instance, if the sender P_g is corrupted, and some honest party P_i accepts a value v in that instance, then by Lemma 15, all other honest parties eventually accept the same value v in that instance. ◀

7 Conclusion and Future Directions

The primary contributions of this work include the introduction of a novel adversarial model that simultaneously enables the corruption of parties and communication channels. The reorder attack is both theoretically interesting and practically viable, enhancing the security analysis for Byzantine Agreement protocols under both parallel and concurrent execution settings.

Building upon this model, our work establishes surprising positive and negative results. On the one hand, authenticated Byzantine Agreement becomes insecure when either $n \leq 3t$ or $n \leq 2c + 2t + 1$. While on the other hand, for unauthenticated Byzantine Agreement, secure protocols exist under both parallel and concurrent compositions when $n > 3t$ and $n > 2c+2t+1$. Notably, these findings provide tight conditions for the security of Byzantine Agreement under compositional executions – a fundamental advance in understanding protocol resilience against corruption and reorder attacks.

The framework and results established in this paper motivate several promising directions for future research:

1. In terms of positive results, we provide a black-box reduction for parallel-executed BA protocols. Specifically, any unauthenticated Byzantine Agreement that is secure under single-instance execution when $t < n/3$ can be efficiently transformed into a parallel-secure protocol when in addition one has $2t + 2c + 1 < n$. For concurrently-executed Byzantine Agreement, we demonstrate our positive result via a concrete protocol using reliable broadcast. We believe that our reliable broadcast protocol can also be used to help other BA protocols achieve concurrent security, but a key open question remains: does there exist a black-box reduction approach that generically transforms a single-instance secure BA protocol into one that is concurrently secure? Also, the composition of BA protocols against dynamic adversaries would be very interesting to explore, since so far all studies on compositions consider static adversaries.

2. The communication complexity of BA protocols has been a central focus in the literature. Under our proposed adversarial model and compositional execution framework, this work focused on polynomial-time computation and communication protocols and didn't investigate whether the complexity bounds of compositionally-secure BA protocols diverge from those of classical single-execution authenticated Byzantine Agreement. Specifically, we are curious about how reorder attacks may impact existing communication lower- and upper-bounds. Intuitively, such attacks necessitate additional communication to preserve security, but the exact complexity trade-offs require a rigorous study.
3. In this work we focus on the standard notion of validity commonly considered in the literature. Since other validity notions such as *weak validity* have also been studied [42], we are interested in whether our conclusions still hold with respect to those notions. Intuitively, weaker validity conditions would relax the safety constraints that BA protocols must satisfy, hence may allow the impossibility results be circumvented. Thus extending our current model to weaker validity notions constitutes a promising direction for future research.

References

1 Michael Abd-El-Malek, Gregory R. Ganger, Garth R. Goodson, Michael K. Reiter, and Jay J. Wylie. Fault-scalable Byzantine fault-tolerant services. *SIGOPS Oper. Syst. Rev.*, 39(5):59–74, October 2005. doi:10.1145/1095809.1095817.

2 Shreya Agrawal and Khuzaima Daudjee. A performance comparison of algorithms for Byzantine agreement in distributed systems. In *2016 12th European Dependable Computing Conference (EDCC)*, pages 249–260. IEEE, 2016. doi:10.1109/EDCC.2016.17.

3 Michael Ben-Or and Ran El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distrib. Comput.*, 16(4):249–262, 2003. doi:10.1007/s00446-002-0083-3.

4 Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10. ACM, 1988. doi:10.1145/62212.62213.

5 Elette Boyle, Ran Cohen, and Aarushi Goel. Breaking the $O(\sqrt{n})$-bit barrier: Byzantine agreement with polylog bits per party. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC '21, pages 319–330. ACM, 2021. doi:10.1145/3465084.3467897.

6 Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987. doi:10.1016/0890-5401(87)90054-X.

7 Marcus Brinkmann, Christian Dresen, Robert Merget, Damian Poddebniak, Jens Müller, Juraj Somorovsky, Jörg Schwenk, and Sebastian Schinzel. ALPACA: Application layer protocol confusion - analyzing and mitigating cracks in TLS authentication. In *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/brinkmann.

8 R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, page 136. IEEE, 2001.

9 Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186. USENIX Association, 1999. URL: https://dl.acm.org/citation.cfm?id=296824.

10 Jo-Mei Chang and Nicholas F. Maxemchuk. Reliable broadcast protocols. *ACM Trans. Comput. Syst.*, 2(3):251–273, 1984. doi:10.1145/989.357400.

11 David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 11–19. ACM, 1988. doi:10.1145/62212.62214.

12 Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. doi:10.1016/J.TCS.2019.02.001.

13 Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, and Jovan Komatovic. As easy as ABC: optimal (a)ccountable (b)yzantine (c)onsensus is easy! *J. Parallel Distributed Comput.*, 181:104743, 2023. doi:10.1016/J.JPDC.2023.104743.

14 Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Anton Paramonov, and Manuel Vidigueira. All Byzantine agreement problems are expensivge. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, PODC '24, pages 157–169. ACM, 2024. doi:10.1145/3662158.3662780.

15 Jeremy Clark and Paul C. van Oorschot. SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *2013 IEEE Symposium on Security and Privacy*, pages 511–525, 2013. doi:10.1109/SP.2013.41.

16 Ran Cohen, Pouyan Forghani, Juan Garay, Rutvik Patel, and Vassilis Zikas. Concurrent asynchronous Byzantine agreement in expected-constant rounds, revisited. In *Theory of Cryptography Conference*, pages 422–451. Springer, 2023. doi:10.1007/978-3-031-48624-1_16.

17 Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient Byzantine agreement and multi-party computation with asynchronous fallback. In *Theory of Cryptography Conference*, pages 623–653. Springer, 2021. doi:10.1007/978-3-030-90459-3_21.

18 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985. doi:10.1145/2455.214112.

19 Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. doi:10.1137/0212045.

20 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

21 Cynthia Dwork, David Peleg, Nicholas Pippenger, and Eli Upfal. Fault tolerance in networks of bounded degree. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 370–379. ACM, 1986. doi:10.1145/12130.12169.

22 Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. Impact of man-in-the-middle attacks on Ethereum. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 11–20, 2018. doi:10.1109/SRDS.2018.00012.

23 Paul Feldman and Silvio Micali. Optimal algorithms for Byzantine agreement. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 148–161. ACM, 1988. doi:10.1145/62212.62225.

24 Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997. doi:10.1137/S0097539790187084.

25 Rex Fernando, Yuval Gelles, and Ilan Komargodski. Scalable distributed agreement from LWE: Byzantine agreement, broadcast, and leader election. In *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, pages 1–23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ITCS.2024.46.

26 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

27 Matthias Fitzi. *Generalized communication and security models in Byzantine agreement.* PhD thesis, ETH Zurich, 2003.

28 Matthias Fitzi, Nicolas Gisin, Ueli M. Maurer, and Oliver von Rotz. Unconditional Byzantine agreement and multi-party computation secure against dishonest minorities from scratch. In *Advances in Cryptology – EUROCRYPT '02*, pages 482–501. Springer, 2002. doi:10.1007/3-540-46035-7_32.

29 Matthias Fitzi and Ueli M. Maurer. From partial consistency to global broadcast. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 494–503. ACM, 2000. doi:10.1145/335305.335363.

30 Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure faut-tolerant protocols and the public-key model. In *Advances in Cryptology – CRYPTO '87*, pages 135–155. Springer, 1987. doi:10.1007/3-540-48184-2_10.

31 Juan A. Garay and Yoram Moses. Fully polynomial Byzantine agreement for $n > 3t$ processors in $t + 1$ rounds. *SIAM J. Comput.*, 27(1):247–290, 1998. doi:10.1137/S0097539794265232.

32 Yuval Gelles and Ilan Komargodski. Optimal load-balanced scalable distributed agreement. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC '24, pages 411–422. ACM, 2024. doi:10.1145/3618260.3649736.

33 Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In *Advances in Cryptology – CRYPTO '02*, pages 178–193. Springer, 2002. doi:10.1007/3-540-45708-9_12.

34 Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 38–49. ACM, 2012. doi:10.1145/2382196.2382204.

35 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play ANY mental game. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229. ACM, 1987. doi:10.1145/28395.28420.

36 Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on computing*, 17(2):281–308, 1988. doi:10.1137/0217017.

37 Anuj Gupta, Prasant Gopal, Piyush Bansal, and Kannan Srinathan. A new look at composition of authenticated Byzantine generals. *arxiv*, 2012. arXiv:1203.1463.

38 Lioba Heimbach and Roger Wattenhofer. SoK: Preventing transaction reordering manipulations in decentralized finance. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, AFT '22, pages 47–60. ACM, 2023. doi:10.1145/3558535.3559784.

39 Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for Byzantine agreement. *J. Comput. Syst. Sci.*, 75(2):91–112, 2009. doi:10.1016/J.JCSS.2008.08.001.

40 Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 990–999. SIAM, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109667.

41 Po-Chun Kuo, Hao Chung, Tzu-Wei Chao, and Chen-Mou Cheng. Fair Byzantine agreements for blockchains. *IEEE Access*, 8:70746–70761, 2020. doi:10.1109/ACCESS.2020.2986824.

42 Leslie Lamport. The weak byzantine generals problem. *Journal of the ACM (JACM)*, 30(3):668–676, 1983. doi:10.1145/2402.322398.

43 Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982. doi:10.1145/357172.357176.

44 Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated Byzantine agreement. *J. ACM*, 53(6):881–917, 2006. doi:10.1145/1217856.1217857.

45 Thomas Locher. Fast Byzantine agreement for permissioned distributed ledgers. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '20, pages 371–382. ACM, 2020. doi:10.1145/3350755.3400219.

46 M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980. doi:10.1145/322186.322188.

47 Birgit Pfitzmann and Michael Waidner. Unconditional Byzantine agreement for any number of faulty processors. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '92, pages 339–350. Springer, 1992. doi:10.1007/3-540-55210-3_195.

48 Zhiguo Qu, Zhexi Zhang, Bo Liu, Prayag Tiwari, Xin Ning, and Khan Muhammad. Quantum detectable Byzantine agreement for distributed data trust management in blockchain. *Information Sciences*, 637:118909, 2023. doi:10.1016/J.INS.2023.03.134.

49 Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 73–85. ACM, 1989. doi:10.1145/73007.73014.

50 R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. doi:10.1145/359340.359342.

51 David Sounthiraraj, Justin Sahs, Garrett Greenwood, Zhiqiang Lin, and Latifur Khan. Smvhunter: Large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in android apps. In *Network and Distributed System Security Symposium (NDSS)*, pages 1–14. The Internet Society, 2014. doi:10.14722/ndss.2014.23205.

52 Ran Tamir, Ariel Livshits, and Yonatan Shadmi. Simple majority consensus in networks with unreliable communication. *Entropy*, 24(3), 2022. doi:10.3390/e24030333.

53 Jianhuan Wang, Jichen Li, Zecheng Li, Xiaotie Deng, and Bin Xiao. n-mvtl attack: Optimal transaction reordering attack on DeFi. In *Computer Security – ESORICS 2023*, pages 367–386. Springer, 2023. doi:10.1007/978-3-031-51479-1_19.

54 Andrew Chi-Chih Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164. IEEE, 1982. doi:10.1109/SFCS.1982.38.

55 Junghun Yoo, Youlim Jung, Donghwan Shin, Minhyo Bae, and Eunkyoung Jee. Formal modeling and verification of a federated Byzantine agreement algorithm for blockchain platforms. In *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 11–21. IEEE, 2019.

56 Zhenwei Zhao, Xiaoming Li, Bing Luan, Weining Jiang, Weidong Gao, and Subramani Neelakandan. Secure internet of things (IoT) using a novel Brooks Iyengar quantum Byzantine agreement-centered blockchain networking (BIQBA-BCN) model in smart healthcare. *Information Sciences*, 629:440–455, 2023. doi:10.1016/J.INS.2023.01.020.

A Consensus Protocol in Figure 4 of [6]

Algorithm 3 The Consensus Protocol in Figure 4 of [6].

Phase(i): (by process p)

Step 1: $Broadcast(p, 3i+1, value_p)$. Wait until validate $n-t$ $(3i+1)$-messages.

- $value_p :=$ majority value of the $n-t$ validated messages.

Step 2: $Broadcast(p, 3i+2, value_p)$. Wait until validate $n-t$ $(3i+2)$-messages.

- **(i)** If more than $\frac{n}{2}$ of the messages have the same value v, then $value_p = (d, v)$.
- **(ii)** Otherwise, $value_p := value_p$.

Step 3: $Broadcast(p, 3i+3, value_p)$. Wait until validate $n-t$ $(3i+3)$-messages.

- **(i)** If validated more than $2t$ messages with value (d, v) then $decision_p := value_p := v$.
- **(ii)** If validated more than t messages with value (d, v) then $value_p := v$.
- **(iii)** Otherwise, $value_p :=$ *coin_toss* (0 or 1 with probability $\frac{1}{2}$).

Go to round 1 of phase $i+1$

Optimistic Message Dissemination

Chen-Da Liu-Zhang
Lucerne University of Applied Sciences and Arts, Rotkreuz, Switzerland
Web3 Foundation, Zug, Switzerland

Christian Matt
Primev, Steinhausen, Switzerland

Søren Eller Thomsen
Partisia, Aarhus, Denmark

Abstract

Message dissemination is a fundamental building block in distributed systems and guarantees that any message sent eventually reaches all parties. State of the art provably secure protocols for disseminating messages have a per-party communication complexity that is linear in the inverse of the fraction of parties that are guaranteed to be honest in the worst case. Unfortunately, this per-party communication complexity arises even in cases where the actual fraction of parties that behave honestly is close to 1. In this paper, we propose an optimistic message dissemination protocol that adopts to the actual conditions in which it is deployed, with optimal worst-case per-party communication complexity. Our protocol cuts the complexity of prior provably secure protocols for 49% worst-case corruption almost in half under optimistic conditions and allows practitioners to combine efficient heuristics with secure fallback mechanisms.

2012 ACM Subject Classification Networks → Peer-to-peer protocols

Keywords and phrases flooding, message dissemination, optimistic

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.14

Related Version *Full Version*: `https://ia.cr/2025/1404` [26]

Funding This work was supported by the Ethereum Foundation, grant number FY24-1527.
Søren Eller Thomsen: The work was partly done at The Alexandra Institute, Aarhus, Denmark.

1 Introduction

1.1 Motivation

A basic task in distributed systems is to disseminate a message to all parties in the system. This is often done using a flooding protocol where a party sends the message to all its neighbors, and the neighbors in turn send the message to all their neighbors, and so on. Due to the inherent redundancy in this process, flooding protocols are robust to message loss and random failures of parties. Practical implementations of flooding protocols often optimize the dissemination latency using several heuristics, e.g., preferring to send messages to parties that are geographically closer and have lower latency [22, 35]. Such heuristic protocols, however, cannot be secure in the presence of Byzantine corruptions: For example, it is possible that all parties geographically close to some honest party are corrupted and eclipse the honest party from the network, even though the corruption fraction in the total network is small.

On the other hand, there are provably secure flooding protocols that guarantee the delivery of messages to all honest parties after a bounded number of steps as long as the overall corruption fraction is below a predetermined threshold [24, 25, 28]. The inherent downside of these protocols is that they have a higher per-party communication complexity and do not allow for heuristic optimizations. Furthermore, the per-party communication complexity of these protocols is linear in the inverse of the fraction of parties guaranteed to

© Chen-Da Liu-Zhang, Christian Matt, and Søren Eller Thomsen;
licensed under Creative Commons License CC-BY 4.0

be honest (which was shown to be optimal in [25]), where the threshold of honest parties needs to be set a priori. This communication complexity arises even when the actual fraction of parties that behave honestly is close to 1. When deploying this type of protocols, the engineers setting the parameters are thus left choosing between a protocol that *always* is secure but *most of the time* has an excessive bandwidth usage, or choosing a protocol that is secure *most of time* and *always* has a low bandwidth.

1.2 Contributions

Consider two thresholds $\gamma_{\text{BC}} \geq \gamma_{\text{WC}}$. We ask whether there is a protocol which is secure as long as at least a γ_{WC} fraction of parties behave honestly, and achieves a strictly better per-party communication complexity when the actual number of honest parties is at least γ_{BC}:

> *Is there flooding protocol that has optimal worst-case per-party communication complexity with at least γ_{WC} honesty-fraction, and a strictly better complexity in the optimistic case with γ_{BC} honesty-fraction?*

We answer this question affirmatively by proposing a flooding protocol with two distinct features: First, our protocol has an optimistic path that is efficient when the actual fraction of honest parties is high. If the actual corruption fraction is high, our protocol employs a fall-back mechanism that guarantees the delivery of messages to all honest parties in this worst-case scenario. Secondly, the optimistic path of our protocol can be instantiated with an arbitrary flooding protocol, including those that are optimized for practical deployment. This allows practitioners to use their favorite flooding protocol together with a secure fallback mechanism. That way, we achieve the best of both worlds: Protocols that take into account practical peculiarities such as physical distance, and provably secure protocols that ensure reliable message delivery even against a Byzantine worst-case adversary.

Applicability of results. Our constructions are built in a black-box manner from existing flooding protocols, and impose no additional assumptions on the fraction of honest parties beyond what these underlying protocols require. Additionally, we assume that all parties have access to a PKI infrastructure and access to a randomness beacon, both of which are readily available in most blockchain systems. As our protocols rely on splitting messages into multiple shares and on cryptographic techniques to guarantee the validity of these shares, our constructions are not suitable for disseminating very short messages. Hence, the protocols are better suited for disseminating blocks than individual transactions.

We assume a synchronous network with point-to-point channels between all parties. The protocol as we describe it requires this synchrony (i.e., knowledge of an upper bound on the delivery time of the point-to-point channels) to guarantee that all honest parties receive all messages in the worst case. The protocol can be slightly modified to guarantee delivery to all honest parties in all cases without synchrony, and only require synchrony to achieve better efficiency in the optimistic case at the expense of slightly more communication in the optimistic case. The protocol can thus be adapted to best suit the synchrony assumption and delivery requirements of the application it is used in.

Warm-up: Asymptotically optimal flooding. As a first step, we present a simple protocol PushPullFlood that achieves asymptotically optimal per-party communication (but is not optimistic). Instead of only "pushing" messages to all parties, it additionally allows parties to "pull" messages from other parties that have already received the message. We will use this mechanism also in our optimistic protocol.

▶ **Theorem 1** (PushPullFlood (informal)). *Let Π_{Flood} be a flooding protocol and let $\Pi_{FracFlood}$ be a flooding protocol that delivers to only a constant fraction of the honest parties. Then, PushPullFlood$(\Pi_{Flood}, \Pi_{FracFlood})$ makes black-box use of both protocols and is a secure flooding protocol with a per-party communication complexity for sending an l-bit message with security parameter κ proportional to that of $\Pi_{FracFlood}$ distributing l-bit messages plus Π_{Flood} distributing κ-bit messages.*

By instantiating Π_{Flood} and $\Pi_{\text{FracFlood}}$ with appropriate protocols from [24] and [25] respectively, it follows that PushPullFlood can provide flooding with asymptotically optimal per-party communication.

Asymptotically optimal optimistic flooding. We then strengthen the result above and present the first flooding protocol OptimisticFlood that is asymptotically-optimal in the worst-case, but also has improved efficiency in the optimistic case.

Below we use γ_{ACTUAL} to quantify the actual fraction of parties behaving honestly and state an informal version of the theorem we show for OptimisticFlood.

▶ **Theorem 2** (OptimisticFlood (informal)). *Let $\gamma_{BC} \geq \gamma_{WC} \geq 0$. Further let Π_{BC} be a flooding protocol secure for the optimistic case $\gamma_{\text{ACTUAL}} \geq \gamma_{BC}$ and let Π_{WC} be a flooding protocol secure for the worst-case $\gamma_{\text{ACTUAL}} \geq \gamma_{WC}$. Then OptimisticFlood$(\Pi_{BC}, \Pi_{WC})$ is a secure flooding protocol assuming a γ_{WC} fraction of honesty, with per-party communication complexity and delivery guarantees roughly equal to that of Π_{BC} in the optimistic case, and the sum of both Π_{BC} and Π_{WC} with small overhead in the worst-case.*

A natural choice is to let both Π_{BC} and Π_{WC} be instantiated with asymptotically optimal flooding protocols with a per-party communication complexity of $O(l \cdot \gamma^{-1})$ such as the one from [25] (where it was also shown that this is a lower bound). By instantiating the corresponding honesty-fraction parameters for the best case to be close to 1, e.g., $\gamma_{\text{BC}} = 0.95$, our protocol shaves off almost a factor of γ_{WC}^{-1} from the communication complexity in the optimistic case. In particular, if $\gamma_{\text{WC}} = 0.5$, it cuts the per-party communication complexity almost in half in the optimistic case compared to using only the worst-case protocol.

Further note that one can also instantiate the optimistic good-case protocol with protocols that do not provide *any* guarantees for the safety of the overall protocol to be guaranteed. In particular, this means that algorithms not designed for Byzantine adversaries aiming to build for example minimum spanning trees such as the Plumtree algorithm [22] (as used in Ethereum [35]) may be used optimistically while still having provable worst-case fallback guarantees if the network is under attack.

1.3 Technical Overview

Overview of PushPullFlood. At a high level, the protocol PushPullFlood works by first "pushing" a message to a large fraction of the parties and then allowing the remaining parties that may not have received the message to "pull" the message from those that have received it, similarly to the seminal work of [10]. The protocol is build modularly by being parametric in two sub protocols: 1) a flooding protocol that must ensure to push a small notification to all parties and 2) a *fractional flooding protocol* that must guarantee to push the message to at least a constant fraction of the honest parties. Combining these, our construction relies on three key techniques to ensure the efficiency of the pull phase:

1. We use a verifiable random function (VRF) to let a party prove that they are allowed to pull the message from another party, similar to [7]. That is, the pulling party will evaluate the VRF to obtain a seed and use this seed to select whom they will pull from. A party

receiving such pull requests will then verify that this seed indeed was the output of the VRF to confirm the validity of the pull request. Thereby, we ensure that honest parties do not need to answer an excessive amount of malicious pull requests while still ensuring that all honest pull requests will be answered. To prevent malicious parties from choosing their VRF keys in such a way that many of them can pull from the same honest party, the VRF is evaluated on an unpredictable value obtained from a randomness beacon, which is assumed to be available to all parties.

2. We split the message into multiple shares using erasure correcting codes while ensuring that a constant fraction of these is sufficient to reconstruct the message, similar to [25]. By letting the answer to a pull request be such individual share, we ensure that honest parties can send sufficiently many pull requests to be certain to talk to a fraction of parties answering such pull request honestly, without allowing an adversary to exploit this to induce excessive communication.
3. We accompany such shares with membership proofs for a cryptographic accumulator similar to [25]. This ensures that honest parties are able to recognize which shares belong together, allowing honest parties to reconstruct efficiently.

Together, these techniques ensure that the per-party communication complexity of the pulling phase is asymptotically optimal and thereby improves the efficiency of previous protocols with a similar design. Note that while a certain message length is required for the protocol to be asymptotically optimal, the usage of erasure correcting codes and cryptographic accumulators was demonstrated to be concretely advantageous in terms of per-party communication complexity for a push-based protocol with messages of a length as small as 2 kilobytes in [25]. In addition to the cryptographic overhead, it must make sense to first send a small notification instead of the full message, so messages should be substantially larger than such a notification. Overall, the protocol is well suited for disseminating blocks in a blockchain system, where each block contains many transactions.

▶ Remark 3. The use of the VRF and the randomness beacon prevent denial-of-service attacks. Often, they are dealt with on the network level by rate-limiting, blocking IP addresses, etc. There are two reasons why such methods are less effective in our setting: First, an adversary may control a large number of nodes, so even a single pull request from every malicious node could overwhelm an honest party. Secondly, a malicious pull request not only incurs incoming traffic to the targeted party, but also forces the party to actively answer the request, leading to additional outgoing communication. Therefore, such attacks are more damaging to our protocols than, e.g., in pure push-based flooding protocol. Nevertheless, if denial-of-service attacks are not a concern, e.g., in a permissioned setting with a limited number of nodes, our protocols can be simplified by omitting the usage of VRFs.

While the idea of using a push-pull based approach is not new [10, 22], to the best of the authors knowledge, this is the first time a provably secure construction with a formal security proof is presented. Additionally, it is the first time erasure correcting codes are used to obtain asymptotically optimal communication complexity in a protocol utilizing pulling.

Overview of OptimisticFlood. We present the first optimistic flooding protocol OptimisticFlood. The protocol is also build modularly and takes two protocols as parameters: 1) A flooding protocol used for the optimistic case and 2) a flooding protocol guaranteed to work even in the worst-case. Using these two, the protocol runs in three phases:

Phase 1: The message is sent using the optimistic case flooding protocol.

Phase 2: The sender estimates whether the message has been received by sufficiently many honest parties by asking a committee of parties whether they have received it.

Phase 3: Depending on the conclusion of Phase 2, one of the following steps is taken:

- If it is concluded that sufficiently many honest parties have received the message, any party that did not receive it is allowed to pull the message from some random peers.
- Otherwise, the sender defaults back to sending the message using the worst-case protocol.

If the protocol is executed in a setting fulfilling the optimistic conditions, then the optimistic case protocol ensures delivery to all honest parties and we set the parameters such that the adversary cannot force defaulting back to the worst-case protocol. Hence, the adversary can only induce additional complexity by pulling, which is ensured to be minimal using the same techniques as for the pull-phase of PushPullFlood.

On the other hand, if the protocol is executed in a setting not fulfilling the optimistic conditions, we do not obtain any guarantees from the optimistic case flooding protocol about how many parties receive the message. This is not a problem if the sender in Phase 3 defaults back to the worst-case protocol. However, an adversary may try to convince the sender that the optimistic protocol succeeded even though it did not. We solve this by carefully tweaking the parameters of the protocol to ensure that if the sender concludes that the protocol succeeded, then it is guaranteed that at least a fraction of the honest parties have received the message. By setting the parameters of the erasure correcting codes correspondingly, we ensure that the remaining honest parties receive the message when pulling.

1.4 Related Work

There is an extensive line of work on message dissemination and flooding protocols. While classic epidemic algorithms and gossip protocols [10, 11, 17, 20, 21] focused mainly on the crash failure setting, a recent line of work [7, 24, 25, 28] introduced flooding protocols that are resilient against byzantine adversaries. Such protocols follow graph-theoretic techniques such as [21], relying on the fact that the graph induced by the neighbor selection procedure among honest parties remains connected. Most recently [25] showed that worst-case per-party communication complexity of a flooding protocol is lower-bounded by $\Omega(l \cdot \gamma_{\text{ACTUAL}}^{-1})$ where l denotes the length of the message and γ_{ACTUAL} denotes the fraction of parties remaining honest. The same work presents a protocol achieving $O(l \cdot \gamma_{\text{WC}}^{-1})$ worst-case per-party communication complexity where γ_{WC} denotes the worst-case fraction of parties remaining honest and thereby almost matches the lower-bound.

When the protocol PushPullFlood, presented in this paper, is instantiated with suitable parameters, it matches that worst-case bound. Further, the protocol OptimisticFlood instantiated with suitable parameters has a per-party communication complexity of only $O(l \cdot \gamma_{\text{BC}}^{-1})$ when $\gamma_{\text{ACTUAL}} \geq \gamma_{\text{BC}}$ while asymptotically matching the protocol of [25] in the worst-case (i.e., if $\gamma_{\text{BC}} > \gamma_{\text{ACTUAL}} \geq \gamma_{\text{WC}}$). Thereby, we improve upon the state of the art for provably secure protocols when the actual fraction of honest parties is higher than in the worst-case.

In contrast to this, other lines of work that target efficiency by following heuristic approaches to minimize per-party communication complexity and latency [16, 34, 37]. A detailed overview of existing protocols can be found in [25].

To the best of our knowledge, current flooding protocols secure under Byzantine corruptions focus on the worst-case performance and do not explicitly attempt to improve the efficiency under optimistic conditions. Nevertheless, there is an extensive literature on optimistic protocols for agreement primitives.

Optimistic latency. The work of Abraham, Nayak, Ren and Xiang [2] considers Byzantine fault-tolerant broadcast and optimizes the *good-case* latency, measured as the number of rounds for all honest parties to commit when the designated broadcaster is honest.

Another traditional line of work investigates protocols that have a number of rounds proportional to the actual number of corruptions f, rather than a known upper bound on the number of corruptions t. In this case, it is known that deterministic broadcast solutions have $\min\{f+2, t+1\}$ rounds of communication [13,14]. A long line of works focused on feasibility results, including protocols without setup [1,3,5,12,13,18,33,36], or protocols with a setup for cryptographic (pseudo-)signatures [9,27,31]. A different line of work optimizes *the delay* of each round (rather than the number of rounds), by making progress as fast as the actual network delay in an optimistic case when the number of corruptions is small [23,30].

Optimistic communication. The work [6] considered protocols with optimistic communication $O(nf)$ for byzantine agreement, improving upon traditional protocols that achieved $O(nt)$ communication.

2 Model and Preliminaries

2.1 Model

We assume a synchronous network, i.e., that all parties are connected via point-to-point channels which guarantee delivery within a known time Δ_{CHANNEL}. We additionally assume that the actual fraction of honest parties γ_{ACTUAL} is at least some worst-case bound on the number of honest parties $\gamma_{\text{WC}} \in (0,1]$. We assume that all parties have access to a PKI (public-key infrastructure). That is, all parties p_i have a public key $\texttt{pk}_i$ and a secret key $\texttt{sk}_i$ where the former is known by all other parties. Additionally, we assume that all parties have access to a randomness beacon, which periodically generates unpredictable random values. These values are updated at regular intervals, referred to as *epochs*, and are accessible to all parties for both the current and past epochs. Such randomness beacons are used in many blockchain protocols, e.g., for leader election in proof-of-stake protocols [8,15].

▶ Remark 4. When one of our protocols is used to disseminate messages in a blockchain network, the functioning of the blockchain and consequently the randomness beacon and progressing epochs rely on message delivery of the flooding protocol. Since our protocols assume such a randomness beacon with access to epochs, we need to avoid circular dependencies. One way to achieve this is to upgrade an already functioning blockchain with a traditional flooding protocol to use our optimistic flooding protocol: In that case, the randomness beacon is already working and the current epoch is known. Our protocols can thus be used and are guaranteed to reliably deliver messages in the current epoch. This in turn guarantees the blockchain to progress until the next epoch, and so on.

Notation. Let $\texttt{H}\colon \{0,1\}^* \to \{0,1\}^\kappa$ denote a collision-resistant hash-function and let $\{\!\{a,a,b\}\!\}$ denote a multiset containing the elements a, a, and b. In Table 1, we summarize the notation used in this paper. Note that several symbols are used in the context of our protocols and their meaning will become clear later.

2.2 Primitives

Below we define properties of flooding protocols and introduce basic primitives our protocols will use and briefly discuss how they can be instantiated.

Table 1 Overview of commonly used symbols.

Symbol	Meaning	Symbol	Meaning
$\mathcal{P}$	Set of all parties.	γ_{BC}, γ_{WC}	Best/worst case honest parties fraction.
n	Number of parties, $n = \lvert\mathcal{P}\rvert$.	c, T	Committee size and complaint threshold.
κ	Security parameter.	k	Number of received complaints.
l	Message length in bits.	η	Bound on the number of pulling parties.
Δ	Delivery time upper bound.	ζ	Erasure coding scheme.
ψ	Randomness beacon value.	μ, τ	Number of shares and tolerated erasures.
$\mathtt{pk}$, $\mathtt{sk}$	Public and secret key.	s	Share of erasure coding scheme.
r, π^{vrf}	Output and proof of VRF.	α	Cryptographic accumulation scheme.
γ_{Actual}	Actual honesty fraction.	z, π^{acc}	Accumulated value and proof.

Flooding. We use a property-based definition of flooding as it was shown in [25] that security w.r.t. this definition implies a secure implementation of the flooding functionality in the UC framework [4]. However, unlike previous property based definitions, we include the fraction of parties that must remain honest for the delivery guarantee to apply in the definition. This allows us to capture optimistic protocols that perform better when the actual fraction of honest parties is high.

▶ **Definition 5** (Flooding). *A* flooding protocol *is a protocol* Π *executed by parties* $\mathcal{P}$*, where each party* $p \in \mathcal{P}$ *can input a message at any time, and as a consequence, parties may get a message as output.*

▶ **Definition 6** ((γ, Δ)-delivery). *We say that a flooding protocol* Π *has* (γ, Δ)-delivery *for* $\gamma \in [0, 1]$ *and* $\Delta > 0$ *if the following holds: When an* honest *party inputs a message* m *at time* t *and* $\gamma_{\text{Actual}} \geq \gamma$*, then all other honest parties output* m *by time* $t + \Delta$*, except with probability negligible in the security parameter* κ*.*

Additionally, we define the key metric for flooding networks most relevant for this paper namely *per-party communication complexity.*

▶ **Definition 7** (Per-Party Communication Complexity). *Let* Π *be a flooding protocol and let* $l \in \mathbb{N}$ *be the bit-length of a message* m*. We say that the* per-party communication complexity *of the protocol* Π *to send a message of length* l *is bounded by* X *if for any adversary, the probability that there is an honest party sending more than* X *bits as a consequence of an honest sender having input* m *is negligible in the security parameter* κ*. We write* $\mathtt{PPCC}(\Pi, l) \leq X$ *to denote this.*

Note that the delivery time parameter Δ is a crucial metric for flooding protocols. To keep the per-party communication asymptotically optimal and in particular independent of the number of parties n, the delivery time must grow at least logarithmically with n. Our protocols, however, introduce only a small constant overhead beyond the latency of the underlying protocols we use in a black-box manner.

Verifiable Random Function (VRF). In our constructions, we will use a VRF to prevent corrupt parties creating excess network traffic. Below, we define an abstraction of VRF as a pair of two algorithms with the standard properties defined in [19].

▶ **Definition 8** (Verifiable Random Function). *A pair* `vrf` *is a VRF if it consists of the following two algorithms:*

- `vrf`.*Eval: An evaluation algorithm that takes an input* $i \in \{0,1\}^*$ *and a secret key* `sk` *as parameters and outputs an output* r *and proof* π^{vrf}.
- `vrf`.*Verify: A verification algorithm that takes an input* i*, an output* r*, a proof* π^{vrf}*, and a public key* `pk` *as parameters and outputs a boolean value* $b \in \{\bot, \top\}$.

With the following properties:

Full uniqueness: *An adversary cannot find a public key* `pk`*, an input* i*, two different outputs* $r_1 \neq r_2$*, and proofs* π_1^{vrf}, π_2^{vrf} *s.t.* $\texttt{vrf}.\mathit{Verify}(i, r_1, \pi_1^{\mathsf{vrf}}) = \top$ *and* $\texttt{vrf}.\mathit{Verify}(i, r_2, \pi_2^{\mathsf{vrf}}) = \top$.

Full pseudorandomness: *An adversary not knowing a secret key* `sk` *cannot distinguish the output value* r *of* $\texttt{vrf}.\mathit{Eval}(i, \texttt{sk})$ *(for any input* i *chosen by the adversary) from a value drawn uniformly at random without the proof* π^{vrf}.

For simplicity, we will assume all these properties to hold throughout all our executions and disregard the negligible probability that they do not hold. Note that there exist several simple implementation of such a VRF [19].

Erasure correcting code scheme. Our protocols will make use of erasure correcting codes. Below, we recap the definition given in [25].

▶ **Definition 9** (Erasure Correcting Code Scheme). *Let* $\mu \in \mathbb{N}$ *be the number of shares, and let* $\tau \in \mathbb{N}$ *be the number of erasures that are to be tolerated. A pair of algorithms* ζ *is a* (μ, τ)*-erasure-correcting-code-scheme (abbreviated* (μ, τ)*-ECCS) if it consists of two deterministic algorithms:*

- ζ.*Enc: An encoding algorithm that takes a message* $m \in \{0,1\}^*$ *and produces a sequence of shares* $s_1, \ldots, s_\mu$.
- ζ.*Dec: A decoding algorithm that if a sequence of shares* $s'_1, \ldots, s'_\mu$ *s.t. it holds for at least* $\mu - \tau$ *of them that* $s'_i = s_i$ *and for the remaining* $s'_i = \bot$ *is input, then the original message* m *is returned.*

We will use the notation $\zeta.\texttt{ShareSize}(l)$ for a function that bounds the size of each share when a message of length l is encoded.

Note that we here assume the algorithms to be deterministic, which we will exploit in our constructions. While this is not the case for all erasure correcting codes, e.g., Reed-Solomon codes [32] are deterministic. Using Reed-Solomon codes with the same encoding of messages as in [25], we obtain $\zeta.\texttt{ShareSize}(l) = O\left(\frac{l}{\mu - \tau}\right)$.

Weak cryptographic accumulator. In our protocols, we will make use of a weak form of cryptographic accumulators. Below we recap the definition from [25].

▶ **Definition 10** (Weak Static Cryptographic Accumulation Scheme). *A pair of algorithms* α *is a* weak static cryptographic accumulation scheme *(abbreviated WSCAS) if it consists of two deterministic algorithms:*

- $\alpha.\mathit{Accumulate}(\{m_1, \ldots, m_\lambda\})$: *An algorithm for accumulating a set of values* $\{m_1, \ldots, m_\lambda\}$*. It returns an accumulated value* z *and a sequence of proofs* $\pi_1^{\mathsf{acc}}, \ldots, \pi_\lambda^{\mathsf{acc}}$ *where* π_i^{acc} *can be used to prove that* m_i *is in the accumulated value* z *where each* $m_i \in \{0,1\}^*$.
- $\alpha.\mathit{Verify}(m, \pi^{\mathsf{acc}}, z)$*: A function that checks if a proof* π^{acc} *proves that a message* m *was in the set of elements used to create the accumulated value* z.

With the following properties:

Completeness: *All honestly generated proofs are accepted by* $\alpha.\mathsf{Verify}$.

Collision-freeness: *No polynomial-time adversary can find a set of values* $M := \{m_1, \ldots, m_\lambda\}$*, a value* $m' \notin M$*, and a proof* π^{acc} *such that* $\alpha.\mathsf{Verify}(m', \pi^{\mathsf{acc}}, z) = \top$ *for* $z \leftarrow \alpha.\mathsf{Accumulate}(M)$.

We use the notation $\alpha.\texttt{AccSize}$ for a bound on the size of the accumulated value and $\alpha.\texttt{ProofSize}(\lambda)$ for a function that bounds the size of each proof as a function of the number of messages accumulated λ.

Note that we here assume deterministic accumulators. Similarly to erasure correcting codes, this is needed for our constructions. One can use Merkle trees [29] to instantiate such a WSCAS, where the accumulated value is the root of the Merkle tree and the proofs are the Merkle proofs for each message. This yields an efficient deterministic scheme with

$$\alpha.\texttt{AccSize} = O(\kappa) \quad \text{and} \quad \alpha.\texttt{ProofSize}(\lambda) = O(\log(\lambda) \cdot \kappa). \tag{1}$$

3 Warmup: PushPullFlood

We design a new flooding protocol based upon the push-pull paradigm from [10], by first pushing a message to a fraction of the parties and then letting the remaining parties pull the message from those that have received it. The original push-pull protocol proposed in [10] relied on periodic pulling but is unfortunately not practical in the Byzantine setting because of two issues:

1. The time period between pull requests must be set. If it is too low, it takes a long time before a message reaches everybody. If it is too high, a lot of excess traffic is created.
2. Byzantine parties may issue an excessive amount of pull request and because honest parties cannot determine whether these request are malicious, they have to answer the requests. Thereby, the bandwidth of honest parties may be exhausted.

We address Issue 1 by letting our new protocol take an existing flooding protocol as a parameter and use this to notify parties that a message is being flooded. Thereby parties can simply issue pull requests when they receive such notification but no message. At first it may seem paradoxical to design a flooding protocol that is parameterized by an existing flooding protocol. However, note that the existing flooding protocol is only used to flood notifications showing that a message has been flooded, and such notifications are therefore significantly shorter than the actual message being flooded. Therefore, previous provably secure flooding protocols [24, 28] for short messages can be used to instantiate this protocol without blowing up the communication complexity of this new protocol.

A naive way to limit the number of pull requests that honest parties must answer is to have each honest party enforce a local (possibly statistical) cap on the number of requests they respond to. However, to prevent dishonest parties from exhausting this response budget – thereby blocking genuine pull requests – such limits would need to be enforced per sending party. Unfortunately, even if an honest party answers just one pull request from each other party, it would still induce a per-party communication cost of $\Omega(n \cdot (1 - \gamma_{\text{WC}}))$, which is already too high. Instead, a partial solution to Issue 2 is to use a VRF to enforce from which neighbors a party is allowed to pull a message from, similar to how connections are established in [7]. In more detail, each party uses a VRF to obtain a random seed, which is then used to deterministically sample the set of parties from which it pulls. Parties receiving pull requests can thus verify the VRF proof and ignore illegitimate pull requests. To further

prevent malicious parties from biasing their VRF keys to skew the sampling, parties evaluate the VRF on an unpredictable value from the randomness beacon, which is updated every epoch.

To ensure that an honest party pulls from at least one honest party, they must be allowed to pull from at least $\Omega(\kappa)$ neighbors. Thus, if parties are allowed to pull the *entire message* from all their neighbors, adversaries can induce a communication of least $\Omega(\kappa \cdot l \cdot (1-\gamma_{\text{ACTUAL}}) \cdot n)$ by letting all corrupt parties pull for a message of length l from all their neighbors. To overcome this, we adapt the techniques from [25] to work for pulling instead of pushing. That is, to reduce the communication complexity of the pull phase, we let parties pull only an erasure correcting share of the original message from each neighbor. As a final ingredient and similarly to [25], we use a weak cryptographic accumulator to let honest parties recognize which shares belong together and thereby ensure that they can reconstruct the message.

We next describe our push-pull protocol for the Byzantine setting. We describe the pull step in a modular fashion, as we will reuse this step for our optimistic flooding protocol in later sections. We therefore first introduce the protocol Pull and analyze its communication complexity before presenting the protocol PushPullFlood.

3.1 Pulling

The pull-phase of the protocol has the purpose of ensuring that if a constant fraction of the parties already knows a message, then if the remaining parties begin pulling, they are all able to obtain the message with a per-communication complexity of just $O(n \cdot l \cdot \gamma_{\text{WC}}^{-1})$.

The protocol makes use of a VRF and the randomness beacon we assume to be available to all parties (see Section 2.1). We assume that the VRF keys of all parties participating in the protocol are generated independently of the current (and future) values of the randomness beacon. This means in practice, users have to register their VRF keys at least one epoch before participating in the protocol. This prevents malicious parties from repeatedly generating VRF keys to skew the probabilities in their favor.

Protocol Pull(ζ, α)

The protocol takes the following parameters:
ζ: An (μ, τ)-ECCS.
α: A WSCAS.
Each party $p_i \in \mathcal{P}$ keeps track of a set of shares received for a particular accumulator z, $\texttt{ReceivedShares}_i[z]$ and a set of received messages $\texttt{Messages}_i$.
Each party accepts the following three commands:

Pull message: When a party $p_i \in \mathcal{P}$ gets the input (Pull, h) they do the following:

1. Obtain random value ψ from randomness beacon for the current epoch.[a]
2. Use $\texttt{vrf}.\mathsf{Eval}((\psi, h), \texttt{sk}_i) = (r, \pi^{\mathsf{vrf}})$ to obtain a random seed r and a proof π^{vrf}. Use the seed to deterministically sample with replacement a multiset $S = \{\!\{p_1, \ldots, p_\mu\}\!\}$ from $\mathcal{P}$ s.t. $|S| = \mu$, and S is distributed uniformly for uniform r.
3. Now, the parties in S are deterministically enumerated $S = \{\!\{p_1, \ldots, p_\mu\}\!\}$. For each party $p_j \in S$ they send $(\mathit{Pull}, r, h, \pi^{\mathsf{vrf}}, j)$ to this party.
4. Whenever a party $p_j \in S$ responds with the requested share and a proof of which accumulator it belongs to $(\mathit{Share}, s_j, \pi^{\mathsf{acc}}, z)$, p_i first check that the proof π^{acc} verifies that the share s_i belongs to the accumulator z. If both checks pass, then p_i adds s_i to $\texttt{ReceivedShares}_i[z]$.

5. When a party has received sufficiently many shares they will reconstruct the shares to get a message m' which they will add to $\texttt{Messages}_i$.

Accept pull: When a party $p_i \in \mathcal{P}$ gets the input $(\mathit{AcceptPull}, m)$ they first share the message m into shares $\zeta.\mathsf{Enc}(m) = s_1, \ldots, s_\mu$. Furthermore, they obtain an accumulated value and proofs for each share and its share number $z, \pi_1^{\mathsf{acc}}, \ldots, \pi_\mu^{\mathsf{acc}} = \alpha.\mathsf{Accumulate}(\{(s_j, j) \mid 1 \leq j \leq \mu\})$.

Afterwards, whenever, a message $(\mathit{Pull}, r, h, \pi^{\mathsf{vrf}}, j)$ is received from a party p_j for the first time, then party p_i checks that p_j should send this message to p_i. This is done by obtaining the random value ψ from randomness beacon for the current epoch, checking that $\texttt{vrf}.\mathsf{Verify}((\psi, h), r, \pi^{\mathsf{vrf}}, \texttt{pk}_j) = \top$, and checking that p_j was indeed sampled as the j'th party using the seed r. If this check passes, the party sends the accumulator and share values $(\mathit{Share}, s_j, \pi^{\mathsf{acc}}, z)$ that belong to the hash value h to party p_j.

Get messages: On input $(\mathit{GetMessages})$ to p_i the party returns the set of messages they have received $\texttt{Messages}_i$.

[a] We here assume for simplicity that all parties agree on what the current epoch is. In practice, this can be achieved by letting the sender of message include the epoch number ν in the message and propagate this epoch number to protocol calls, i.e., the party would in this case get the input (Pull, ν, h) instead of (Pull, h) etc.

We now state and prove the necessary property of this protocol.

▶ **Lemma 11.** *Let $\delta \in (0, 1]$, let ζ be an (μ, τ)-ECCS and let α be a WSCAS. For any $\beta \in (0, 1]$ and any message m, if*

1. *at least $\beta \cdot n$ of the parties are honest and have input $(\mathit{AcceptPull}, m)$,*
2. *the last honest party inputs $(\mathit{Pull}, \mathtt{H}(m))$ to the protocol $\mathit{Pull}(\zeta, \alpha)$ at time t,*
3. *and $\tau \geq \mu \cdot (1 - (1 - \delta) \cdot \beta)$,*

then the probability that some party has not received the message m by time $t + 2 \cdot \Delta_{\mathrm{CHANNEL}}$ is less than $\gamma_{\mathrm{ACTUAL}} \cdot n \cdot e^{-\frac{\delta^2 \cdot \mu \cdot \beta}{2}}$.

Proof. Consider an honest party p that has not received $(\mathit{AcceptPull}, m)$ as input. We introduce indicator random variables $X_1, \ldots, X_\mu$ where X_j indicates whether the j'th party from which p requests a share is honest and has already received $(\mathit{AcceptPull}, m)$ before time t. Since we assume both the erasure correcting code as well as the weak cryptographic accumulator to be deterministic, that party will in this case have generated the same shares and accumulator proofs as other parties. Therefore, p will in this case have received share j and a valid proof by time $t + 2 \cdot \Delta_{\mathrm{CHANNEL}}$. Further, note that if p has received at least $\mu - \tau$ valid shares by time $t + 2 \cdot \Delta_{\mathrm{CHANNEL}}$, then p is able to reconstruct the original message timely by the correctness property of the ECCS and the unforgeability of the WSCAS. Therefore, by the assumption that

$$\tau \geq \mu \cdot (1 - (1 - \delta) \cdot \beta) \iff \mu - \tau \leq (1 - \delta) \cdot \mu \cdot \beta, \tag{2}$$

we have

$$\begin{aligned}
&\Pr\left[p \text{ has not received message } m \text{ by time } t + 2 \cdot \Delta_{\mathrm{CHANNEL}}\right] \\
&\leq \Pr\left[\sum_{j=1}^{\mu} X_j \leq \mu - \tau\right] \\
&\leq \Pr\left[\sum_{j=1}^{\mu} X_j \leq (1 - \delta) \cdot \mu \cdot \beta\right].
\end{aligned} \tag{3}$$

Parties sample the μ neighbors with replacement using a random seed r obtained from the VRF given the message hash and a fresh value from the randomness beacon. Since we assume the randomness beacon produces unpredictable values, independent from the VRF keys, the X_j are computationally indistinguishable from independent and identically distributed values. We further note that, up to some negligible distinguishing advantage, the expected value of any X_j is given by $\mathrm{E}[X_j] = \beta$, and there Chernoff implies

$$\Pr\left[\sum_{j=1}^{\mu} X_j \leq (1-\delta)\cdot\mu\cdot\beta\right] \leq e^{-\frac{\delta^2\cdot\mu\cdot\beta}{2}}. \tag{4}$$

Noting that there are at most $\gamma_{\text{ACTUAL}}\cdot n$ such parties and using a union-bound together with Equation (3), we get the desired bound

$$\begin{aligned}&\Pr[\text{some honest party has not received message } m \text{ by time } t + 2\cdot\Delta_{\text{CHANNEL}}]\\ &\leq \gamma_{\text{ACTUAL}}\cdot n\cdot e^{-\frac{\delta^2\cdot\mu\cdot\beta}{2}}.\end{aligned} \tag{5}$$

◀

Communication complexity of pulling. Let us now analyze the communication complexity of the pulling protocol. We first concentrate on the communication complexity induced for a party to pull a message. For each party pulling, there will be μ pulling requests. The pulling requests will each consist of:

1. A tag *Pull* of size $O(1)$,
2. the random seed from the VRF to determine whom to pull from of size $O(\kappa)$,
3. the proof that this seed has been correctly calculated of size $O(\kappa)$,
4. the hash of the message of size $O(\kappa)$,
5. and the index of the requested share of size $O\big(\log(\mu)\big)$.

Therefore, each such pull request will have size

$$O(1) + O(\kappa) + O(\kappa) + O(\kappa) + O\big(\log(\mu)\big) = O\big(\kappa + \log(\mu)\big), \tag{6}$$

and the total communication complexity for such pulling party will be

$$\mu\cdot O\big(\kappa + \log(\mu)\big) = O\big(\mu\cdot(\kappa + \log(\mu))\big). \tag{7}$$

Next, let us analyze the communication complexity of responding to such *valid*[1] pull requests. A response to such pulling request will consist of:

1. A tag *Share* of size $O(1)$,
2. a share of size $\zeta.\texttt{ShareSize}(l)$,
3. the accumulated value of all shares and indexes with size $\alpha.\texttt{AccSize}$,
4. and the proof that the share is a part of the accumulator with size $\alpha.\texttt{ProofSize}(\mu)$.

Therefore, each such response will have size

$$O(1) + \zeta.\texttt{ShareSize}(l) + \alpha.\texttt{AccSize} + \alpha.\texttt{ProofSize}(\mu). \tag{8}$$

The total communication complexity for a party having accepted pulls, will therefore be what is given in Equation (8) multiplied with the number of such valid pull requests the party receives.

[1] That is, a pull request where the attached VRF-proof proofs that the share should actually be requested from this specific party.

By the pseudorandomness property of the VRF, a verifying pull requests can only be established by letting the party knowing their secret key evaluate the VRF.[2] To upper bound the per-party communication complexity let η be an upper bound on the number of secret keys that are used to evaluate the VRF for a particular message. Each VRF evaluation gives rise to μ pull requests. The outputs of each such evaluation from a secret key is guaranteed to be unique by the full uniqueness property which ensures that there are therefore at most $\eta \cdot \mu$ valid pull requests in total for a particular message.

For a particular party p, we introduce an indicator random variable X_i for each of the pull requests $i \in \{1, \dots \eta \cdot \mu\}$ where $X_i = 1$ if and only if the i'th of these potential pull requests targets p as a valid receiver of the pull request. As the target of a valid pull request is drawn uniformly at random among all parties (up to some negligible distance by the pseudorandomness property of the VRF and the definition of the protocol), we have for any X_i that the expected value is $\mathrm{E}[X_i] = n^{-1}$ and therefore $\mathrm{E}\left[\sum_{i=1}^{\eta \cdot n \cdot \mu} X_i\right] = \eta \cdot \mu \cdot n^{-1}$. Further, because the sampling is done with replacement, we have as in the proof of Lemma 11 that the values X_i are indistinguishable from independent and identically distributed random variables. We can thus apply the Chernoff bound to obtain (up to negligible distance) for any $\delta \in [0, 1]$

$$\Pr\left[\sum_{i=1}^{\eta \cdot n \cdot \mu} X_i \geq (1+\delta) \cdot \eta \cdot \mu \cdot n^{-1}\right] \leq e^{-\frac{\delta^2 \cdot \eta \cdot \mu}{3 \cdot n}}. \tag{9}$$

Further, using the union bound, we can bound the probability that any party receives more pull requests than what we used in the bound above:

$$\Pr\left[\exists p \text{ receiving more than } (1+\delta) \cdot \eta \cdot \mu \cdot n^{-1} \text{ pull requests}\right] \leq n \cdot e^{-\frac{\delta^2 \cdot \eta \cdot \mu}{3 \cdot n}}. \tag{10}$$

Letting δ be constant, we note that the probability that any party receives more than $O\big(\eta \cdot \mu \cdot n^{-1}\big)$ valid pull requests is negligible in κ when $\mu \geq 3 \cdot n \cdot (\log(n) + \kappa) \cdot (\delta^2 \cdot \eta)^{-1}$. Hence, the per-party communication complexity for a party accepting pull requests with these parameters will be

$$O\big(\eta \cdot \mu \cdot n^{-1} \cdot (\zeta.\texttt{ShareSize}(l) + \alpha.\texttt{AccSize} + \alpha.\texttt{ProofSize}(\mu))\big). \tag{11}$$

3.2 Push-Pull Flooding

We now present the full protocol that combines the push and pull phases. Before presenting the actual protocol, we introduce a weakened delivery guarantee, that will be used for the push-phase of the protocol.

Fractional delivery. We here introduce a weakened version of the (γ, Δ)-delivery guarantee, namely a version where it is not required that a message is delivered to all parties, but rather only to a fraction of the parties. The idea is that we will use a protocol with this weaker delivery guarantee to spread out messages to a large fraction of the parties before the pulling phase is initiated. We dub this weakened property *fractional delivery* and define it formally below.

[2] To see that this follows from the pseudorandomness, consider for the sake of contradiction an adversary with a non-negligible probability of evaluating a VRF without knowing a corresponding secret key. This can be used to distinguish an output of the VRF from a uniformly random value non-negligibly by using the adversary capable of evaluating such VRF with a non-negligible probability and if the output matches the challenge, guess that it was produced by the VRF.

▶ **Definition 12** ((β, γ, Δ)-fractional delivery)**.** *We say that a flooding protocol* Π *has* (β, γ, Δ)-fractional delivery *for* $\beta, \gamma \in [0, 1]$ *and* $\Delta > 0$ *if the following holds: When a message* m *is input to an* honest *party at time* t *and* $\gamma_{\text{ACTUAL}} \geq \gamma$*, then at least a* β *fraction of honest parties output* m *by time* $t + \Delta$*, except with probability negligible in the security parameter* κ*.*

Sometimes, we will refer to a flooding protocol with this property as a *fractional* flooding protocol.

The idea of not delivering messages to all parties was also considered in [7].[3] However, their work builds a custom consensus protocol on top of the weaker message dissemination functionality. In contrast, our work uses the weaker property as a step towards building a flooding protocol that ensures message delivery to *all* parties.

Push and then pull flooding protocol. We now present our flooding protocol that works by first making the parties push out a notification for that a message is about to arrive (the hash of the message) and push out the actual message using a fractional flooding protocol. Afterwards, all parties are allowed to pull for the message if they did not receive the message that they were notified about.

Protocol $\mathsf{PushPullFlood}(\Pi_{\text{Flood}}, \Pi_{\text{FracFlood}}, \zeta, \alpha)$

The protocol takes the following parameters:
$\boldsymbol{\Pi_{\mathsf{Flood}}}$**:** a flooding protocol with $(\gamma_{\text{WC}}, \Delta_{\text{Flood}})$-delivery.
$\boldsymbol{\Pi_{\mathsf{FracFlood}}}$**:** a flooding protocol with $(\beta, \gamma_{\text{WC}}, \Delta_{\text{FracFlood}})$-fractional delivery.
$\boldsymbol{\zeta}$**:** A ECCS.
$\boldsymbol{\alpha}$**:** A WSCAS.
Each party p_i keeps track of a set of received messages $\texttt{Messages}_i$ that initially is empty, and runs an instance of the protocols $\mathsf{Pull}(\zeta, \alpha)$, Π_{Flood}, and $\Pi_{\text{FracFlood}}$.
Each party accepts the following two commands:
Send: When party p_i receives input (Send, m) they:

1. Send a hash of the message $(\mathit{Hash}, \mathtt{H}(m))$ to all parties using Π_{Flood}.
2. Send the message using $\Pi_{\text{FracFlood}}$.
3. Add m to $\texttt{Messages}_i$
4. Input $(\mathit{AcceptPull}, m)$ to $\mathsf{Pull}(\zeta, \alpha)$.

Get messages: On input $(\mathit{GetMessages})$ to p_i the party returns the set of messages they have received $\texttt{Messages}_i$.

Additionally, at all times the parties do the following:

1. Whenever a party p_i receives message h in the protocol Π_{Flood}, the party notes down the time t. If there is no message $m' \in \texttt{Messages}_i$ s.t. $\mathtt{H}(m') = h$ at time $t + \Delta_{\text{FracFlood}}$, then they issue (Pull, h) to $\mathsf{Pull}(\zeta, \alpha)$.
2. Whenever a party p_i receives a message m in the protocol $\Pi_{\text{FracFlood}}$, they add m to $\texttt{Messages}_i$ and input $(\mathit{AcceptPull}, m)$ to $\mathsf{Pull}(\zeta, \alpha)$.
3. Whenever a party p_i receives a message m in $\mathsf{Pull}(\zeta, \alpha)$, they add m to $\texttt{Messages}_i$.

Below, we state and prove the security of PushPullFlood.

[3] In particular, the F_{sync} functionality of [7, p. 7] allows a fraction of the parties to be eclipsed in which case the delivery guarantees will not apply.

▶ **Theorem 13.** *Let $\mu, \tau, \Delta_{Flood}, \Delta_{FracFlood} \in \mathbb{N}$, let $\beta, \delta \in (0,1]$, and let α be a WSCAS. If*
1. *Π_{Flood} ensures $(\gamma_{WC}, \Delta_{Flood})$-delivery,*
2. *$\Pi_{FracFlood}$ ensures $(\beta, \gamma_{WC}, \Delta_{FracFlood})$-fractional delivery,*
3. *and ζ is a (μ, τ)-ECCS with*
   a. *$\mu \geq 2 \cdot (\log(n) + \kappa) \cdot (\beta \cdot \gamma_{WC} \cdot \delta^2)^{-1}$ and*
   b. *$\tau \geq \mu \cdot (1 - (1-\delta) \cdot \beta \cdot \gamma_{WC})$,*

then $\mathsf{PushPullFlood}(\Pi_{Flood}, \Pi_{FracFlood}, \zeta, \alpha)$ ensures $(\gamma_{WC}, \Delta_{Flood} + \Delta_{FracFlood} + 2 \cdot \Delta_{CHANNEL})$-delivery.

Proof. Assume there are at least $\gamma_{\text{WC}} \cdot n$ honest parties and let m be a message input to some honest party at time t. We let
- A be the event that all honest parties have received m by time $\Delta_{\text{Flood}} + \Delta_{\text{FracFlood}} + 2 \cdot \Delta_{\text{CHANNEL}}$,
- B be the event that all honest parties have received $\mathtt{H}(m)$ by time $t + \Delta_{\text{Flood}}$,
- and let C be the event that at least a β fraction of the honest parties have received m by time $t + \Delta_{\text{FracFlood}}$.

By the law of total probability and the assumptions on Π_{Flood} and $\Pi_{\text{FracFlood}}$, we have that

$$\Pr[A] \geq \Pr[A \mid B \cap C] \cdot \Pr[B \cap C] \geq \Pr[A \mid B \cap C] \cdot (1 - \text{negl}(\kappa)). \tag{12}$$

Further, as $\Pr[A \mid B \cap C] = 1 - \Pr[\neg A \mid B \cap C]$, it is sufficient to prove that $\Pr[\neg A \mid B \cap C] \leq \text{negl}(\kappa)$. Note that C ensures that a β fraction of the honest parties, i.e., at least a fraction $\beta' := \beta \cdot \gamma_{\text{WC}}$, has received m in the protocol $\Pi_{\text{FracFlood}}$ by time $t + \Delta_{\text{FracFlood}}$, and thus has input $(\mathit{AcceptPull}, m)$ to $\mathsf{Pull}(\zeta, \alpha)$ by then. Further note that B ensures that by time $t + \Delta_{\text{Flood}} + \Delta_{\text{FracFlood}}$, all honest parties either have received m or have input $(\mathit{Pull}, \mathtt{H}(m))$ to $\mathsf{Pull}(\zeta, \alpha)$. Finally note that we have $\tau \geq \mu \cdot (1 - (1-\delta) \cdot \beta')$ by assumption. Hence, the preconditions for Lemma 11 are fulfilled, which gives us that:

$$\Pr[\neg A \mid B \cap C] \leq \gamma_{\text{ACTUAL}} \cdot n \cdot e^{-\frac{\delta^2 \cdot \mu \cdot \beta \cdot \gamma_{\text{WC}}}{2}} \leq n \cdot e^{-\frac{\log(n)+\kappa}{2}} \leq \text{negl}(\kappa). \tag{13}$$

Communication complexity of pushing and pulling. In the full version [26], we show that the per-party communication complexity of $\mathsf{PushPullFlood}$ is bounded by

$$\begin{aligned}&\mathsf{PPCC}(\mathsf{PushPullFlood}(\Pi_{\text{Flood}}, \Pi_{\text{FracFlood}}, \zeta, \alpha), l) \\ &\quad \leq \mathsf{PPCC}(\Pi_{\text{Flood}}, \kappa) + \mathsf{PPCC}(\Pi_{\text{FracFlood}}, l) + \tilde{O}\big(l \cdot (\beta \cdot \gamma_{\text{WC}})^{-1} + \kappa^2 \cdot (\beta \cdot \gamma_{\text{WC}})^{-1}\big).\end{aligned} \tag{14}$$

This implies that the protocol is asymptotically optimal for messages of length $l = \tilde{\Omega}(\kappa^2 \cdot (\beta \cdot \gamma_{\text{WC}})^{-1})$ for appropriate instantiations of the underlying protocols.

4 OptimisticFlood

In this section, we present our optimistic flooding protocol $\mathsf{OptimisticFlood}$ that has an optimistic path such that under certain conditions it is guaranteed to have a communication complexity that is much lower than in worst-case scenarios.

4.1 Protocol

Protocol intuition. Our protocol is parameterized by two flooding protocols: 1) a best-case flooding protocol that only works if some best-case conditions are fulfilled, 2) a worst-case flooding protocol that is ensured to work in all remaining cases. The best-case conditions will be that at least a fraction γ_{BC} of the parties remain honest throughout the execution.

A first skeleton of an optimistic flooding protocol relying on two such existing flooding protocols could look like the following:

1. Run a best-case protocol to disseminate the actual message.
2. Check if the best-case protocol succeeds. If not, default to sending the entire message using the worst-case protocol.

While the skeleton reads fairly straightforward, it is easier said than done to reliably detect if the best-case protocol fails while still tolerating a small fraction of corrupted parties. Because flooding protocols only ensure the delivery of the message when the initial sender is honest, a first step towards this could be to let the initial sender ask all parties if they have received the message. We refer to a party reporting that they have not received the message as a *complaint*. Based on the answers given by the parties, a decision must be taken on whether we default back to use the worst-case flooding protocol and use this to send the entire message. The decision procedure and the following actions should account for the following two cases depending on the actual fraction of parties being honest γ_{ACTUAL}:

$\gamma_{\mathsf{bc}} \leq \gamma_{\mathsf{Actual}}$: In this case, the best-case protocol is guaranteed to deliver the message to all honest parties and therefore the worst-case protocol should not be executed, independently of the actions of the malicious parties. In particular, these up to $n \cdot (1 - \gamma_{\text{BC}})$ malicious parties may complain about not having received the message even though they have.

$\gamma_{\mathsf{wc}} \leq \gamma_{\mathsf{Actual}} < \gamma_{\mathsf{bc}}$: In this case, we have no guarantees from the best-case flooding protocol about how many parties have received the message and the adversary can choose this arbitrarily (by delivering the message to specific parties). In particular, it may be that the adversary does not deliver the message to $n \cdot (1 - \gamma_{\text{BC}})$ parties.

Note that from the sender's point of view, it will be impossible to distinguish which of the two above cases they are in as an adversary can make the views appear exactly the same for the sender. So how do we ensure that $n \cdot (1 - \gamma_{\text{BC}})$ parties cannot force the execution of the worst-case protocol in the first case while ensuring that all honest parties receive the message in the second case?

To balance this we introduce a subsequent pull-phase in case we decide that the best-case protocol "succeeded" allowing parties not having received the message to pull similar to the pull-phase from the protocol PushPullFlood presented in Section 3. That is, instead of requiring that no honest party complains to conclude a success, we only require that not more than a fraction of the parties complain in order to conclude that the best-case protocol succeeded. Concretely, we introduce a threshold T for how many complaints we will accept and still conclude that the best-case protocol "succeeded". We will set this threshold such that $(1 - \gamma_{\text{BC}}) \cdot n$ parties cannot produce enough complaints to conclude that the protocol failed if it did not, but still it should ensure that a sufficient fraction of the honest parties have received the message to ensure that pull requests will be responded to appropriately. Thereby, we can use the protocol Pull without prohibitively high communication.

What is left is only to combat the impracticality of letting the sender ask *all* parties. We do this by letting the sender sample a subset of the parties and ask this subset of parties about whether or not they have received the message. This allows the sender to statistically conclude whether or not the best-case protocol succeeded.

Protocol description. Below, we present our protocol for optimistic flooding.

Protocol OptimisticFlood$(\Pi_{\mathrm{BC}}, \Pi_{\mathrm{WC}}, c, T, \zeta, \alpha)$

The protocol has the following parameters:

Π_{bc}: A flooding protocol that should work in the best-case ensuring delivery within Δ_{BC} time when at least a γ_{BC} fraction of the parties remains honest.

Π_{wc}: A flooding protocol that should work in the worst-case ensuring delivery within Δ_{WC} time when at least a γ_{WC} fraction of the parties remains honest.

c: The size of the subset the sender should ask for complaints.

T: A threshold that decides how many complaints are acceptable.

ζ: A ECCS.

α: A WSCAS.

Each party p_i keeps track of a set of received messages $\texttt{Messages}_i$ that initially is empty, and runs an instance of the protocols $\mathsf{Pull}(\zeta, \alpha)$, Π_{Flood}, and $\Pi_{\mathrm{FracFlood}}$.

Send: When a party s receives input (Send, m) they do the following:

1. The sender s sends $(\mathit{Message}, m)$ to all parties using Π_{BC}. We let the time that this happens be denoted t_{INIT}.
2. At time $t_{\mathrm{INIT}} + \Delta_{\mathrm{BC}}$[a] the sender uniformly at random (with repetition) samples a committee of parties $C = \{p_1, \ldots, p_c\} \subseteq \mathcal{P}$. For each party $p \in C$ the sender sends a direct message using an authenticated channel $(\mathit{Received?}, \mathtt{H}(m), t_{\mathrm{INIT}} + \Delta_{\mathrm{BC}})$.
3. At time $t_{\mathrm{INIT}} + \Delta_{\mathrm{BC}} + 2 \cdot \Delta_{\mathrm{CHANNEL}}$[b], the sender counts how many unique complaints they have received from valid committee members for $\mathtt{H}(m)$. We let the count be denoted by k, and based on this the sender does one of following two things:
   a. If $k > T$, then the sender sends the original message $(\mathit{Message}, m)$ to all parties using Π_{WC}.
   b. Otherwise if $k \leq T$, the sender initializes a pull-phase by signing and sending $(\mathit{PullPhaseBegun}, \mathtt{H}(m))$ to all parties using Π_{WC}.

Get messages: On input $(\mathit{GetMessages})$ to p_i, the party returns the set of messages they have received $\texttt{Messages}_i$.

Additionally, at all times the parties do the following:

- When party p_i receives a message $(\mathit{Received?}, h, t)$ over an authenticated channel from a party s, the party checks if there is any message $m \in \texttt{Messages}_i$ s.t. $\mathtt{H}(m) = h$ which has been received before time t.[c] If no such message exists, then they send $(\mathit{Complaint}, h)$ to s over the authenticated channel.
- When a party p_i receives $(\mathit{PullPhaseBegun}, h)$ over Π_{WC} at time t they do the following:
  1. If there is any message $m \in \texttt{Messages}_i$ s.t. $\mathtt{H}(m) = h$, then they input $(\mathit{AcceptPull}, m)$ to $\mathsf{Pull}(\zeta, \alpha)$.
  2. Otherwise, if no such message exists, then they input (Pull, h) to $\mathsf{Pull}(\zeta, \alpha)$ at time $t + \Delta_{\mathrm{WC}}$.[d]
- When party p_i receives a message m in either the best-case protocol, the worst-case protocol or the pulling protocol, they add it to $\texttt{Messages}_i$ and store the time they received it together with the message.

[a] This timing ensures that the best-case protocol have had sufficient time to deliver the message.
[b] At this time it is ensured that all honest parties' complaints have reached the sender.
[c] Note that it is necessary to require that the message have been received before time t to ensure that the number of complaints from the set of parties sampled accurately reflects the share of parties that have actually received the message. If this condition was not enforced, an adversary could choose to deliver the message to all the parties part of the committee once the sender sends the *Received?*-message to them.

[d] The reason that the party does not immediately input the message to the pull protocol, is that it must be ensured that sufficiently many parties have already input an accept of the message to the pulling protocol.

Note that for the provably secure protocols, we often have $\Delta_{\text{BC}}, \Delta_{\text{WC}} = O\big(\log(n) \cdot \Delta_{\text{CHANNEL}}\big)$. Hence, the direct communication steps that happen using a channel are comparatively "cheap" time-wise.

It is also worth noting that instead of using just two different flooding protocols, one could consider using three different protocols. A natural example of this would be to use one worst-case protocol for short messages (notifications), one worst-case protocol for long-messages, and one best-case protocol for long messages.

The described version of the protocol requires a synchrony assumption on the channels for safety (i.e., guaranteeing message delivery) in the worst case to ensure that sufficiently many complaints reach the sender in time. In the optimistic case, it does not require synchrony. However, if we instead changed the protocol to require a certain number of "confirmations" from parties having received the message instead of complaints, the protocol would achieve safety without relying on synchrony. On the other, it would still only achieve the best-case communication complexity under synchronous conditions and require slightly more communication in the optimistic case to send the confirmations. This allows fine-tuning the protocol for the specific settings it is deployed in.

4.2 Correctness and Communication Complexity

In this section, we prove the correctness of the protocol OptimisticFlood under relevant conditions and analyze its communication complexity. We start out by stating that if the actual fraction honest parties γ_{ACTUAL} is bigger than the best-case threshold γ_{BC}, then for certain parameters, we achieve the delivery guarantees of the best-case protocol.

▶ **Lemma 14** (Best-case correctness). *Let $c \in \mathbb{N}$ be the size of the committee, $T \in \mathbb{N}$ be the complaint threshold, Π_{WC} a protocol, ζ a ECCS, α be a WSCAS, and $\delta \in (0, 1]$. If*

1. *Π_{BC} has $(\gamma_{BC}, \Delta_{BC})$-delivery*
2. *and $T \geq (1 + \delta) \cdot (1 - \gamma_{BC}) \cdot c$*

then OptimisticFlood$(\Pi_{BC}, \Pi_{WC}, c, T, \zeta, \alpha)$ has $(\gamma_{BC}, \Delta_{BC})$-delivery and if $\gamma_{ACTUAL} \geq \gamma_{BC}$ and the sender is honest, then the probability that Step 3a is activated is less than $e^{-\frac{\delta^2 \cdot (1-\gamma_{BC}) \cdot c}{3}}$.

The intuition for Condition 2 is that it corresponds to requiring that under best-case conditions, the threshold for the number of complaints is set sufficiently large such that only dishonest parties in the committee cannot make the sender default back to using the worst-case protocol. We now proceed with the proof.

Proof. We note that because the entire message is immediately input to Π_{BC}, the delivery guarantees for Π_{BC} directly apply and hence OptimisticFlood$(\Pi_{\text{BC}}, \Pi_{\text{WC}}, c, T, \zeta, \alpha)$ has $(\gamma_{\text{BC}}, \Delta_{\text{BC}})$-delivery.

We now bound the probability that Step 3a is activated for an honest sender. Let the time that the honest sender sends a message m be denoted t_{INIT} and note that by the above, all honest parties have received the message at time $t_{\text{INIT}} + \Delta_{\text{BC}}$. Hence, no honest party will send back $(\mathit{Complaint}, \mathtt{H}(m))$ to the sender. It is therefore sufficient to show that the number of corrupted parties in the committee is at most T with overwhelming probability.

To do so, we let $X_1, \ldots, X_c$ denote indicator variables s.t. $X_i = 1$ if and only if party p_i of the committee C in Step 2 is corrupted and note that for the actual number of complaints k, it holds that $\sum_{i=1}^{c} X_i \geq k$. Further, note that

$$\mathrm{E}\left[\sum_{i=1}^{c} X_i\right] = (1 - \gamma_{\text{ACTUAL}}) \cdot c \leq (1 - \gamma_{\text{BC}}) \cdot c, \tag{15}$$

and that the variables are identically and independently distributed as the honest sender samples the committee at random *with repetition*. Hence, using the Chernoff bound, we conclude that

$$\Pr\left[\sum_{i=1}^{c} X_i \geq (1+\delta) \cdot (1 - \gamma_{\text{BC}}) \cdot c\right] \leq e^{-\frac{\delta^2 \cdot (1-\gamma_{\text{BC}}) \cdot c}{3}}. \tag{16}$$

Therefore, by Condition 2, we have $\Pr[k > T] \leq e^{-\frac{\delta^2 \cdot (1-\gamma_{\text{BC}}) \cdot c}{3}}$. ◀

Next, we state that when the parameters of the protocol are instantiated carefully, then the protocol also ensures delivery assuming just the worst-case bound on the number of honest parties. It is worth noting that the theorem only makes assumptions about the worst-case protocol. Hence, message delivery is ensured independently of which best-case protocol is deployed. In particular, this allows to use protocols that based on for example heuristics about practice.

▶ **Lemma 15** (Worst-case correctness). *Let $\tau, \mu, T, c \in \mathbb{N}$, let Π_{BC} be a protocol, let α be a WSCAS, and let $\delta_1, \delta_2, \beta \in (0, 1]$. If*

1. *Π_{WC} has $(\gamma_{WC}, \Delta_{WC})$-delivery,*
2. *$T \leq (1 - \delta_1) \cdot (\gamma_{WC} - \beta) \cdot c$,*
3. *$\mu \geq 2 \cdot (\log(n) + \kappa) \cdot (\beta \cdot \delta_2^2)^{-1}$,*
4. *$c \geq \frac{\kappa}{\delta_1^2 \cdot (\gamma_{WC} - \beta)}$,*
5. *$\tau \geq \mu \cdot (1 - (1 - \delta_2) \cdot \beta)$,*
6. *and ζ is a (μ, τ)-ECCS,*

then $\mathsf{OptimisticFlood}(\Pi_{BC}, \Pi_{WC}, c, T, \zeta, \alpha)$ has $(\gamma_{WC}, \Delta_{BC} + 4 \cdot \Delta_{CHANNEL} + 2 \cdot \Delta_{WC})$-delivery. Further, if the sender is honest, then the probability that there are less than $\beta \cdot n$ honest parties that have received the message and Step 3b is activated is negligible in κ.

The intuition for β is that it corresponds to a threshold for the fraction of parties that are honest and must have had the message delivered in case the sender does not receive sufficiently many complaints to default back to the worst-case protocol.

Proof. Let s be an honest sender that sends a message m at time t_{INIT}. Further, let $\theta \in [0, \gamma_{\text{ACTUAL}}]$ be the fraction of parties that are honest and have received the message m at time $t_{\text{INIT}} + \Delta_{\text{BC}}$. Because we have no guarantees about how the best-case protocol performs assuming only $\gamma_{\text{ACTUAL}} \geq \gamma_{\text{WC}}$, we have no guarantees about the value of θ. Instead, we make a case distinction whether $\theta > \beta$ or not:

$\theta > \beta$: For this case, we again make a case distinction based upon whether or not the actual number of complaints collected by the sender k is above the threshold T:

- **$k > T$:** In this case the sender will enter Step 3a at time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}}$ and flood the entire message using the worst-case protocol Π_{WC}. Hence, by Condition 1, it is guaranteed that with overwhelming probability, all parties have learned the message at the latest at time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}} + \Delta_{\text{WC}}$.

$k \leq T$: In this case the sender enters Step 3b at time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}}$. Hence, the delivery guarantees of the worst-case flooding protocol Π_{WC} ensures that with overwhelming probability, all parties will have received $(\mathit{PullPhaseBegun}, \mathtt{H}(m))$ before time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}} + \Delta_{\text{WC}}$. Now, this implies that more than $\beta \cdot n$ parties are honest and have input $(\mathit{AcceptPull}, \mathtt{H}(m))$ to $\mathsf{Pull}(\zeta, \alpha)$ by this time. Furthermore, it is guaranteed that all remaining honest parties will have input $(\mathit{Pull}, \mathtt{H}(m))$ before time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}} + 2 \cdot \Delta_{\text{WC}}$. Additionally, Conditions 5 and 6 ensure that the final precondition for Lemma 11 is fulfilled. Hence, using Condition 3, the probability that at time $t_{\text{INIT}} + \Delta_{\text{BC}} + 4 \cdot \Delta_{\text{CHANNEL}} + 2 \cdot \Delta_{\text{WC}}$, there is some party who has not received the message is less than

$$\gamma_{\text{ACTUAL}} \cdot n \cdot e^{-\frac{\delta_2^2 \cdot \mu \cdot \beta}{2}} \leq n \cdot e^{-\frac{\delta_2^2 \cdot \mu \cdot \beta}{2}} \leq n \cdot e^{-\frac{\log(n)+\kappa}{2}} \leq \text{negl}(\kappa). \tag{17}$$

$\theta \leq \beta$: First, note (similarly to the previous case) that if for the actual number of complaints k it holds that $k > T$, then the sender enters Step 3a in which case the probability that all parties have received the message at time $t_{\text{INIT}} + \Delta_{\text{BC}} + 2 \cdot \Delta_{\text{CHANNEL}} + \Delta_{\text{WC}}$ is overwhelming in the security parameter (by Condition 1). Therefore, it is sufficient to show that the probability that $k \leq T$ is negligible in the security parameter. To show this, let $X_1, \ldots, X_c$ be indicator variables s.t. X_i indicates if the committee member i is honest and has not received the message by time $t_{\text{INIT}} + \Delta_{\text{BC}}$. Now, note that any honest party that is part of the committee and has not received the message by time $t_{\text{INIT}} + \Delta_{\text{BC}}$ will send $(\mathit{Complaint}, \mathtt{H}(m))$ at latest at time $t_{\text{INIT}} + \Delta_{\text{BC}} + \Delta_{\text{CHANNEL}}$, which means that the sender will receive it at most Δ_{CHANNEL} time later. Hence, we have that $\sum_{i=1}^{c} X_i \leq k$. By Condition 2, it is therefore to sufficient to show that

$$\Pr\left[\sum_{i=1}^{c} X_i \leq (1-\delta_1) \cdot (\gamma_{\text{WC}} - \beta) \cdot c\right] \leq \text{negl}(\kappa). \tag{18}$$

Now note that for any i

$$\Pr[X_i = 1] = \gamma_{\text{ACTUAL}} - \theta \geq \gamma_{\text{WC}} - \theta \geq \gamma_{\text{WC}} - \beta. \tag{19}$$

Hence, we have $\mathrm{E}\left[\sum_{i=1}^{c} X_i\right] \geq (\gamma_{\text{WC}} - \beta) \cdot c$. Further, because the sampling of the committee is done with replacement, we can apply the Chernoff bound, which when using $c \geq \frac{\kappa}{\delta_1^2 \cdot (\gamma_{\text{WC}} - \beta)}$ (Condition 4) gives us:

$$\Pr\left[\sum_{i=1}^{c} X_i \leq (1-\delta_1) \cdot (\gamma_{\text{WC}} - \beta) \cdot c\right] \leq e^{-\frac{\delta_1^2 \cdot (\gamma_{\text{WC}} - \beta) \cdot c}{2}} \leq e^{-\frac{\kappa}{2}} \leq \text{negl}(\kappa). \tag{20}$$

Hence, the total failure probability will be bounded by the three negligible probabilities from above. Therefore, with overwhelming probability all parties will have received the message before time $t_{\text{INIT}} + \Delta_{\text{BC}} + 4 \cdot \Delta_{\text{CHANNEL}} + 2 \cdot \Delta_{\text{WC}}$. ◀

Finally, combining Lemmas 14 and 15, we can conclude that for suitable parameters the protocol OptimisticFlood performs well in both the best-case and the worst-case.

▶ **Corollary 16.** *Let $\tau, T, c \in \mathbb{N}$ be the size of the committee, let $T \in \mathbb{N}$ be the complaint threshold, let α be a WSCAS, and let $\delta_1, \delta_2, \beta, \delta \in (0, 1]$. If*

1. *Π_{BC} has $(\gamma_{BC}, \Delta_{BC})$-delivery,*
2. *Π_{WC} has $(\gamma_{WC}, \Delta_{WC})$-delivery,*
3. *$T \geq (1+\delta) \cdot (1 - \gamma_{BC}) \cdot c$*

4. $T \leq (1-\delta_1)\cdot(\gamma_{WC}-\beta)\cdot c$,
5. $\mu \geq 2\cdot(\log(n)+\kappa)\cdot(\beta\cdot\delta_2^2)^{-1}$,
6. $c \geq \frac{\kappa}{\delta_1^2\cdot(\gamma_{WC}-\beta)}$
7. $\tau \geq \mu\cdot(1-(1-\delta_2)\cdot\beta)$,
8. *and* ζ *is a* (μ,τ)*-ECCS,*

then $\mathsf{OptimisticFlood}(\Pi_{BC},\Pi_{WC},c,T,\zeta,\alpha)$ *has both* $(\gamma_{BC},\Delta_{BC})$*-delivery* ***and*** $(\gamma_{WC},\Delta_{BC}+4\cdot\Delta_{CHANNEL}+2\cdot\Delta_{WC})$*-delivery. Additionally, when* $\gamma_{ACTUAL}\geq\gamma_{BC}$*, then the probability that Step 3a is activated for an honest sender is less than* $e^{-\frac{\delta^2\cdot(1-\gamma_{BC})\cdot c}{3}}$.

Finally, we state the per-party communication of OptimisticFlood. Due to space constraints we postpone the analysis to the full version [26].

▶ **Theorem 17.** *Let* $\mathsf{OptimisticFlood}(\Pi_{BC},\Pi_{WC},c,T,\zeta,\alpha)$ *be instantiated with variables as stated in Corollary 16 while minimizing the communication complexity, let* α *be a implemented by a merkle tree, and let* ζ *be a* (μ,τ)*-ECCS be implemented with Reed-Solomon codes.*

If $\gamma_{ACTUAL}\geq\gamma_{BC}$*, then*

$$\mathtt{PPCC}(\mathsf{OptimisticFlood}(\Pi_{BC},\Pi_{WC},c,T,\zeta,\alpha),l) \leq \mathtt{PPCC}(\Pi_{BC},l)+\mathtt{PPCC}(\Pi_{WC},\kappa)+\tilde{O}(l+\kappa^2\cdot\gamma_{WC}^{-1}), \quad (21)$$

Further, if $\gamma_{ACTUAL}\geq\gamma_{WC}$*, then*

$$\mathtt{PPCC}(\mathsf{OptimisticFlood}(\Pi_{BC},\Pi_{WC},c,T,\zeta,\alpha),l) \leq \mathtt{PPCC}(\Pi_{BC},l)+\mathtt{PPCC}(\Pi_{WC},l)+\tilde{O}(l\cdot\gamma_{WC}^{-1}+\kappa^2\cdot\gamma_{WC}^{-1}). \quad (22)$$

For the best-case, we emphasize that there is only an asymptotic overhead compared to running only the best-case protocol that is directly linear in the message length for messages of length $l=\tilde{\Omega}(\kappa^2\cdot\gamma_{WC}^{-1})$. Hence, as noticed in Section 1.2, this allows the protocol to shave off a factor of γ_{WC} when instantiated with asymptotically optimal flooding protocols.

Finally, we note that this protocol is also optimistically responsive in the network delay if the best case protocol is optimistically responsive. I.e. if there is a high fraction of honest parties, then OptimisticFlood propagates the message with the actual delivery time of the best-case protocol.

5 Conclusion

In this work, we presented two new protocols for message dissemination based on a push-pull mechanism. Both are asymptotically optimal in terms of per-party communication complexity. The protocol OptimisticFlood has an even better communication complexity in the best-case, where the fraction of honest parties is high. Furthermore, OptimisticFlood is designed modularly such that it remains provably secure when instantiated with a heuristically optimized best-case protocol with high practical efficiency. This improves the state of the art in theoretical research on message dissemination protocols and at the same time provides a protocol with practical efficiency gains.

References

1 Ittai Abraham and Danny Dolev. Byzantine agreement with optimal early stopping, optimal resilience and polynomial complexity. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 605–614. ACM Press, June 2015. doi:10.1145/2746539.2746581.

2 Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: a complete categorization. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *40th ACM PODC*, pages 331–341. ACM, July 2021. doi:10.1145/3465084.3467899.

3 Piotr Berman, Juan A Garay, and Kenneth J Perry. Optimal early stopping in distributed consensus. In *Distributed Algorithms: 6th International Workshop, WDAG'92 Haifa, Israel, November 2–4, 1992 Proceedings 6*, pages 221–237. Springer, 1992.

4 Ran Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020. doi:10.1145/3402457.

5 Brian A Coan. Efficient agreement using fault diagnosis. *Distributed computing*, 7:87–98, 1993. doi:10.1007/BF02280838.

6 Shir Cohen, Idit Keidar, and Alexander Spiegelman. Make Every Word Count: Adaptive Byzantine Agreement with Fewer Words. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems (OPODIS 2022)*, volume 253 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:21, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.OPODIS.2022.18.

7 Sandro Coretti, Aggelos Kiayias, Cristopher Moore, and Alexander Russell. The generals' scuttlebutt: Byzantine-resilient gossip protocols. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 595–608. ACM Press, November 2022. doi:10.1145/3548606.3560638.

8 Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78375-8_3.

9 Giovanni Deligios, Ivana Klasovita, and Chen-Da Liu-Zhang. Optimal early termination for dishonest majority broadcast. Cryptology ePrint Archive, Report 2024/1656, 2024. URL: https://eprint.iacr.org/2024/1656.

10 Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. In Fred B. Schneider, editor, *6th ACM PODC*, pages 1–12. ACM, August 1987. doi:10.1145/41840.41841.

11 Benjamin Doerr and Mahmoud Fouz. Asymptotically optimal randomized rumor spreading. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part II*, volume 6756 of *LNCS*, pages 502–513. Springer, Berlin, Heidelberg, July 2011. doi:10.1007/978-3-642-22012-8_40.

12 Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. 'Eventual' is earlier than 'Immediate'. In *23rd FOCS*, pages 196–203. IEEE Computer Society Press, November 1982. doi:10.1109/SFCS.1982.51.

13 Danny Dolev, Ruediger Reischuk, and H Raymond Strong. Early stopping in byzantine agreement. *Journal of the ACM (JACM)*, 37(4):720–741, 1990. doi:10.1145/96559.96565.

14 Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. doi:10.1137/0212045.

15 Ben Edgington. Upgrading ethereum: 2.9.2 randomness, 2025. URL: https://eth2book.info/capella/part2/building_blocks/randomness/.

16 Muntadher Fadhil, Gareth Owenson, and Mo Adda. A bitcoin model for evaluation of clustering to improve propagation delay in bitcoin network. In *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pages 468–475, 2016. doi:10.1109/CSE-EUC-DCABES.2016.226.

17 Uriel Feige, David Peleg, Prabhakar Raghavan, and Eli Upfal. Randomized broadcast in networks. *Random Structures & Algorithms*, 1(4):447–460, 1990. doi:10.1002/RSA.3240010406.

18 Juan A Garay and Yoram Moses. Fully polynomial byzantine agreement for n> 3 t processors in t+ 1 rounds. *SIAM Journal on Computing*, 27(1):247–290, 1998. doi:10.1137/S0097539794265232.

19 Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Včelák. Verifiable Random Functions (VRFs). RFC 9381, August 2023. doi:10.17487/RFC9381.

20 Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized rumor spreading. In *41st FOCS*, pages 565–574. IEEE Computer Society Press, November 2000. doi:10.1109/SFCS.2000.892324.

21 Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. Parallel Distributed Syst.*, 14(3):248–258, 2003. doi:10.1109/TPDS.2003.1189583.

22 João Leitão, José Pereira, and LuÍs Rodrigues. *Gossip-Based Broadcast*, pages 831–860. Springer US, Boston, MA, 2010. doi:10.1007/978-0-387-09751-0_29.

23 Chen-Da Liu-Zhang, Julian Loss, Ueli Maurer, Tal Moran, and Daniel Tschudi. MPC with synchronous security and asynchronous responsiveness. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 92–119. Springer, Cham, December 2020. doi:10.1007/978-3-030-64840-4_4.

24 Chen-Da Liu-Zhang, Christian Matt, Ueli Maurer, Guilherme Rito, and Søren Eller Thomsen. Practical provably secure flooding for blockchains. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 774–805. Springer, Cham, December 2022. doi:10.1007/978-3-031-22963-3_26.

25 Chen-Da Liu-Zhang, Christian Matt, and Søren Eller Thomsen. Asymptotically optimal message dissemination with applications to blockchains. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part III*, volume 14653 of *LNCS*, pages 64–95. Springer, Cham, May 2024. doi:10.1007/978-3-031-58734-4_3.

26 Chen-Da Liu-Zhang, Christian Matt, and Søren Eller Thomsen. Optimistic message dissemination. Cryptology ePrint Archive, Paper 2025/1404, 2025. URL: https://eprint.iacr.org/2025/1404.

27 Julian Loss and Jesper Buus Nielsen. Early stopping for any number of corruptions. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part III*, volume 14653 of *LNCS*, pages 457–488. Springer, Cham, May 2024. doi:10.1007/978-3-031-58734-4_16.

28 Christian Matt, Jesper Buus Nielsen, and Søren Eller Thomsen. Formalizing delayed adaptive corruptions and the security of flooding networks. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 400–430. Springer, Cham, August 2022. doi:10.1007/978-3-031-15979-4_14.

29 Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238. Springer, New York, August 1990. doi:10.1007/0-387-34805-0_21.

30 Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 3–33. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78375-8_1.

31 Kenneth J Perry and Sam Toueg. An authenticated byzantine generals algorithm with early stopping. Technical report, Cornell University, 1984.

32 I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. doi:10.1137/0108018.

33 Rüdiger Reischuk. A new solution for the byzantine generals problem. *Information and Control*, 64(1-3):23–42, 1985. doi:10.1016/S0019-9958(85)80042-5.

34 Elias Rohrer and Florian Tschorsch. Kadcast: A structured approach to broadcast in blockchain networks. In *AFT*, pages 199–213. ACM, 2019. doi:10.1145/3318041.3355469.

35 Louis Thibault and Dan Marzec. The hitchhiker's guide to p2p overlays in ethereum, 2023. Accessed: 2024-10-18. URL: `https://hackmd.io/@dmarz/ethereum_overlays`.

36 Sam Toueg, Kenneth J Perry, and TK Srikanth. Fast distributed agreement. *SIAM Journal on Computing*, 16(3):445–457, 1987. `doi:10.1137/0216031`.

37 Huy Vu and Hitesh Tewari. An efficient peer-to-peer bitcoin protocol with probabilistic flooding. In Mahdi H. Miraz, Peter S. Excell, Andrew Ware, Safeeullah Soomro, and Maaruf Ali, editors, *Emerging Technologies in Computing*, pages 29–45, Cham, 2019. Springer International Publishing.

Beyond Optimal Fault-Tolerance

Andrew Lewis-Pye
London Schoool of Economics (LSE), UK

Tim Roughgarden
Columbia University, New York, NY, USA
a16z Crypto Research, New York, NY, USA

Abstract

One of the most basic properties of a consensus protocol is its fault-tolerance – the maximum fraction of faulty participants that the protocol can tolerate without losing fundamental guarantees such as safety and liveness. Because of its importance, the optimal fault-tolerance achievable by any protocol has been characterized in a wide range of settings. For example, for state machine replication (SMR) protocols operating in the partially synchronous setting, it is possible to simultaneously guarantee consistency against α-bounded adversaries (i.e., adversaries that control less than an α fraction of the participants) and liveness against β-bounded adversaries if and only if $\alpha + 2\beta \leq 1$.

This paper characterizes to what extent "better-than-optimal" fault-tolerance guarantees are possible for SMR protocols when the standard consistency requirement is relaxed to allow a bounded number r of consistency violations, each potentially leading to the rollback of recently finalized transactions. We prove that bounded rollback is impossible without additional timing assumptions and investigate protocols that tolerate and recover from consistency violations whenever message delays around the time of an attack are bounded by a parameter Δ^* (which may be arbitrarily larger than the parameter Δ that bounds post-GST message delays in the partially synchronous model). Here, a protocol's fault-tolerance can be a non-constant function of r, and we prove, for each r, matching upper and lower bounds on the optimal "recoverable fault-tolerance" achievable by any SMR protocol. For example, for protocols that guarantee liveness against 1/3-bounded adversaries in the partially synchronous setting, a 5/9-bounded adversary can always cause one consistency violation but not two, and a 2/3-bounded adversary can always cause two consistency violations but not three. Our positive results are achieved through a generic "recovery procedure" that can be grafted on to any accountable SMR protocol and restores consistency following a violation while rolling back only transactions that were finalized in the previous $2\Delta^*$ timesteps.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed computing, consensus, recovery

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.15

Related Version *See full version for appendix*: `https://eprint.iacr.org/2025/070`

Funding *Tim Roughgarden*: The research of Tim Roughgarden at Columbia University was supported in part by NSF award CNS-2212745.

1 Introduction

We consider protocols for the state machine replication (SMR) problem, in which processes receive transactions from an environment and are responsible for finalizing a common sequence of transactions. We focus on the partially synchronous setting [14], in which message delays are bounded by a known parameter Δ following an unknown "global stabilization time" GST (and unbounded until that point).

The two most basic requirements of an SMR protocol are consistency, meaning that no two processes should finalize incompatible sequences of transactions (one should be a prefix of the other), and liveness, which stipulates that valid transactions should eventually be finalized (ideally, following GST, within an amount of time proportional to Δ). Guaranteeing

© Andrew Lewis-Pye and Tim Roughgarden;
licensed under Creative Commons License CC-BY 4.0

consistency and liveness becomes impossible if too many of the processes are faulty (i.e., deviate from the intended behavior of a protocol). For the SMR problem in partial synchrony, it is possible to simultaneously guarantee consistency against α-bounded adversaries (i.e., adversaries that control less than an α fraction of the participants) and liveness against β-bounded adversaries if and only if $\alpha + 2\beta \leq 1$ [19, 14].

The focus of this paper is consistency violations – the type of violation that enables, for example, double-spending a cryptocurrency native to a blockchain protocol. What can be said about a protocol when the adversary is large enough to cause a consistency violation? For example, is it already in a position to cause an unbounded number of consistency violations (as opposed to just one), or could the protocol "fight back" in some way?

To make sense of this question and the idea of multiple consistency violations, we must formalize a sense in which a protocol might restore consistency following a violation, necessarily by rolling back transactions that had been viewed as finalized by some non-faulty processes. One key parameter is then the *recovery time* d, meaning the number of timesteps after a violation before a protocol returns to healthy operation. A second is the *rollback* h, meaning that the recovery process "unfinalizes" only transactions that have been finalized within the previous h time steps.

The natural wishlist for an SMR protocol in partial synchrony would then be:

(1) All of the "usual" guarantees, such as optimal fault-tolerance (i.e., consistency with respect to α-bounded adversaries and liveness with respect to β-bounded adversaries for some $\alpha, \beta > 0$ with $\alpha + 2\beta = 1$).

(2) Automatic recovery from a consistency violation with the worst-case recovery time d and worst-case rollback h as small as possible (if nothing else, independent of the specific execution).

(3) Never suffers more than r consistency violations overall, where r is as small as possible.

To what extent are these properties simultaneously achievable?

This paper provides a thorough investigation of this question. To expose the richness of the answer, we work with a timing model that can be viewed as an interpolation between the synchronous and partially synchronous settings. In addition to the usual parameters Δ and GST (known and unknown, respectively) of the partially synchronous model, we allow for a known parameter $\Delta^* \geq \Delta$ which may or may not bound message delays prior to GST. Canonically, Δ^* should be thought of as orders of magnitude larger than Δ, with Δ indicating the speed of communication between processes when all is well (no network issues, no attacks) and Δ^* a safe (and possibly large) upper-bound on the speed of (possibly out-of-band) communication around the time of an attack.[1] We will be interested in protocols that always satisfy all the "usual" guarantees (1) and that finalize transactions in time $O(\Delta)$ (rather than $O(\Delta^*)$) after GST, whether or not pre-GST message delays are bounded by Δ^*, and also satisfy the additional recovery guarantees (2) and (3) in the event that pre-GST message delays are in fact bounded by Δ^*.[2]

Our main positive result, stated formally in Theorem 1 and proved in Section 7, shows that such protocols do indeed exist. For example, we show that there is a protocol that satisfies:

[1] Indeed, for our positive results, message delays must be bounded by Δ^* for the duration of our recovery procedure, but not otherwise.

[2] In particular, a synchronous protocol with respect to the parameter Δ^* will not generally satisfy consistency and liveness if message delays do not happen to bounded above by Δ^*.

- $\frac{1}{3}$-resilience in partial synchrony (independent of whether Δ^* bounds pre-GST message delays), with expected latency $O(\Delta)$ after GST;
- should pre-GST message delays be bounded by Δ^*, recovers from consistency violations in expected time $O(\Delta^*)$ and with rollback $2\Delta^*$; and
- should pre-GST message delays be bounded by Δ^*, never suffers from more than one consistency violation with a $\frac{5}{9}$-bounded attacker, and never suffers from more than two consistency violations with a $\frac{2}{3}$-bounded attacker.

We achieve this result by designing a generic "recovery procedure" that can be grafted on to any accountable SMR protocol, including protocols with asymmetric fault-tolerance with respect to consistency and liveness attacks. Sections 4 and 5 give informal and formal, respectively, descriptions of this procedure.

Our results are tight in several senses. For example, we prove in Theorem 2 that recovery from a consistency violation necessarily requires a rollback proportional to the parameter Δ^*. In particular, in the pure partially synchronous model ($\Delta^* = +\infty$, in effect), recovery from consistency violations with bounded rollback is impossible. Theorems 3 and 4 show that the bounds we obtain on adversary size (as a function of the number r of consistency violations) are optimal. For example, in the symmetric case above, an attacker controlling five-ninths of the processes can always force two consistency violations, and one controlling two-thirds of the processes can cause unbounded rollback.

A comment on the benefits of automated recovery. It is sometimes assumed in the literature (e.g. [26]) that, in the event of a consistency violation, an "administrator" will somehow (through out-of-protocol means) remove perpetrators from the system and co-ordinate an appropriate "reboot". Our automated recovery procedure is "in-protocol," and therefore has the significant benefits that it does not rely on a centralized entity, has no single point of failure, and formalizes a process by which one can guarantee bounded rollback (a principal focus of this paper).

A comment on the (Δ, Δ^*) timing assumptions. In practical settings, partial synchrony may hold with respect to Δ of the order of a few hundred milliseconds. It may also be reasonable to suppose that synchrony will hold with respect to some much larger bound Δ^*, but making direct use of this larger bound to run a synchronous protocol (and increase resilience beyond 1/3) would result in an impractically slow protocol. Our use of the two bounds Δ and Δ^* reflects our interest in considering protocols that give *all* the usual guarantees of protocols for partial synchrony (with no requirement that the larger bound Δ^* should hold), but which *also* give recovery guarantees in the case that the larger bound for synchrony should hold.

2 The setup

We consider a set $\Pi = \{p_1, \ldots, p_n\}$ of n processes. Each process p_i is told its "name" i as part of its input. We focus on the case of a static adversary, which chooses a set of processes to corrupt at the start of the protocol execution.[3] We call a process corrupted by the adversary

[3] All results in this paper hold more generally for adaptive adversaries (with essentially identical proofs), with the exception of the bound on the expected termination time for the recovery procedure asserted in part (iii) of Theorem 1 (which requires that a random permutation of the processes be chosen subsequent to the adversary deciding which processes to corrupt).

faulty. Faulty processes may behave arbitrarily (i.e., we consider *Byzantine* faults), subject to our cryptographic assumptions (stated below). Processes that are not faulty are *correct*. The adversary is *ρ-bounded* if it corrupts less than a ρ fraction of the n processes.

Cryptographic assumptions. We assume that processes communicate by point-to-point authenticated channels and that a public key infrastructure (PKI) is available for generating and validating signatures. For simplicity of presentation (e.g., to avoid the analysis of negligible error probabilities), we work with ideal versions of these primitives (i.e., we assume that faulty processes cannot forge signatures). We also assume that all processes have access to a random permutation of Π, denoted $\Pi^* : [1, n] \to \Pi$, which is sampled after the adversary chooses which processes to corrupt.

Message delays. We consider a discrete sequence of timeslots $t \in \mathbb{N}_{\geq 0}$. As discussed in the introduction, we consider protocols that operate in partial synchrony (with some parameter Δ, perhaps in the order of seconds or milliseconds) and satisfy additional recovery properties should synchrony hold (with some different parameter Δ^*, which may be set much larger than Δ) while running the "recovery procedure".

Synchrony. In the synchronous setting, a message sent at time t must arrive by time $t + \Delta^*$, where Δ^* is known to the protocol.

Partial synchrony. In the partially synchronous setting, a message sent at time t must arrive by time $\max\{\text{GST}, t\} + \Delta$. While Δ is known, the value of GST is unknown to the protocol. The adversary chooses GST and also message delivery times, subject to the constraints already specified.

Clock synchronization. In the partially synchronous setting, we suppose all correct processes begin the protocol execution before GST. When considering the synchronous setting, we suppose all correct processes begin the protocol execution by time Δ^*. A correct process begins the protocol execution with its local clock set to 0; thus, we do not suppose that the clocks of correct processes are synchronized. For simplicity, we assume that the clocks of correct processes all proceed in real time, meaning that if $t' > t$ then the local clock of correct p at time t' is $t' - t$ in advance of its value at time t.[4]

Notation concerning executions and received messages. We use the following notation when discussing any execution of a protocol:

- $M_i(t)$ denotes the set of messages received by process p_i by timeslot t;
- $M_c(t)$ denotes the set of all messages received by any correct process by timeslot t;
- M_c denotes the set of all messages received by any correct process during the execution.

Transactions. Transactions are messages of a distinguished form, signed by the *environment*. Each timeslot, each process may receive some finite set of transactions directly from the environment.

Determined inputs. A value is *determined* if it known to all processes, and is otherwise *undetermined*. For example, Δ, Δ^* and Π are determined, while GST is undetermined.

[4] Using standard arguments, our protocol and analysis can easily be extended to the case in which there is a known upper bound on the difference between the clock speeds of correct processes.

State machine replication. Informally, a protocol for state machine replication (SMR) must cause correct processes to finalize *logs* (sequences of transactions) that are live and consistent with each other, and must also produce "certificates" of finalization that can be presented to clients who may not always be online and observing the protocol execution. Formally, if σ and τ are sequences, we write $\sigma \preceq \tau$ to denote that σ is a prefix of τ. We say σ and τ are *compatible* if $\sigma \preceq \tau$ or $\tau \preceq \sigma$. If two sequences are not compatible, they are *incompatible*. If σ is a sequence of transactions, we write $\mathtt{tr} \in \sigma$ to denote that the transaction $\mathtt{tr}$ belongs to the sequence σ.

Fix a process set Π and *genesis log*, denoted $\mathtt{log}_G$. If $\mathcal{P}$ is a protocol for SMR, then it must specify a function $\mathcal{F}$, which may depend on Π and $\mathtt{log}_G$, that maps any set of messages to a sequence of transactions extending $\mathtt{log}_G$. We require the following conditions to hold in every execution (for any M_1, M_2, p_i, p_j and $\mathtt{tr}$):

Consistency. If $M_1 \subseteq M_2 \subseteq M_c$, then $\mathcal{F}(M_1) \preceq \mathcal{F}(M_2)$.[5]

Liveness. If p_i and p_j are correct and if p_i receives the transaction $\mathtt{tr}$ then, for some t, $\mathtt{tr} \in \mathcal{F}(M_j(t))$.

This definition of consistency ensures that correct processes never finalize incompatible logs: for any sets $M_1, M_2 \subseteq M_c$ that two such processes might have received, $\mathcal{F}(M_1) \preceq \mathcal{F}(M_1 \cup M_2)$ and $\mathcal{F}(M_2) \preceq \mathcal{F}(M_1 \cup M_2)$. We say a set of messages M is a *certificate* for a sequence of transactions σ if $\mathcal{F}(M) \succeq \sigma$: intuitively, any process can present the set of (potentially signed) messages M to a "client" that has not been observing the protocol execution as proof that it has finalized a sequence of transactions extending σ. If we wish to make $\mathcal{F}$ explicit, we may also say that M is an $\mathcal{F}$-certificate for σ.

SMR (informal discussion). The selection $\mathcal{F}$ of finalized transactions by a correct process depends only on the set of messages it has received, and not on the times at which these messages were received. The motivation for this restriction is to formalize the requirement of SMR [22] that "clients" wishing to verify the finality of transactions without observing the entire execution of the protocol should be able to do so (via a suitable certificate). As discussed in [25], this crucial distinction between SMR and Total-Order-Broadcast is sometimes overlooked in the literature: In the context of an honest majority, any report of finality by a majority of processes constitutes proof of finality, so a protocol for Total-Order-Broadcast directly gives a protocol for SMR. However, the distinction becomes non-trivial in the context of player reconfiguration, or if there is no guarantee of honest majority (such as in this paper). In partial synchrony, certificates are anyways required for guaranteed consistency and liveness [16], so our approach is general for the context considered here.

The liveness parameter. If there exists some fixed ℓ that is a function of determined inputs[6] other than Δ^* and such that the following holds in all executions of $\mathcal{P}$, we say $\mathcal{P}$ has *liveness parameter* ℓ: If p_i and p_j are correct and if p_i receives the transaction $\mathtt{tr}$ at time t then, for $t' = \max\{t, \mathrm{GST}\} + \ell$, $\mathtt{tr} \in \mathcal{F}(M_j(t'))$.

[5] This is equivalent to the seemingly stronger condition in which M_c is replaced by the set of messages received by any process (correct or otherwise), as faulty processes always have the option of echoing any messages they receive to correct processes.

[6] The requirement that ℓ is not a function of Δ^* (while Δ^* is not necessarily $O(\Delta)$) means that having liveness parameter ℓ may require finalization of transactions in time less than Δ^* after GST.

Liveness and consistency resilience. Recall that $n = |\Pi|$. When the protocol $\mathcal{P}$ is clear from context, we write ρ_C to denote the consistency resilience of $\mathcal{P}$, which is the largest ρ such that, for all n, the protocol satisfies consistency so long as the adversary is ρ-bounded. We write ρ_L to denote the liveness resilience, which is the largest ρ such that, for all n, the protocol satisfies liveness so long as the adversary is ρ-bounded. It is well known that $\rho_C + 2\rho_L \leq 1$ in the partially synchronous setting [14] and that $\rho_C + \rho_L \leq 1$ in the synchronous setting [19].

The number of consistency violations. When $\mathcal{F}$ is clear from context, we say the set of messages M *has* r *consistency violations* if there exist $M_0 \subset M_1 \subset \cdots \subset M_r \subseteq M$ such that, for each $s \in \{0, 1, \ldots, r-1\}$, $\mathcal{F}(M_s) \not\preceq \mathcal{F}(M_{s+1})$. We also say M *has a consistency violation* (w.r.t. $\mathcal{F}$) if it has at least one consistency violation. An execution has r consistency violations if M_c has r consistency violations.

Accountable protocols (informal discussion). Informally, a protocol is *accountable* if it produces *proofs of guilt* for some faulty processes in the event of a consistency violation. We cannot generally require proofs of guilt for a fraction $\lambda > \rho_C$ of processes, since consistency violations may occur when less than a fraction λ of processes are faulty. On the other hand, all standard protocols that provide accountability produce proofs of guilt for a ρ_C fraction of processes in the event of a consistency violation [24].

Accountable protocols (formal definition). Consider an SMR protocol $\mathcal{P}$:

- We say the set of messages M is a *proof of guilt* for $p \in \Pi$ if there does not exist any execution of $\mathcal{P}$ in which p is correct and for which $M \subseteq M_c$.[7]
- We say $\mathcal{P}$ is λ-*accountable* if the following holds at every timeslot t of any execution of $\mathcal{P}$: if $M_c(t)$ has a consistency violation, then $M_c(t)$ is a proof of guilt for at least a λ fraction of processes in Π.

Given that all standard protocols that are λ-accountable for any $\lambda > 0$ are also ρ_C-accountable, we will say that a protocol is *accountable* to mean that it is ρ_C-accountable. It is important to note that, while an accountable protocol ensures the existence of proofs of guilt for a ρ_C fraction of processes in the event of a consistency violation, it *does not* automatically ensure *consensus* between correct processes as to a set of faulty processes for which a proof of guilt exists. One role of the *recovery* procedure (as specified in Section 5) will be to ensure such consensus.

Message gossiping. In our recovery procedure, it will be convenient to assume that correct processes gossip all messages received. Then, if synchrony does hold with respect to Δ^*, any message received by correct p at some timeslot t is received by all correct processes by time $t + \Delta^*$. It will not generally be necessary to gossip all messages; for example, for standard quorum-based protocols, it will suffice to gossip blocks that have received quorum certificates (QCs) along with those QCs.

A comment on setup assumptions. Given an accountable SMR protocol $\mathcal{P}$ and a process set Π, our wrapper will initiate a sequence of executions of $\mathcal{P}$, with process sets that are progressively smaller subsets of Π. Of course, a PKI for Π suffices to provide a PKI for each

[7] If we wish to make $\mathcal{P}$, $\mathtt{log}_G$, and Π explicit, we may also say that M is a $(\mathcal{P}, \Pi, \mathtt{log}_G)$-proof of guilt.

subset of Π and a random permutation of Π naturally induces a random permutation of each subset. Moreover, the maximum number of executions of $\mathcal{P}$ initiated by the wrapper will be small and the size of the process set of each is known ahead of time. (For example, if $\rho_C = 1/3$ and the adversary is 5/9-bounded, the wrapper will initiate at most two executions of $\mathcal{P}$; if $\rho_C = 1/3$ and the adversary is 2/3-bounded, at most three.) Thus, for setup assumptions such as threshold signatures, one can simply run each required setup in advance, before executing the wrapper.

3 Recovery metrics

In this section, we introduce definitions to quantify how well a protocol recovers from consistency violations.

Generalizing resilience to take recovery into account. Is a protocol vulnerable to one consistency violation inexorably doomed to an unbounded number of them? Or could a protocol achieve strictly higher levels of resilience by tolerating (and recovering from) a bounded number of consistency violations? The following definitions generalize consistency and liveness resilience to account for the possibility of recovery from consistency violations.

- *Recoverable consistency resilience.* Consider a function $g : \mathbb{N}_{\geq 0} \to [0, 1]$. We say a protocol $\mathcal{P}$ has recoverable consistency resilience g if the following holds for each $r \in \mathbb{N}_{\geq 0}$: $g(r)$ is the largest ρ such that, for all n, provided the adversary is ρ-bounded, executions of $\mathcal{P}$ have at most r consistency violations.
- *Recoverable liveness resilience.* Consider a function $g : \mathbb{N}_{\geq 0} \to [0, 1]$. We say a protocol $\mathcal{P}$ has recoverable liveness resilience g if the following holds for each $r \in \mathbb{N}_{\geq 0}$: $g(r)$ is the largest ρ such that, for all n, provided the adversary is ρ-bounded, liveness holds in all executions with precisely r consistency violations.[8]

Suppose $\mathcal{P}$ has consistency resilience ρ_C and recoverable consistency resilience g. Note that $g(0) = \rho_C$. Also, g is nondecreasing (i.e., $g(s) \geq g(r)$ for $s > r$): if executions of $\mathcal{P}$ have at most r consistency violations when the adversary is ρ-bounded, then this is also true of all $s > r$. If $g(r + 1) > g(r)$, the protocol effectively has increased consistency resilience after r consistency violations.

Recoverable resilience for our wrapper. Suppose $\mathcal{P}$ is accountable and has consistency resilience ρ_C and liveness resilience ρ_L for partial synchrony with $\rho_C + 2\rho_L = 1$. If we identify some fraction x of the processes in Π as faulty and then run an execution of $\mathcal{P}$ using the remaining processes, there will be no consistency violation so long as less than a fraction $x + \rho_C(1 - x)$ of the processes in Π are faulty. Given this, let us define a sequence $\{x_r\}_{r \in \mathbb{N}_{\geq 0}}$ by recursion:

$$x_0 = 0, \quad x_{r+1} = x_r + \rho_C(1 - x_r).$$

Define:

$$g_1(r) = \min\{x_{r+1}, 1 - \rho_L\}, \quad g_2(r) = \min\{x_r + \rho_L(1 - x_r), 1 - \rho_L\}. \tag{1}$$

[8] Prior to the rth consistency violation, a sufficiently large adversary may still be in a position to cause a liveness violation.

Given $\mathcal{P}$ as input, our wrapper produces an SMR protocol with recoverable consistency resilience g_1 and recoverable liveness resilience g_2 as in (1). For example, if $\rho_C = \rho_L = \frac{1}{3}$, then $g_1(0) = g_2(0) = \frac{1}{3}$, $g_1(1) = g_2(1) = \frac{5}{9}$, and $g_1(r) = g_2(r) = \frac{2}{3}$ for all $r \geq 2$.

Specifying the recovery time. Next, we provide a definition that captures the time required by a protocol to recover from consistency violations. Suppose $\mathcal{P}$ has recoverable liveness resilience g. We say $\mathcal{P}$ has *recovery time d with liveness parameter ℓ* if the following holds for all executions $\mathcal{E}$ of $\mathcal{P}$:

($\dagger_{d,\ell}$) If there exists r such that $\mathcal{E}$ has precisely r consistency violations, let t be least such that $M_c(t)$ has r consistency violations (otherwise set $t = \infty$). If correct p_i receives the transaction $\mathtt{tr}$ at any timeslot t' then, for every correct p_j and for $t'' = \max\{t + d, \mathrm{GST}, t'\} + \ell$, $\mathtt{tr} \in \mathtt{log}_j(t'')$.

In the above, d should be thought of as a "grace period" after consistency violations, after which liveness with parameter ℓ must hold. In our construction, d is governed by the length of time it takes to run our recovery procedure.

Probabilistic recovery time. Our recovery procedure uses the random permutation Π^* – chosen after the adversary chooses which processes to corrupt – to select "leaders," and as such it will run for a random duration. To analyze this, we allow the grace period parameter d in the definition above to depend on an error probability $\varepsilon \in [0, 1]$ and sometimes write d_ε to emphasize this dependence. We then make the following definitions:

- We say that ($\dagger_{d,\ell}$) *is ensured with probability at least p* if, for every choice of corrupted processes (consistent with a static ρ-bounded adversary), with probability at least p over the choice of Π^* (sampled from the uniform distribution), ($\dagger_{d,\ell}$) holds in every execution consistent with these choices and with the setting.
- We say that $\mathcal{P}$ has *probabilisitic recovery time d_ε with liveness parameter ℓ* if it holds for every $\varepsilon \in [0, 1]$ that ($\dagger_{d_\varepsilon,\ell}$) is ensured with probability at least $1 - \varepsilon$.

Recovery time for our wrapper. Given $\mathcal{P}$ with liveness parameter ℓ as input, our wrapper will produce an SMR protocol with (worst-case) recovery time $O(\Delta^* \cdot f_a)$, probabilisitic recovery time $O(\Delta^* \cdot \log \frac{1}{\varepsilon})$, and liveness parameter ℓ, where f_a denotes the actual (undetermined) number of faulty processes.

Bounding rollback. We say that a protocol has *rollback bounded by h* if the following holds for every execution consistent with the setting and every correct $p_i, p_j \in \Pi$: if there exists an interval $I = [t, t + h]$ such that $\sigma \preceq \mathtt{log}_i(t')$ for all $t' \in I$, then $\sigma \preceq \mathtt{log}_j(t')$ for all sufficiently large t'. That is, consistency violations can "unfinalize" only transactions that have been finalized recently, within the previous h time steps. Here, h can be any value that depends only on determined inputs.

Bounding rollback for our wrapper. Given an SMR protocol $\mathcal{P}$ with liveness resilience ρ_L as input, our wrapper will produce an SMR protocol with rollback bounded by $h = 2\Delta^*$ so long as synchrony holds for Δ^* and the adversary is $(1 - \rho_L)$-bounded. In fact, while the recovery procedure described in Section 5 requires a common choice for Δ^*, rollback can be bounded on an individual basis, with each correct process making their own personal choice of message delay bound $\leq \Delta^*$. rollback will be bounded by twice their personal choice of bound, so long as that bound on message delay holds.[9]

[9] The requirement that the choice be $\leq \Delta^*$ stems from the fact that the recovery procedure requires delays to be bounded by Δ^* to function correctly.

4 The intuition behind the wrapper

We describe a wrapper, which takes an accountable and optimally resilient SMR protocol $\mathcal{P}$ as input, and which runs an execution of $\mathcal{P}$ until a consistency violation occurs.[10] Once this happens, the wrapper triggers a "recovery procedure", which achieves consensus on a set of faulty processes F for which a proof of guilt exists, together with a long initial segment of the log produced by $\mathcal{P}$ below which no consistency violation has occurred. The wrapper then initiates another execution of $\mathcal{P}$ that takes this log as its genesis log, with the players in F removed from the process set. This next execution is run until another consistency violation occurs, and so on.

Specifying $\mathtt{log}_i(t)$ and $\mathcal{F}$. While the formal definition of SMR in Section 2 requires us to specify the finalization rule $\mathcal{F}$ (from which the transactions $\mathtt{log}_i(t)$ finalized by p_i can then be defined as $\mathcal{F}(M_i(t))$), it will be more natural when defining our wrapper to specify $\mathtt{log}_i(t)$ directly, and then later to define $\mathcal{F}$ such that $\mathtt{log}_i(t) = \mathcal{F}(M_i(t))$. Recall that the given protocol $\mathcal{P}$ satisfies consistency and liveness with respect to a function that may depend on the process set and the genesis log. We write $\mathcal{F}(\Pi, \mathtt{log}_G)$ to denote this function.

The structure of this section. In Section 4.1, we describe the intuition behind a feature of the wrapper which allows us to ensure rollback bounded by $2\Delta^*$. We stress that *bounding rollback is non-trivial*: this is the requirement on the recovery procedure that requires the most delicate analysis. Section 4.2 then describes the intuition behind the recovery procedure.

In what follows, we use the variable $\mathcal{E}$ to denote an execution of the wrapper (with process set Π and $\mathtt{log}_G$ as the genesis log), which initiates successive executions $\mathcal{E}_1, \mathcal{E}_2, \ldots$ of $\mathcal{P}$, where $\mathcal{E}_r$ has process set Π_r and $\mathtt{log}_{G_r}$ as the genesis log. Process p_i maintains local variables $\mathtt{M}_i$ and $\mathtt{M}_{i,r}$ for each $r \geq 1$.[11] The former records all messages so far received in execution $\mathcal{E}$, while the latter records all messages so far received in execution $\mathcal{E}_r$. We suppose messages have tags identifying the execution in which they are sent, and that $\mathtt{M}_{i,r} \subseteq \mathtt{M}_i$ at every timeslot, for all correct p_i and all r.

4.1 Ensuring bounded rollback

In what follows, we write ρ_C and ρ_L to denote the consistency and liveness resilience of $\mathcal{P}$. Each process p_i executing the wrapper maintains a value $\mathtt{log}_i$. Suppose the currently running execution of $\mathcal{P}$ is $\mathcal{E}_r$. To ensure rollback bounded by $2\Delta^*$, p_i proceeds as follows:

- While running the execution $\mathcal{E}_r$ of $\mathcal{P}$, and when p_i finds that some subset of $\mathtt{M}_{i,r}$ is an $\mathcal{F}(\Pi_r, \mathtt{log}_{G_r})$-certificate for σ properly extending $\mathtt{log}_i$, it will set $\mathtt{log}_i$ to extend σ.
- Process p_i will only *strongly finalize* σ, however, once $\mathtt{log}_i$ has extended σ for an interval of length $2\Delta^*$.
- Upon finding that $\mathtt{M}_{i,r}$ has a consistency violation w.r.t. $\mathcal{F}(\Pi_r, \mathtt{log}_{G_r})$, p_i will:
  - Send a signed *r-genesis* message $(\text{gen}, \mathtt{log}_i, r)$ to all processes (motivation below);
  - Temporarily set $\mathtt{log}_i$ to be $\mathtt{log}_{G_r}$;
  - Stop running $\mathcal{E}_r$, and;
  - Begin the recovery procedure.

[10] By "optimally resilient," we mean that the protocol's consistency resilience ρ_C and liveness resilience λ_L in partial synchrony are both positive and satisfy $\rho_C + 2\rho_L = 1$ (as is the case for all of the "usual" SMR protocols designed for the partially synchronous setting). This assumption is merely to simplify the presentation. For a non-optimally resilient protocol, the "$1 - \rho_L$" term in (1) should be replaced by "$(1 + \rho_C)/2$".

[11] We use $\mathtt{M}_i$ when specifying the pseudocode, rather than $M_i(t)$, since p_i only has access to its local clock and does not know the "global" value of t.

To see what this achieves (modulo complications that may later be introduced by the recovery procedure), suppose that synchrony holds for Δ^*. Then, due to our assumptions on message gossiping described in Section 2, $2\Delta^*$ bounds the round-trip time between any two correct processes. In particular, suppose that p_i finalizes σ at t because there exists $M \subseteq \mathtt{M}_{i,r}$ which is an $\mathcal{F}(\Pi_r, \mathtt{log}_{G_r})$-certificate for σ. Then every correct process p_j will receive the messages in M by $t + \Delta^*$, and will then finalize σ (never to subsequently finalize anything incompatible with σ), unless $\mathtt{M}_{j,r}$ has a consistency violation (w.r.t. $\mathcal{F}(\Pi_r, \mathtt{log}_{G_r})$) by that time. In the latter case, p_i will begin the recovery procedure by timeslot $t + 2\Delta^*$ and will not strongly finalize σ. So far, this approach is similar to the "safety-favoring" construction of [25].

Complications introduced by the recovery procedure. Our recovery procedure introduces the complication that there is not necessarily consensus on which logs have been strongly finalized by some correct process. If a single correct process has strongly finalized σ when the recovery procedure is triggered, and if the procedure determines that a log $\sigma' \not\succeq \sigma$ should be used as the genesis log in the next execution of $\mathcal{P}$, then this may violate the condition that the protocol has rollback bounded by $2\Delta^*$. We must therefore ensure that the recovery procedure reaches consensus on a log that extends all logs strongly finalized by correct processes. As in explained in Section 4.2, the r-genesis messages sent by processes before entering the recovery procedure will be used to achieve this.

4.2 The intuition behind the recovery procedure

Recall that ρ_C (ρ_L) is the consistency (liveness) resilience of $\mathcal{P}$ in partial synchrony (with parameter Δ), and that the wrapper aims to deliver extra functionality in the case that synchrony happens to hold with respect to the (possibly large) bound Δ^*, and so long as the adversary is $(1 - \rho_L)$-bounded. So, suppose these conditions hold.

As noted in Section 4.1, while running execution $\mathcal{E}_r$ of $\mathcal{P}$, process p_i will enter the recovery procedure upon finding that $\mathtt{M}_{i,r}$ has a consistency violation. Given our gossip assumption, described in Section 2, this means that correct processes will begin the recovery procedure within time Δ^* of each other. The key observation behind the recovery procedure is that, if one has a proof of guilt for processes in some set F, where $|F| \geq \rho_C n$, then the fact that the adversary is $(1 - \rho_L)$-bounded (and $2\rho_L + \rho_C = 1$) means that the adversary controls less than half the processes in $\Pi - F$. This follows since:

$$1 - \rho_L - \rho_C = \rho_L, \text{ and so } (1 - \rho_L - \rho_C)/(1 - \rho_C) = \rho_L/2\rho_L.$$

As a consequence, we can run a modified version of a standard ($\frac{1}{2}$-resilient) SMR protocol for synchrony (our protocol is most similar to [1]), in which the instructions are divided into views, each with a distinct leader. In each view, the leader makes a proposal for the set of processes F that should be removed from Π_r to form Π_{r+1}, and *the processes outside F then vote on that proposal.*

Ensuring an appropriate value for $\mathtt{log}_{G_{r+1}}$. As well as proposing F, the leader p_i must also suggest a sequence σ to be used as $\mathtt{log}_{G_{r+1}}$ and this sequence must extend all logs strongly finalized by correct processes. To achieve this (while keeping the probabilistic recovery time small), we run a short sub-procedure at the beginning of the recovery procedure, before leaders start proposing values. We proceed as follows:

- Each correct p_j waits time $2\Delta^*$ upon beginning the recovery procedure and then sets $P_j(r)$ to be the set of processes in Π_r from which it has received an r-genesis message.
- Process p_j then enters view $(r, 1)$ (the 1^{st} view of the r^{th} execution of the recovery procedure).

To form an appropriate proposal σ for $\mathtt{log}_{G_{r+1}}$ while in view (r, v), the leader p_i of the view waits for $2\Delta^*$ after entering the view (to accommodate possible lags between the progress of and information received by different correct processes), and then proceeds as follows. If M is the set of r-genesis messages that p_i has received by that time and which are signed by processes in $\Pi_r - F$, then let M' be a maximal subset of M that contains at most one message signed by each process. We say σ is extended by the r-genesis message (gen, σ', r) if $\sigma \preceq \sigma'$. Process p_i then sets σ so that the following condition is satisfied:

$\dagger(M', \sigma)$: σ is the longest sequence extended by more than $\frac{1}{2}|\Pi_r - F|$ elements of M'.

Process p_i then sends M' along with σ as a *justification* for its proposal. A correct process p_j will be prepared to vote on the proposal if $\dagger(M', \sigma)$ is satisfied and M' includes messages from every member of $P_j(r)$.

To see that this achieves the desired outcome, note that if p_j is correct and M' includes messages from every member of $P_j(r)$, then it must contain a message from every correct process. If σ' has been strongly finalized by some correct process, then every correct process must have finalized σ' before entering the recovery procedure, and cannot have subsequently finalized any value incompatible with σ'. So, for each r-genesis message $(\text{gen}, \sigma'', r)$ sent by a correct process, σ'' must extend σ'. It therefore holds that σ' is extended by more than $\frac{1}{2}|\Pi_r - F|$ elements of M', so that, if $\dagger(M', \sigma)$ is satisfied, σ must extend σ'.

5 The formal specification of the wrapper

In what follows, we suppose that, when a correct process sends a message to "all processes", it regards that message as immediately received by itself. The pseudocode uses a number of inputs, local variables, functions and procedures, detailed below.

Inputs. The wrapper takes as input an SMR protocol $\mathcal{P}$, a process set Π, a random permutation Π^* of Π, a value $\mathtt{log}_G$, and message delay bounds Δ^* and Δ. The consistency resilience ρ_C of $\mathcal{P}$ is also given as input. Recall that the given protocol $\mathcal{P}$ satisfies consistency and liveness with respect to a finalization function that may depend on the process set Π' and the value $\mathtt{log}'_G$ for the genesis log. (For example, signatures from a certain fraction of the processes in Π' may be required for transaction finalization.) We write $\mathcal{F}(\Pi', \mathtt{log}'_G)$ to denote this function, and suppose also that this function is known to the protocol.

Permutations and the variables Π_r. Process p_i maintains a variable Π_r for each $r \in \mathbb{N}_{\geq 1}$. Π_1 is initially set to Π, while each Π_r for $r > 1$ is initially undefined.[12] Once Π_r is defined, Π^*_r is the permutation of Π_r induced by Π^*.

Views and leaders. Views are indexed by ordered pairs and ordered lexicographically: one should think of view (r, v) as the v^{th} view in the r^{th} execution of the recovery procedure. For $r, v \in \mathbb{N}_{\geq 1}$, we set $\mathtt{lead}(r, v) = p_i$, where $p_i = \Pi^*_r(v)$; this function is used to specify the leader of each view.[13]

[12] We write $x \uparrow$ to indicate that the variable x is undefined, and $x \downarrow$ to indicate that x is defined.

[13] We can write $p_i = \Pi^*_r(v)$ because the number of views in the r^{th} execution of the recovery procedure will be bounded by $|\Pi_r|$.

Received messages and executions. We let $\mathtt{M}_i$ be a local variable that specifies the set of all messages so far received by p_i. The wrapper will also initiate executions $\mathcal{E}_1, \mathcal{E}_2, \ldots$ of $\mathcal{P}$: for each $r \geq 1$, $\mathtt{M}_{i,r}$ specifies all messages so far received by p_i in execution $\mathcal{E}_r$. We suppose that messages have tags identifying the execution in which they are sent, and that all messages received by p_i in $\mathcal{E}_r$ are also received by p_i in the present execution of the wrapper, so that $\mathtt{M}_{i,r} \subseteq \mathtt{M}_i$ for all r.

The variables $\mathtt{log}_{G_r}$. Process p_i maintains a variable $\mathtt{log}_{G_r}$ for each $r \in \mathbb{N}_{\geq 1}$. Initially, $\mathtt{log}_{G_1}$ is set to $\mathtt{log}_G$, while each $\mathtt{log}_{G_r}$ for $r > 1$ is undefined. If the execution $\mathcal{E}_r$ of $\mathcal{P}$ is initiated by the wrapper, then this will be an execution with $\mathtt{log}_{G_r}$ as the genesis log and with process set Π_r.

Logs. Process p_i maintains two variables $\mathtt{log}_i$ and $\mathtt{log}_i^*$. The former should be thought of as the sequence of transactions that p_i has finalized, while the latter is the sequence that p_i has strongly finalized.

Signatures. We write m_{p_i} to denote the message m signed by p_i.

***r*-*genesis* messages.** An *r-genesis* message is a message of the form $(\text{gen}, \sigma, r)_{p_j}$, where σ is a sequence of transactions and $p_j \in \Pi$. These are used during the r^{th} execution of the recovery procedure to help reach consensus on an appropriate value for $\mathtt{log}_{G_{r+1}}$. We say σ' is extended by the r-genesis message $(\text{gen}, \sigma, r)_{p_j}$ if $\sigma' \preceq \sigma$.

***r*-proposals.** An *r-proposal* is a tuple $P = (F, \sigma, M, r)$, where $F \subset \Pi$, σ is a sequence of transactions, M is a set of r-genesis messages, and $r \in \mathbb{N}_{\geq 1}$. The last entry r indicates that this is a proposal corresponding to the r^{th} execution of the recovery procedure. One should think of F as a suggestion for $\Pi_r - \Pi_{r+1}$, while σ is a suggestion for $\mathtt{log}_{G_{r+1}}$ and M is a *justification* for σ.

(r, v)-proposals. An (r, v)-proposal is a message of the form $R = (P, v, Q)_{p_j}$, where P is an r-proposal, $p_j \in \Pi$, and either $Q = \perp$ or else Q is a QC (as specified below) for some (r, v')-proposal with $v' < v$.

Votes. A *vote* for the (r, v)-proposal $R = (P, v, Q)_{p_j}$, where $P = (F, \sigma, M, r)$, is a message of the form $V = R_{p_k}$, where $p_k \in \Pi$. We also say V is a *vote by* p_k. At timeslot t, p_i will regard V as *valid* if it is signed by one of the processes that, from p_i's perspective, remains in the active process set – i.e., if Π_r is defined and $p_k \in \Pi_r - F$.

QCs. A QC for an (r, v)-proposal $R = (P, v, Q')_{p_j}$, where $P = (F, \sigma, M, r)$, is a set Q of votes for R. At timeslot t, p_i will regard Q as valid if every vote in Q is valid and Q contains more than $\frac{1}{2}|\Pi_r - F|$ votes, each by a different process. If Q is a QC for an (r, v)-proposal $R = (P, v, Q')_{p_j}$, we set $\mathtt{view}(Q) = (r, v)$ and $\mathtt{P}(Q) = P$, and we may also just refer to Q as a QC.

Locks. Each process p_i maintains a value Q_i^+, which is initially undefined. This variable should be thought of as playing the same role as locks in Tendermint. The variable Q_i^+ may be set to a valid QC for an (r, v)-proposal during view (r, v).

The variables $P_i(r)$ and t_0. Process p_i maintains a local variable $P_i(r)$ for each $r \geq 1$, initially undefined. Upon halting execution $\mathcal{E}_r$ and entering the recovery procedure at timeslot t (according to its local clock), p_i will set $t_0 := t$, wait $2\Delta^*$, and then set $P_i(r)$ to be the set of processes in Π_r from which it has received signed r-genesis messages.

The time for each view. Each view is of length $8\Delta^*$. Having set t_0 upon halting execution $\mathcal{E}_r$ and entering the recovery procedure, p_i will start view (r, v) (for $v \geq 1$) at time $t_0 + 2\Delta^* + 8(v-1)\Delta^*$.

Detecting equivocation. At timeslot t, we say p_i *detects equivocation in view* (r, v) if $\mathtt{M}_i$ contains at least two distinct (r, v)-proposals signed by $\mathtt{lead}(r, v)$.[14]

Valid (r, v)-proposals. Consider an (r, v)-proposal $R = (P, v, Q)_{p_j}$, where $P = (F, \sigma, M, r)$. At timeslot t (according to p_i's local clock), process p_i will regard R as valid if:

- **(i)** Π_r and $\mathtt{log}_{G_r}$ are defined;
- **(ii)** $F \subset \Pi_r$, and $|F| \geq \rho_C|\Pi_r|$;
- **(iii)** $\mathtt{M}_i$ is a $(\mathcal{P}, \Pi_r, \mathtt{log}_{G_r})$-proof of guilt for every process in F;
- **(iv)** M is a set of r-genesis messages, each signed by a different process in $\Pi_r - F$;
- **(v)** For each $p_k \in P_i(r)$, there exists an r-genesis message signed by p_k in M;
- **(vi)** σ is the longest sequence extended by more than $\frac{1}{2}|\Pi_r - F|$ elements of M;
- **(vii)** $p_j = \mathtt{lead}(r, v)$;
- **(viii)** Q_i^+ is undefined, or Q is a valid QC with (a) $\mathtt{view}(Q) \geq \mathtt{view}(Q_i^+)$, and (b) $\mathtt{P}(Q) = P$, and;
- **(ix)** p_i does not detect equivocation in view (r, v).

The local variables `voted` and `lockset`. For each (r, v), $\mathtt{voted}(r, v)$ and $\mathtt{lockset}(r, v)$ are initially set to 0. These values are used to indicate whether p_i has yet voted or set its lock during view (r, v).

***r*-finish votes and QCs.** An *r-finish vote for P* is a message of the form P_{p_j}, where $P = (F, \sigma, M, r)$ is an r-proposal and $p_j \in \Pi$. At timeslot t, p_i will regard the r-finish vote as valid if Π_r is defined and $p_j \in \Pi_r - F$. A *valid finish-QC* for P is a set of more than $\frac{1}{2}|\Pi_r - F|$ valid r-finish votes for P, each signed by a different process.

The procedure `Makeproposal`. If $p_i = \mathtt{lead}(r, v)$, then it will run this procedure during view (r, v). To carry out the procedure, p_i checks to see whether there exists some greatest $v' < v$ such that it has received a valid QC, Q say, with $\mathtt{view}(Q) = (r, v')$. If so, then p_i sends the (r, v)-proposal $R = (\mathtt{P}(Q), v, Q)_{p_i}$ to all processes. If not, then it sets F to be the set of all processes $p_j \in \Pi_r$ such that $\mathtt{M}_i$ is a $(\mathcal{P}, \Pi_r, \mathtt{log}_{G_r})$-proof of guilt for p_j. Let M be the set of r-genesis messages that p_i has received and which are signed by processes in $\Pi_r - F$, and let M' be a maximal subset of M that contains at most one message signed by each process. Process p_i then sets σ to be the longest sequence extended by more than $\frac{1}{2}|\Pi_r - F|$ elements of M' and sends to all processes the (r, v)-proposal $R = (P, v, \perp)_{p_i}$, where $P = (F, \sigma, M', r)$.

Message gossiping. We adopt the message gossiping conventions described in Section 2.

[14] If $\mathtt{M}_i$ contains a vote for an (r, v)-proposal, we consider it as also containing that (r, v)-proposal.

The function $\mathcal{F}$. While the function $\mathcal{F}$ is not explicitly used in the pseudocode, we will show in Section 7 that, at every t, $\texttt{log}_i = \mathcal{F}(\texttt{M}_i)$ (where $\texttt{log}_i$ and $\texttt{M}_i$ are as locally defined for p_i at t). The function $\mathcal{F}$ is specified in Algorithm 2.

Pseudocode walk-through. The pseudocode appears in Algorithm 1. Below, we summarise the function of each section of code.

Line 14. This line starts the execution of the wrapper by initiating $\mathcal{E}_1$, the first execution of $\mathcal{P}$, which has process set $\Pi_1 = \Pi$ and $\texttt{log}_{G_1} = \texttt{log}_G$ as the genesis log.

Lines 16 – 19. During the r^{th} execution of $\mathcal{P}$, these lines check whether the recovery procedure should be triggered. If so, then p_i disseminates an r-genesis message, temporarily resets its log, and starts the recovery procedure.

Lines 21 – 25. During the r^{th} execution of $\mathcal{P}$, these lines check whether p_i should extend its finalized and strongly finalized logs.

Lines 28 – 30. These lines initialize the r^{th} execution of the recovery procedure by setting t_0 and $P_i(r)$.

Lines 32 – 41. These lines specify the instructions for view (r, v). Initially, the leader waits $2\Delta^*$ and then makes an (r, v)-proposal. Processes vote upon receiving a first valid (r, v)-proposal. Upon receiving a first valid QC for an (r, v)-proposal, Q say, p_i sets its lock to Q and then waits $2\Delta^*$. If, at this time, it still does not detect equivocation in view (r, v), then it sends a finish vote for $\texttt{P}(Q)$.

Lines 43 – 47. These lines determine when p_i stops carrying out the r^{th} execution of the recovery procedure. This happens when p_i receives a valid finish-QC for some r-proposal P. The r-proposal P then specifies Π_{r+1} and $\texttt{log}_{G_{r+1}}$.

Informal discussion: how does the recovery procedure ensure consensus? To establish that at most one r-proposal can receive a valid finish-QC, suppose that some correct p_i sends a finish vote for the r-proposal P during view (r, v). In this case, p_i must set its lock to some valid QC, Q say, at some timeslot t while in view v. Suppose that Q is a QC for the (r, v)-proposal R, and note that $\texttt{P}(Q) = P$. We will observe that:

1. All correct processes set their locks to some valid QC for R while in view v.
2. No (r, v)-proposal other than R can receive a QC that is regarded as valid by any correct process.

From (1) and (2) it will be easy to argue by induction on $v' > v$ that no correct process votes for any proposal $R' = (P', v', Q')_{p_j}$ such that $P' \neq P$, since their locks will forever prevent voting for such proposals. It follows that if any correct p_k sends a finish vote for an r-proposal P' during some view $v' \geq v$, then $P = P'$. We conclude that, assuming (1) and (2), at most one r-proposal can receive a valid finish-QC.

To see that (1) holds, note that all correct processes will be in view (r, v) at $t + \Delta^*$ and will have received Q by this time. They will therefore set their lock to be some QC for R, unless they have already received a valid QC for some (r, v)-proposal $R' \neq R$. The latter case is not possible, since then p_i would detect equivocation in view (r, v) by $t + 2\Delta^*$, and so would not send the finish vote for P.

Algorithm 1 The instructions for p_i.

Local variables
$\mathtt{r}$, initially 1. ▷ Number of executions of $\mathcal{P}$ initiated
$\mathtt{rec}$, initially 0. ▷ 1 if carrying out recovery
$\mathtt{log}_i$, $\mathtt{log}_i^*$, initially set to $\mathtt{log}_G$ ▷ Finalized and strongly finalized transactions
Π_r, $r \geq 1$. Initially, $\Pi_1 = \Pi$, while $\Pi_r \uparrow$ for $r > 1$. ▷ Process set for $\mathcal{E}_r$
$\mathtt{log}_{G_r}$. Initially, $\mathtt{log}_{G_1} = \mathtt{log}_G$, while $\mathtt{log}_{G_r} \uparrow$ for $r > 1$. ▷ Genesis log for $\mathcal{E}_r$
$\mathtt{M}_i$, $\mathtt{M}_{i,r}$, initially empty. ▷ As specified in Section 5
Q_i^+, initially undefined. ▷ The lock
t_0, initially undefined. ▷ Timeslot at start of recovery
$P_i(r)$, initially undefined. ▷ A set of processes
$\mathtt{voted}(r,v)$, $\mathtt{lockset}(r,v)$ $(r,v \geq 1)$, initially 0. ▷ As specified in Section 5

At timeslot t:
  If $t = 0$, start execution $\mathcal{E}_1$ of $\mathcal{P}$, with process set Π_1 and with $\mathtt{log}_{G_1}$ as genesis log;

  If $\mathtt{rec} = 0$:
    If $\mathtt{M}_{i,\mathtt{r}}$ has a consistency violation w.r.t. $\mathcal{F}(\Pi_\mathtt{r}, \mathtt{log}_{G_\mathtt{r}})$:
      Send $(\mathtt{gen}, \mathtt{log}_i, \mathtt{r})_{p_i}$ to all processes; ▷ Send $\mathtt{r}$-genesis message
      Set $\mathtt{log}_{p_i} := \mathtt{log}_{G_\mathtt{r}}$; Stop running $\mathcal{E}_\mathtt{r}$; Set $\mathtt{rec} := 1$; ▷ Start recovery

  If $\mathtt{rec} = 0$:
    If $\exists \sigma, M$ s.t. $\sigma \succ \mathtt{log}_i$ and $M \subseteq \mathtt{M}_{i,\mathtt{r}}$ is an $\mathcal{F}(\Pi_\mathtt{r}, \mathtt{log}_{G_\mathtt{r}})$-certificate for σ;
      Let σ be the longest such; Set $\mathtt{log}_i := \sigma$; ▷ Extend log
    If there exists a longest $\sigma \succ \mathtt{log}_i^*$ s.t. $\mathtt{log}_i$ has extended σ for time $2\Delta^*$:
      Set $\mathtt{log}_i^* := \sigma$; ▷ Extend strongly finalized log

  If $\mathtt{rec} = 1$:
    If $t_0 \uparrow$, set $t_0 := t$; ▷ Set t_0 upon entering recovery
    If $t = t_0 + 2\Delta^*$: ▷ Set $P_i(\mathtt{r})$
      Set $P_i(\mathtt{r}) := \{p_j \in \Pi_\mathtt{r} :$ $\mathtt{M}_i$ contains an $\mathtt{r}$-genesis message signed by $p_j\}$;

    If $t = t_0 + 4\Delta^* + 8(v-1)\Delta^*$ (for some $v \in \mathbb{N}_{\geq 1}$) **and** $p_i = \mathtt{lead}(\mathtt{r}, v)$:
      $\mathtt{Makeproposal}$; ▷ Leader makes new proposal $2\Delta^*$ after starting view
    If $t \in [t_0 + 2\Delta^* + 8(v-1)\Delta^*, t_0 + 2\Delta^* + 8v\Delta^*)$ (for some $v \in \mathbb{N}_{\geq 1}$):
      If $\mathtt{voted}(\mathtt{r}, v) = 0$ **and** $\mathtt{M}_i$ contains a valid $(\mathtt{r}, v)$-proposal R:
        Send R_{p_i} to all processes; Set $\mathtt{voted}(\mathtt{r}, v) := 1$; ▷ Vote
      If $\mathtt{lockset}(\mathtt{r}, v) = 0$ **and** $\mathtt{M}_i$ contains a valid QC for an $(\mathtt{r}, v)$-proposal, Q say:
        Set $Q_i^+ := Q$, $\mathtt{lockset}(\mathtt{r}, v) := 1$; ▷ Set lock
        Set the $(\mathtt{r}, v)$-timer to expire in time $2\Delta^*$;
      If $(\mathtt{r}, v)$-timer expires and p_i does not detect equivocation in view $(\mathtt{r}, v)$:
        Send $\mathtt{P}(Q_i^+)_{p_i}$ to all processes; ▷ Send finish vote

    If $\mathtt{M}_i$ contains a valid finish-QC for some $P = (F, \sigma, M, \mathtt{r})$:
      Set $\Pi_{\mathtt{r}+1} := \Pi_\mathtt{r} - F$, $\mathtt{log}_{G_{\mathtt{r}+1}} := \sigma$; ▷ Start new execution of $\mathcal{P}$
      Set $\mathtt{r} := \mathtt{r} + 1$ and make t_0 and Q_i^+ undefined;
      Start execution $\mathcal{E}_\mathtt{r}$ of $\mathcal{P}$, with process set $\Pi_\mathtt{r}$ and with $\mathtt{log}_{G_\mathtt{r}}$ as genesis log;
      Set $\mathtt{rec} := 0$; Set $\mathtt{log}_i := \mathtt{log}_{G_\mathtt{r}}$;

Algorithm 2 The function $\mathcal{F}$.

Inputs
M ▷ A set of messages
Π, $\mathtt{log}_G$ ▷ Process set and genesis log
$\mathcal{F}(\Pi', \mathtt{log}'_G)$ ▷ A function for each possible Π' and $\mathtt{log}'_G$
Local variables
$\mathtt{r}$, initially 1.
Π_r, $r \geq 1$. Initially, $\Pi_1 = \Pi$, while $\Pi_r \uparrow$ for $r > 1$.
$\mathtt{log}_{G_r}$. Initially, $\mathtt{log}_{G_1} = \mathtt{log}_G$, while $\mathtt{log}_{G_r} \uparrow$ for $r > 1$.
$\mathtt{end}$, initially 0

While $\mathtt{end} = 0$ **do**:
  If M does not have a consistency violation w.r.t. $\mathcal{F}(\Pi_{\mathtt{r}}, \mathtt{log}_{G_{\mathtt{r}}})$:
    Let σ be longest such that M is an $\mathcal{F}(\Pi_{\mathtt{r}}, \mathtt{log}_{G_{\mathtt{r}}})$-certificate for σ;
    Return σ; Set $\mathtt{end} := 1$;
  Else if there does not exist a unique $\mathtt{r}$-proposal with a valid finish-QC in M:
    Return $\mathtt{log}_{G_{\mathtt{r}}}$; Set $\mathtt{end} := 1$;
  Else if there exists a unique $\mathtt{r}$-proposal $P = (F, \sigma, M', \mathtt{r})$ with a valid finish-QC in M:
    Set $\Pi_{\mathtt{r}+1} := \Pi_{\mathtt{r}} - F$, $\mathtt{log}_{G_{\mathtt{r}+1}} = \sigma$;
    Set $\mathtt{r} := \mathtt{r} + 1$;

To see that (2) holds, the argument is similar. All correct processes will be in view (r, v) at $t + \Delta^*$ and will have received R by this time. Item (ix) in the validity conditions for (r, v)-proposals prevents correct processes from voting for (r, v)-proposals $R' \neq R$ at later timeslots, and correct processes cannot vote for such proposals at any timeslot $\leq t + \Delta^*$ because p_i would detect equivocation in view (r, v) in this case.

Having established that at most one r-proposal can receive a valid finish-QC, suppose now, towards a contradiction, that no r-proposal ever receives a valid finish-QC. Let v be the least such that $\mathtt{lead}(r, v)$ is correct and let i be such that $p_i = \mathtt{lead}(r, v)$. Since p_i waits $2\Delta^*$, until some timeslot t say, before disseminating an (r, v)-proposal $R = (P, v, Q)_{p_i}$, it will have seen all locks held by correct processes by this time, and will have received r-genesis messages from all processes in any set $P_j(r)$ for correct p_j. At t, p_i will disseminate an (r, v)-proposal which all correct processes regard as valid by timeslot $t + \Delta^*$. All correct processes will therefore vote for the proposal by this time and will receive a valid QC for the proposal by time $t + 2\Delta^*$. All correct processes will then set their locks. They will still be in view (r, v) by time $t + 4\Delta^*$ (since they enter the view within time Δ^* of each other) and will send r-finish votes for P by this time.

6 The theorem statements

Given functions $g, g' : \mathbb{N} \to \mathbb{R}$, we say $g \leq g'$ if $g(r) \leq g'(r)$ for all $r \in \mathbb{N}$. If $x \in \mathbb{R}$, we say $g \leq x$ if $g(r) \leq x$ for all $r \in \mathbb{N}$. We say $g < g'$ if $g \leq g'$ and $g(r) < g'(r)$ for some n.

We begin with our main positive result, which states the guarantees our wrapper achieves for recoverable consistency and liveness, worst-case and probabilistic recovery time, and rollback. The proof of Theorem 1 is given in Section 7.

▶ **Theorem 1.** *Suppose the wrapper is given an accountable SMR protocol $\mathcal{P}$ as input, where $\mathcal{P}$ has consistency resilience ρ_C and liveness resilience ρ_L in partial synchrony, such that $\rho_C + 2\rho_L = 1$. Let g_1 and g_2 be as defined in expression (1) in Section 3. If $\mathcal{P}$ has liveness parameter ℓ and is accountable for $(1-\rho_L)$-bounded adversaries, then the wrapper produces a protocol with the same consistency and liveness resilience as $\mathcal{P}$ in partial synchrony, and with the following properties for $(1-\rho_L)$-bounded adversaries when message delays are bounded by Δ^*:*

- **(i)** *Recoverable consistency resilience $\geq g_1$ and recoverable liveness resilience $\geq g_2$.*
- **(ii)** *Recovery time $O(f_a\Delta^*)$ with liveness parameter ℓ, where f_a is the actual (unknown) number of faulty processes.*
- **(iii)** *Probabilistic recovery time $O(\Delta^* log\frac{1}{\varepsilon})$ with liveness parameter ℓ.*
- **(iv)** *Rollback bounded by $2\Delta^*$.*

The next three results describe senses in which Theorem 1 is tight. We say a protocol has *bounded rollback* if there exists some h such that the protocol has rollback bounded by h. Our first impossibility result states that the rollback of a protocol must scale with Δ^*, and hence bounded rollback in the partially synchronous setting is impossible.

▶ **Theorem 2** (Impossibility result 1)**.** *Suppose partial synchrony holds w.r.t. Δ and synchrony holds w.r.t. Δ^*. Suppose $\mathcal{P}$ is a protocol for SMR with liveness resilience ρ_L, consistency resilience $\rho_C \geq \rho_L$, liveness parameter ℓ, and with rollback bounded by h. If we are given only that the adversary is ρ-bounded for $\rho > 1 - 2\rho_L$, then $h = \Omega(\Delta^*)$. In particular, $\mathcal{P}$ does not have bounded rollback in the pure partially synchronous setting.*

The proof of Theorem 2 is given in the full online version of the paper. Our second impossibility result justifies our restriction to $(1-\rho_L)$-bounded adversaries: with a larger adversary, bounded rollback is impossible (even in the synchronous setting).

▶ **Theorem 3** (Impossibility result 2)**.** *Consider the synchronous setting and suppose $\mathcal{P}$ is a protocol for SMR with liveness resilience ρ_L and consistency resilience $\rho_C \geq \rho_L$. If we are given only that the adversary is ρ-bounded for $\rho > 1 - \rho_L$, then $\mathcal{P}$ does not have bounded rollback. (The same result also holds in partial synchrony.)*

The proof of Theorem 3 is given in the full online version of the paper. Our final impossibility result shows that the recoverable consistency and liveness functions g_1 and g_2 in Theorem 1 cannot be improved upon, giving an analog of the "$\rho_C + 2\rho_L \leq 1$" constraint for all positive values of r.

▶ **Theorem 4** (Impossibility result 3)**.** *Given ρ_C and ρ_L such that $\rho_C + 2\rho_L = 1$, let g_1 and g_2 be as defined in Section 3. Suppose $g_1', g_2' \leq 1 - \rho_L$ and that $\mathcal{P}$ is an SMR protocol for partial synchrony with recoverable consistency resilience $\geq g_1'$ and recoverable liveness resilience $\geq g_2'$ when message delays are bounded by Δ^*. Suppose that, for some d and ℓ, $\mathcal{P}$ has recovery time d with liveness parameter ℓ when the adversary is $1-\rho_L$-bounded. Then:*

1. *If $g_2' \geq g_2$, then $g_1' \leq g_1$, and;*
2. *If $g_2' > g_2$, then $g_1' < g_1$.*

The proof of Theorem 4 is given in the full online version of the paper.

7 The proof of Theorem 1

We assume throughout this section that the adversary is $(1-\rho_L)$-bounded and that message delays are bounded by Δ^*.

Some further terminology. We make the following definitions:

- Process p_i begins the r^{th} execution of the recovery procedure at the first timeslot at which $\mathtt{r} = r$ and $\mathtt{rec} = 1$ (where those values are as locally defined for p_i).
- The r^{th} execution of the recovery procedure begins at the first timeslot at which some correct process begins the r^{th} execution of the recovery procedure.
- Execution $\mathcal{E}_r$ begins at the first timeslot at which some correct process begins execution $\mathcal{E}_r$. If $r > 1$, then the $(r-1)^{\text{th}}$ execution of the recovery procedure also ends at this timeslot.
- If a QC/finish-QC is regarded as valid by all correct processes, we refer to it as a valid QC/finish-QC.

▶ **Lemma 5.** *If the r^{th} execution of the recovery procedure begins at t_0, then:*

(i) *All correct processes begin the r^{th} execution of the recovery procedure by time $t_0 + \Delta^*$.*

(ii) *There exists a unique r-proposal, P say, that receives a finish-QC that is regarded as valid by some correct process.*

(iii) *If v_0 is least such that $\mathtt{lead}(r, v_0)$ is correct, all correct processes receive a valid finish-QC for P by time $t_0 + 2\Delta^* + 8v_0\Delta^*$.*

(iv) *All correct processes begin execution $\mathcal{E}_{r+1}$ within time Δ^* of each other and with the same local values for Π_{r+1} and $\mathtt{log}_{G_{r+1}}$.*

The proof of Lemma 5 is given in the full online version of the paper.

Further notation. Given statement (iv) of Lemma 5, each value Π_r or $\mathtt{log}_{G_r}$ is either undefined at all timeslots for all correct processes, or else is eventually defined and takes the same value for each correct process. We may therefore write Π_r and $\mathtt{log}_{G_r}$ to denote these globally agreed values.

▶ **Lemma 6.** *If p_i is correct, then, at the end of every timeslot, $\mathtt{log}_i = \mathcal{F}(\mathtt{M}_i)$.*

Proof. Let $\mathtt{rec}$, $\mathtt{r}$, $\mathtt{M}_i$, $\mathtt{M}_{i,r}$ and $\mathtt{log}_i$ be as locally defined for p_i. Consider first the case that $\mathtt{rec} = 0$ at the end of timeslot t. In this case, $\mathtt{M}_i$ has a consistency violation with respect to $\mathcal{F}(\Pi_r, \mathtt{log}_{G_r})$ for each $r < \mathtt{r}$ but does not have a consistency violation with respect to $\mathcal{F}(\Pi_\mathtt{r}, \mathtt{log}_{G_\mathtt{r}})$. Also, $\mathtt{M}_i$ contains a valid finish-QC for some r-proposal for each $r < \mathtt{r}$, which must be unique by Lemma 5. At the end of timeslot t, $\mathtt{log}_i$ is the longest string σ such that $\mathtt{M}_i$ (and $\mathtt{M}_{i,\mathtt{r}}$) is an $\mathcal{F}(\Pi_\mathtt{r}, \mathtt{log}_{G_\mathtt{r}})$-certificate for σ. The iteration defining $\mathcal{F}$ in Algorithm 2 will not return a value until it has defined all values Π_r and $\mathtt{log}_{G_r}$ for $r \leq \mathtt{r}$. Upon discovering that $\mathtt{M}_i$ does not have a consistency violation with respect to $\mathcal{F}(\Pi_\mathtt{r}, \mathtt{log}_{G_\mathtt{r}})$, it will return the same value σ, as the longest string for which $\mathtt{M}_i$ is an $\mathcal{F}(\Pi_\mathtt{r}, \mathtt{log}_{G_\mathtt{r}})$-certificate.

Next, consider the case that $\mathtt{rec} = 1$ at the end of timeslot t. In this case, $\mathtt{M}_i$ has a consistency violation with respect to $\mathcal{F}(\Pi_r, \mathtt{log}_{G_r})$ for each $r \leq \mathtt{r}$, and also contains a valid finish-QC for some r-proposal for each $r < \mathtt{r}$, which must be unique by Lemma 5. However, $\mathtt{M}_i$ does not contain a valid finish-QC for any $\mathtt{r}$-proposal. At the end of timeslot t, $\mathtt{log}_i = \mathtt{log}_{G_\mathtt{r}}$. The iteration defining $\mathcal{F}$ will not return a value until it has defined all values Π_r and $\mathtt{log}_{G_r}$ for $r \leq \mathtt{r}$, and will then also return $\mathtt{log}_{G_\mathtt{r}}$. ◀

▶ **Lemma 7.** *The wrapper has rollback bounded by $2\Delta^*$. Also, $\mathtt{log}_{G_{r+1}} \succeq \mathtt{log}_{G_r}$ whenever $\mathtt{log}_{G_{r+1}} \downarrow$.*

Proof. We say p_i finalizes σ if it sets $\mathtt{log}_i$ to extend σ and that p_i strongly finalizes σ if it sets $\mathtt{log}_i^*$ to extend σ. Suppose p_i finalizes σ while running $\mathcal{E}_r$ at t because there exists $M \subseteq \mathtt{M}_{i,r}$ which is an $\mathcal{F}(\Pi_r, \mathtt{log}_{G_r})$-certificate for σ. By (iv) of Lemma 5, every correct process p_j will begin $\mathcal{E}_r$ by $t + \Delta^*$, and will receive the messages in M by that time. This means p_j will finalize σ, never to subsequently finalize any sequence incompatible with σ

while running $\mathcal{E}_r$, unless $\mathtt{M}_{j,r}$ has a consistency violation w.r.t. $\mathcal{F}(\Pi_r, \mathtt{log}_{G_r})$ by $t + \Delta^*$. In the latter case, p_i will begin the recovery procedure by timeslot $t + 2\Delta^*$ and will not strongly finalize σ at that time. We conclude that, if p_i strongly finalizes σ while running $\mathcal{E}_r$, then all correct processes finalize σ while running $\mathcal{E}_r$.

If p_i strongly finalizes σ while running $\mathcal{E}_r$ and if the r^{th} execution of the recovery procedure does not begin at any timeslot, it follows that, for all correct p_j, $\sigma \preceq \mathtt{log}_j$ thereafter. So, suppose that the r^{th} execution of the recovery procedure begins at some timeslot t_0. Note that, if p_j is correct, then it waits $2\Delta^*$ after beginning the r^{th} execution of the recovery procedure before defining $P_j(r)$. By Lemma 5, all correct processes begin the r^{th} execution of the recovery procedure within time Δ^* of each other. Since correct processes send r-genesis messages immediately upon beginning the recovery procedure, it follows that $P_j(r)$ includes all correct processes.

By Lemma 5, there exists a unique r-proposal, $P = (F, \sigma', M', r)$ say, that receives a valid finish-QC. For this to occur, there must exist v and an (r, v)-proposal $R = (P, v, \bot)$ (signed by $\mathtt{lead}(r, v)$) which receives a valid QC. This QC must include at least one vote by a correct process, p_j say. It follows that M' must contain r-genesis messages from every member of $P_j(r)$, and so from every correct process. As we noted previously, if p_i strongly finalizes σ while running $\mathcal{E}_r$, then every correct process must finalize σ before beginning the r^{th} execution of the recovery procedure. So, for each r-genesis message $(\text{gen}, \sigma'', r)$ sent by a correct process, σ'' must extend σ, and also extends $\mathtt{log}_{G_r}$. It therefore holds that σ (and $\mathtt{log}_{G_r}$) is extended by more than $\frac{1}{2}|\Pi_r - F|$ elements of M'. Recall that σ' is as specified by P. No correct process will vote for R unless σ' is the longest sequence extended by more than $\frac{1}{2}|\Pi_r - F|$ elements of M, meaning that σ' must extend σ and $\mathtt{log}_{G_r}$. So far, we conclude that $\mathtt{log}_{G_{r+1}}$ extends σ and $\mathtt{log}_{G_r}$. Since it follows by the same argument that $\mathtt{log}_{G_s}$ extends σ for all $s > r$, the claim of the lemma holds. ◀

▶ **Lemma 8.** *If an execution of the wrapper has r consistency violations, then the r^{th} execution of the recovery procedure must begin at some timeslot (and so, by Lemma 5 must also end at some timeslot).*

The proof of Lemma 8 is given in the full online version of the paper.

▶ **Lemma 9.** *The wrapper has recoverable consistency resilience $\geq g_1$ and also recoverable liveness resilience $\geq g_2$.*

Proof. Recall that, in Section 3, we set $x_0 = 0$ and $x_{r+1} = x_r + \rho_C(1 - x_r)$, and then defined:

$$g_1(r) = \min\{x_{r+1}, 1 - \rho_L\}, \quad g_2(r) = \min\{x_r + \rho_L(1 - x_r), 1 - \rho_L\}.$$

Note that x_r lower bounds the fraction of the processes removed to form Π_{r+1}, i.e. $|\Pi - \Pi_{r+1}| \geq x_r n$. By Lemma, 8, if there exist r consistency violations, then the r^{th} execution of the recovery procedure must end at some timeslot. If the adversary is $g_2(k)$-bounded, and since $\mathcal{P}$ has liveness resilience ρ_L, it follows that liveness must hold. If the adversary is $g_1(r)$-bounded, then since $\mathcal{P}$ has consistency resilience ρ_C, the $(r+1)^{\text{th}}$ execution of the recovery procedure cannot begin at any timeslot. From Lemma 8, it follows that there are at most r consistency violations. ◀

▶ **Lemma 10.** *The wrapper has recovery time $O(f_a\Delta^*)$ with liveness parameter ℓ, where f_a is the actual (unknown) number of faulty processes. It also has probabilistic recovery time $O(\Delta^* log\frac{1}{\varepsilon})$ with liveness parameter ℓ.*

Proof. The fact that the wrapper has recovery time $O(f_a\Delta^*)$ with liveness parameter ℓ follows directly from (iii) of Lemma 5, since views are of length $O(\Delta^*)$. To establish the claim regarding probabilistic recovery time, note that we required $\rho_C > 0$ in the definition of optimal resilience. Some finite power of $(1 - \rho_C)$ is therefore less than ρ_L, so there exists r such that any execution in which the adversary is $1 - \rho_L$-bounded can have most r consistency violations. If the adversary is ρ-bounded, then the probability that, for one of the (at most r) executions of the recovery procedure, the first d views all have faulty leaders is $O(r\rho^d) = O(\rho^d)$ for fixed ρ_C. Since each view is of length $O(\Delta^*)$, it follows from (iii) of Lemma 5 that the wrapper therefore has probabilistic recovery time $O(\Delta^* \log \frac{1}{\varepsilon})$ with liveness parameter ℓ, as claimed. ◀

8 Related Work

Positive results. A sequence of papers, including Buterin and Griffith [7], Civit et al. [10], and Shamis et al. [23], describe protocols that satisfy accountability. Sheng et al. [24] analyze accountability for well-known permissioned protocols such as HotStuff [29], PBFT [8], Tendermint [4, 5], and Algorand [9]. Civit et al. [12, 11] describe generic transformations that take any permissioned protocol designed for the partially synchronous setting and provide a corresponding accountable version. These papers do not describe how to reach consensus on which guilty parties to remove in the event of a consistency violation (i.e. how to achieve "recovery"), and thus fall short of our goals here. One exception to this point is the ZLB protocol of Ranchal-Pedrosa and Gramoli [21], but the ZLB protocol only achieves recovery if the adversary controls less than a 5/9 fraction of participants, and does not achieve bounded rollback. Freitas de Souza et al. [13] also describe a process for removing guilty parties in a protocol for lattice agreement (this abstraction is weaker than SMR/consensus and can be implemented in an asynchronous system), but their protocol assumes an honest majority and the paper does not consider bounded rollback. Sridhar et al. [26] specify a "gadget" that can be applied to blockchain protocols operating in the synchronous setting to reboot and maintain consistency after an attack, but they do not describe how to implement recovery and assume that an honest majority is somehow reestablished out-of-protocol.

Budish et al. [6] consider "slashing" in proof-of-stake protocols in the "quasi-permissionless" setting. Their main positive result is a protocol that, in the same timing model considered in this paper (with additional guarantees provided pre-GST message delays are bounded by a known parameter Δ^*), guarantees what they call the "EAAC property" – honest players never have their stake slashed, and some Byzantine stake is guaranteed to be slashed following a consistency violation. Budish et al. [6] do not contemplate repeated consistency violations, a prerequisite to the notions of recoverable consistency and liveness that are central to this paper. To the extent that it makes sense to compare their "recovery procedure" with our "wrapper," our protocol is superior in several respects, with worst-case recovery time $O(n\Delta^*)$ (as opposed to $O(n^2\Delta^*)$); probabilistic recovery time $O(\Delta^* \log \frac{1}{\varepsilon})$, where ε is an error-probability bound (as opposed to $O(n\Delta^* \log \frac{1}{\varepsilon})$); and rollback $2\Delta^*$ (as opposed to unbounded rollback).

Gong et al. [15] consider the task of recovery and achieve results that are incomparable to those in this paper since they consider different timing and fault models: they restrict to the case of "ABC faults"[15] but consider full partial synchrony, meaning that it is not possible to

[15] Processes subject to "ABC faults" can only display faulty behaviour so as to cause consistency failures, and not to threaten liveness.

achieve bounded rollback. Given this setup, their protocol satisfies the condition that after a single execution of the recovery procedure, a 5/9-bounded adversary is unable to cause further consistency and liveness violations, and after two executions of the recovery procedure a 2/3-bounded adversary is unable to cause further consistency and liveness violations. The authors also describe sufficient conditions for a general BFT SMR protocol to allow for "complete and sound fault detection" in the case of consistency violations, even when the actual (unknown) number of faulty processes is as large as $n - 2$.

Distinct from our aims here, i.e., the *removal* of guilty parties so as to achieve recovery in the event of a consistency violation, a number of papers consider the ability of protocols to recover from temporary dishonest majorities (without consideration of any mechanism to ensure that the dishonest majority is temporary). Avarikioti et al. [2] establish a formal sense in which Bitcoin is secure under temporary dishonest majority. We note that, since Bitcoin is dynamically available [17], it cannot be accountable [20], meaning that there can be no mechanism to remove guilty parties (and only guilty parties) in the event of a consistency violation. Badertscher et al. [3] extend this analysis to consider dynamically available proof-of-work and proof-of-stake protocols more generally and also establish negative results for BFT-style protocols that do not make use of accountability to remove guilty parties (as we do here).

Prior to the study of accountability, Li and Mazieres [18] considered how to design BFT protocols that still offer certain guarantees when more than f failures occur. The describe a protocol called BFT2F which has the same liveness and consistency guarantees as PBFT when no more than $f < n/3$ players fail; with more than f but no more than $2f$ failures, BFT2F prohibits malicious players from making up operations that clients have never issued and prevents certain kinds of consistency violations.

Negative results. There are a number of papers that describe negative results relating to accountability and the ability to punish guilty parties in the "permissionless setting" (for a definition of the permissionless setting see [17]). Neu et al. [20] prove that no protocol operating in the "dynamically available" setting (where the number of "active" parties is unknown) can provide accountability. The authors then provide an approach to addressing this limitation by describing a "gadget" that checkpoints a longest-chain protocol. The "full ledger" is then live in the dynamically available setting, while the checkpointed prefix ledger provides accountability. Tas et al. [28, 27] and Budish et al. [6] also prove negative results regarding the possibility of punishing guilty participants of proof-of-stake protocols before they are able to cash out of their position.

9 Final Comments

For the sake of simplicity, we have presented our wrapper in the "permissioned" setting (with a fixed and known set of always active participants). However, since the procedure produces certificates that suffice to verify each new genesis log and the set of processes removed after each consistency violation, it can also be applied directly to proof-of-stake protocols in the quasi-permissionless setting [17]. Specifically, our procedure can be used to give a practical replacement for the (proof-of-concept) recovery procedure of Budish, Lewis-Pye and Roughgarden [6] within the formal setup they consider.

While Theorems 2-3 show senses in which Theorem 1 is tight, a number of natural questions remain. For example, our recovery procedure implements a synchronous protocol and has recovery time $O(f_a\Delta^*)$. While Theorem 2 establishes that some bound on message delays is required if we are to achieve bounded rollback, one might still make use of a

recovery procedure that does not require synchrony: could such a procedure achieve recovery time $O(f_a\Delta)$ after GST? Also, while our recovery procedure has rollback bounded by $2\Delta^*$, Theorem 2 only establishes a lower bound of Δ^*. Is this lower bound tight, or is $2\Delta^*$ optimal?

References

1. Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 106–118. IEEE, 2020. doi:10.1109/SP40000.2020.00044.
2. Georgia Avarikioti, Lukas Käppeli, Yuyi Wang, and Roger Wattenhofer. Bitcoin security under temporary dishonest majority. In *International Conference on Financial Cryptography and Data Security*, pages 466–483. Springer, 2019. doi:10.1007/978-3-030-32101-7_28.
3. Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Consensus redux: distributed ledgers in the face of adversarial supremacy. In *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*, pages 143–158. IEEE, 2024.
4. Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.
5. Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint*, 2018. arXiv:1807.04938.
6. Eric Budish, Andrew Lewis-Pye, and Tim Roughgarden. The economic limits of permissionless consensus. In *Proceedings of the 25th ACM Conference on Economics and Computation*, pages 704–731, 2024. doi:10.1145/3670865.3673548.
7. Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017. arXiv:1710.09437.
8. Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation OsDI*, pages 173–186, 1999.
9. Jing Chen, Sergey Gorbunov, Silvio Micali, and Georgios Vlachos. Algorand agreement: Super fast and partition resilient byzantine agreement. *IACR Cryptol. ePrint Arch.*, 2018:377, 2018.
10. Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable byzantine agreement. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 403–413. IEEE, 2021. doi:10.1109/ICDCS51616.2021.00046.
11. Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, and Jovan Komatovic. As easy as abc: Optimal (a) ccountable (b) yzantine (c) onsensus is easy! *Journal of Parallel and Distributed Computing*, 181:104743, 2023. doi:10.1016/J.JPDC.2023.104743.
12. Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, Zarko Milosevic, and Adi Seredinschi. Crime and punishment in distributed byzantine decision tasks. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pages 34–44. IEEE, 2022. doi:10.1109/ICDCS54860.2022.00013.
13. Luciano Freitas de Souza, Petr Kuznetsov, Thibault Rieutord, and Sara Tucci-Piergiovanni. Accountability and reconfiguration: Self-healing lattice agreement. *arXiv preprint*, 2021. arXiv:2105.04909.
14. Cynthia Dwork, Nancy A. Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988. doi:10.1145/42282.42283.
15. Tiantian Gong, Gustavo Franco Camilo, Kartik Nayak, Andrew Lewis-Pye, and Aniket Kate. Recovery from excessive faults in partially synchronous bft smr, 2025. URL: https://eprint.iacr.org/2025/083.
16. Andrew Lewis-Pye and Tim Roughgarden. How does blockchain security dictate blockchain implementation? In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1006–1019, 2021. doi:10.1145/3460120.3484752.
17. Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus. *arXiv preprint arXiv:2304.14701*, 2023. doi:10.48550/arXiv.2304.14701.

18 Jinyuan Li and David Mazieres. Beyond one-third faulty replicas in byzantine fault tolerant systems. In *NSDI*, pages 10–10, 2007.

19 Atsuki Momose and Ling Ren. Multi-threshold byzantine fault tolerance. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1686–1699, 2021. doi:10.1145/3460120.3484554.

20 Joachim Neu, Ertem Nusret Tas, and David Tse. The availability-accountability dilemma and its resolution via accountability gadgets. In *International Conference on Financial Cryptography and Data Security*, pages 541–559. Springer, 2022. doi:10.1007/978-3-031-18283-9_27.

21 Alejandro Ranchal-Pedrosa and Vincent Gramoli. Zlb: A blockchain to tolerate colluding majorities. *arXiv preprint*, 2020. arXiv:2007.10541.

22 Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *Acm Computing Surveys (CSUR)*, 22(4):299–319, 1990. doi:10.1145/98163.98167.

23 Alex Shamis, Peter Pietzuch, Burcu Canakci, Miguel Castro, Cédric Fournet, Edward Ashton, Amaury Chamayou, Sylvan Clebsch, Antoine Delignat-Lavaud, Matthew Kerner, et al. {IA-CCF}: Individual accountability for permissioned ledgers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 467–491, 2022.

24 Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. Bft protocol forensics. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pages 1722–1743, 2021. doi:10.1145/3460120.3484566.

25 Srivatsan Sridhar, Ertem Nusret Tas, Joachim Neu, Dionysis Zindros, and David Tse. Consensus under adversary majority done right. *arXiv preprint*, 2024. arXiv:2411.01689.

26 Srivatsan Sridhar, Dionysis Zindros, and David Tse. Better safe than sorry: Recovering after adversarial majority. *arXiv preprint*, 2023. arXiv:2310.06338.

27 Ertem Nusret Tas, David Tse, Fangyu Gai, Sreeram Kannan, Mohammad Ali Maddah-Ali, and Fisher Yu. Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 126–145. IEEE, 2023. doi:10.1109/SP46215.2023.10179426.

28 Ertem Nusret Tas, David Tse, Fisher Yu, and Sreeram Kannan. Babylon: Reusing bitcoin mining to enhance proof-of-stake security. *arXiv preprint*, 2022. arXiv:2201.07946.

29 Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019. doi:10.1145/3293611.3331591.

Nakamoto Consensus from Multiple Resources

Mirza Ahad Baig
Institute of Science and Technology Austria, Klosterneuburg, Austria

Christoph U. Günther
Institute of Science and Technology Austria, Klosterneuburg, Austria

Krzysztof Pietrzak
Institute of Science and Technology Austria, Klosterneuburg, Austria

Abstract

The blocks in the Bitcoin blockchain "record" the amount of work W that went into creating them through proofs of work. When honest parties control a majority of the work, consensus is achieved by picking the chain with the highest recorded weight. Resources other than work have been considered to secure such longest-chain blockchains. In Chia, blocks record the amount of disk-space S (via a proof of space) and *sequential* computational steps V (through a VDF).

In this paper, we ask what weight functions $\Gamma(S, V, W)$ (that assign a weight to a block as a function of the recorded space, speed, and work) are secure in the sense that whenever the weight of the resources controlled by honest parties is larger than the weight of adversarial parties, the blockchain is secure against private double-spending attacks.

We completely classify such functions in an idealized "continuous" model: $\Gamma(S, V, W)$ is secure against private double-spending attacks if and only if it is homogeneous of degree one in the "timed" resources V and W, i.e., $\alpha\Gamma(S, V, W) = \Gamma(S, \alpha V, \alpha W)$. This includes the Bitcoin rule $\Gamma(S, V, W) = W$ and the Chia rule $\Gamma(S, V, W) = S \cdot V$. In a more realistic model where blocks are created at discrete time-points, one additionally needs some mild assumptions on the dependency on S (basically, the weight should not grow too much if S is slightly increased, say linear as in Chia).

Our classification is more general and allows various instantiations of the same resource. It provides a powerful tool for designing new longest-chain blockchains. E.g., consider combining different PoWs to counter centralization, say the Bitcoin PoW W_1 and a memory-hard PoW W_2. Previous work suggested to use $W_1 + W_2$ as weight. Our results show that using e.g., $\sqrt{W_1} \cdot \sqrt{W_2}$ or $\min\{W_1, W_2\}$ are also secure, and we argue that in practice these are much better choices.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Nakamoto Consensus, Heaviest-chain Rule, Resource Theory

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.16

Related Version *Full Version*: https://arxiv.org/abs/2508.01448 [5]
Full Version: https://eprint.iacr.org/2025/1410 [6]

Funding This research was funded in whole or in part by the Austrian Science Fund (FWF) 10.55776/F85. For open access purposes, the author has applied a CC BY public copyright license to any author-accepted manuscript version arising from this submission.

1 Introduction

Achieving consensus in a permissionless setting is a famously difficult problem. Nakamoto solved it by introducing the Bitcoin blockchain [31] that achieves consensus on a chain of blocks by having parties expend a *resource*: parallelizable computation (commonly called *work*).

In Bitcoin, appending a block to a chain requires a *proof-of-work* (PoW), i.e., solving a computationally-expensive puzzle. This puzzle is designed such that each block represents the (expected) amount of computation that was expended to append it. As a consequence,

© Mirza Ahad Baig, Christoph U. Günther, and Krzysztof Pietrzak;
licensed under Creative Commons License CC-BY 4.0

each chain represents the total amount of computation required to create it. This allows for a simple consensus mechanism commonly called the *longest-chain rule*: Given two different chains, pick the one that required more computation to create. Note that a more accurate term is *heaviest-chain rule*, which we will use interchangeably throughout the paper.

While this design achieves consensus, more importantly it also achieves a property called *persistence* [19] under a simple economic assumption: As long as honest parties control more than half of the computational resources committed to Bitcoin, a block that has been part of the chain for some time will always be part of the chain. Since Bitcoin blocks contain transactions, this effectively means that an adversary cannot *double-spend* a coin.

While Bitcoin's design is simple, its reliance on PoW has its flaws. For example, it wastes a lot of energy, and the manufacturing of the PoW hardware has become increasingly centralized. Amongst other reasons, this has lead to the development of other blockchain protocols. These protocols can be broadly categorized along three axes.

The Underlying Resources Bitcoin relies on parallelizable computation, which is a *physical resource*. Two natural alternatives are *disk space* and *sequential computation*. A different class of resources is not physical, but *on-chain* [2, 30]. The most well-known is *stake*, which comprises different approaches that essentially rely on the on-chain coin balance of a party.

The Consensus Design A broad distinction is between Byzantine-fault-tolerant-style (e.g., Algorand [24] or Filecoin [17]) and longest-chain protocols. Within themselves, longest-chain protocols differ in their *fork-choice rule*, which prescribes how they select the longest chain. Some are *Nakamoto-like* and, like Bitcoin, pick the heaviest chain, i.e., the one whose blocks cumulatively required the most resources to create (e.g., Chia [10]). Others rely on more complex fork-choice rules that, e.g., take into account where two chains fork (e.g., Ouroboros [3]).

The Degree of Permissionlessness Roughgarden and Lewis-Pye [30] observe that "permissionless" is colloquially used to describe different settings that vary in how permissionless they are. *Fully-permissionless* protocols function obliviously to current protocol participants (e.g., Bitcoin or Chia). These differ from protocols that require some information about participants (e.g., how many coins they are staking in Algorand, or commitments to disk space in Filecoin). While the latter are still permissionless in the sense that anyone can participate, they impose stricter requirements on participants.

1.1 Our Contributions

In this paper, we completely characterize the design space of *Nakamoto-like* protocols operating in the *fully-permissionless* setting using the *physical resources* disk space, sequential computation, and parallelizable computation that are secure against *private double-spending* attacks.

We observe that Nakamoto-like protocols only differ in what resources their blocks record, and – especially if multiple resources are used – how they decide which of two blocks required *more resources* to create. We model these differences using an abstraction called the *weight function* $\Gamma\colon \mathbb{R}^3 \to \mathbb{R}$. It takes as input the three resources possibly recorded by a block, i.e., disk space S, sequential computation V[1] and parallelizable computation/work W, and

[1] Think V as in velocity of the sequential computation or V as in verifiable delay function (VDF), the cryptographic primitive usually used to capture the number of sequential computation steps.

outputs the *weight* $\Gamma(S, V, W)$ of a block. In the context of weight functions, the heaviest-chain rule now picks the chain with the highest weight where a chain's weight is defined as the sum of the weight of all its blocks.

To get an intuition for the weight function abstraction, let us provide some examples. The weight function $\Gamma_{\text{Bitcoin}}(S, V, W) = W$ describes Bitcoin (or any other similar PoW-based Nakamoto-like chain, e.g., Litecoin[2]). More interesting is Chia [10], a Nakamoto-like chain combining disk space and sequential computation following $\Gamma_{\text{Chia}}(S, V, W) = S \cdot V$.

Our main result, informally stated in Theorem 1.1 below, fully characterizes which weight functions result in a Nakamoto-like blockchain that is secure against private double-spending attacks [31, 14] in the fully-permissionless setting [30].

In this work we address double spending, but not economic attacks such as selfish mining [16]. Such attacks are an orthogonal issue and require a different set of tools. Preventing double spending gives some additional guarantees, like the fact that one can trust the timestamps on the chain [40].

To achieve a broad and simple characterization, we necessarily have to abstract implementation details and generalize over different blockchain designs. We operate under the maxim that a good design principle is that chains should reflect the resources that went into creating them. In practical instantiation of a blockchain this may not be fully guaranteed. Network delay and limited block space lead to resources only being approximated by blocks.

Network delays have been well-studied for single resource blockchains like Bitcoin, Chia and Ethereum [14, 22, 25, 21] and they add a multiplicative factor $\chi(\Delta) < 1$ (Δ is the network delay) to the honest resource in the honest majority assumption, where χ depends on the particular blockchain. Thus taking network delays into account would only quantitatively affect our honest majority assumption and not give any new interesting insights. As for resources being accurately reflected on chain: this also depends on the precise implementation of the chain. There are multiple options: either take an average over multiple blocks (akin to how Bitcoin difficulty changes), or put multiple proofs into one block/epoch (for example, including the top k partial PoW solutions in a block, or like in Chia where multiple PoSpace blocks come from the same challenge). This leads to a good approximation of the total resources available at any point of time. To compensate for any loss we again need to include a multiplicative factor to the honest resources in the honest majority assumption. The precise formula would depend on the exact implementation details. Since our focus is on a unified idealized model, we abstract away these details and leave the question of best practical design and trade-offs involved in it as future work. There are other attacks like grinding and double dipping[3] against chains that use space, but we have techniques to prevent them [33, 4]. Thus we assume they are implicitly taken care of in the design.

One issue are so called replotting attacks. As we'll discuss in §4.3, in practice replotting can be prevented with a careful design putting bounds on the total weight of individual blocks. Since replotting is not as well understood as the other issues, we will explicitly exclude replotting in the statement of the theorem below. We'll discuss our model in more detail in §1.3.

▶ **Theorem 1.1** (Main, Informal). *In the fully-permissionless [30] setting and ignoring replotting attacks, a Nakamoto-like blockchain is secure against private double-spending attacks under the honest majority assumption (cf. below)* if and only if *the weight function* $\Gamma(S, V, W)$ *fulfills the following conditions:*

[2] `https://litecoin.org/`

[3] A high level overview on these attacks can be found on `https://docs.chia.net/longest-chain-protocols/`

1. ***Monotone:*** Γ *is monotonically increasing*
2. ***Homogeneous in*** V, W***:*** $\alpha\Gamma(S, V, W) = \Gamma(S, \alpha V, \alpha W)$ *for* $\alpha > 0$

The honest majority assumption states that at any point in time during the attack Γ *applied to the resources of the honest parties is larger than* Γ *applied to the resources of the adversary.*

Thm. 1.1 is in an ideal model. In § 4 we provide a less idealized, discrete model and prove a related result in this model which essentially states that every weight function that is insecure in ideal model is also insecure in the more realistic discrete model. On the other hand every weight function secure in the ideal model is secure in the discrete model with a slightly stronger honest majority condition.

Finally, we deal with the replotting attacks and how to mitigate them in § 4.3.

1.2 Implications of our Result

1.2.1 Space-based Blockchains

Three very different blockchain designs whose main resource is disk space are Chia [10], Filecoin [17], and Spacemint [33]. The first two are deployed and running in practice, while the latter is an academic proposal.

Of the above, Chia is the only one captured by our result, i.e., it is a Nakamoto-like protocol in the fully-permissionless setting. Its weight function is $\Gamma_{\text{Chia}}(S, V, W) = S \cdot V$ which is secure against double-spending attacks by our Thm. 1.1.

Spacemint was an early proposal of a fully-permissionless blockchain based solely on proofs of space, and thus cannot be secure against double-spending attacks according to Thm. 1.1. The design of Spacemint slightly defers from Nakamoto's chain-selection rule as older blocks are given less weight than more recent ones, but even with this twist the security of Spacemint against double-spending only holds if the honest space never decreases by too much, and never increases too fast.

In contrast, Filecoin is not captured by our result because it is not fully-permissionless, but instead operates in the stronger quasi-permissionless setting (cf. [30]).[4] Since Filecoin's weight function is $\Gamma_{\text{Filecoin}}(S, V, W) = S$, Thm. 1.1 essentially shows that a setting stronger than the fully-permissionless one is necessary.

Let us stress that, even in the fully-permissionless setting and when only relying on space, we only rule out secure constructions of Nakamoto-like blockchains (where the weight of a chain is the sum of the weights of its blocks, and the chain selection rule picks the heaviest chain). While most fully-permissionless blockchains are of this form, this does not rule out the possibility that a completely different chain selection rule would be secure. A recent work [7] shows that, unfortunately, this is not the case, and no such chain-selection rule exists. It gives a concrete attack against any chain-selection rule, and an almost matching lower-bound, i.e., a concrete (albeit very strange) chain-selection rule for which this attack is basically optimal.

[4] Filecoin is also not Nakamoto-like since it is a DAG-based protocol (using GHOST, the *Greediest Heaviest-Observed Sub-Tree* rule [38]) together with a finality gadget [1]. Note that the finality gadget is not essential, and GHOST is the DAG-analogue to Bitcoin's longest/heaviest-chain rule. So, for our purposes, Filecoin could easily be modified to be Nakamoto-like (this has also been mentioned in [2]). As we'll elaborate in §4.3, it seems running a space based chain in the quasi-permissionless setting is quite expensive as to prevent reploting parties must constantly prove they hold the committed space and this proofs need to be recorded on chain.

1.2.2 Combining multiple PoWs

Since the production of Bitcoin mining-hardware has become increasingly centralized, one might consider combining two different PoWs, e.g., W_1 using SHA256 and W_2 using Argon2. As stated above, Thm. 1.1 only considers one parallel work W, but it naturally extends to multiple resources of each type. In particular, our results capture weight functions such as $\Gamma(W_1, \ldots, W_k)$ with Condition 2 being $\alpha\Gamma(W_1, \ldots, W_k) = \Gamma(\alpha W_1, \ldots, \alpha W_k)$ for $\alpha > 0$.

Prior work [18] suggested the weight function $\Gamma(W_1, \ldots, W_k) = \sum_{i=1}^{k} \omega_i W_i$ with constants ω_i. By Thm. 1.1, this is secure against private double-spending, but not a desirable weight function in practice. Even though the constants ω_i can be used to calibrate the contribution of each PoW, it seems difficult to realize this in a way that would prevent miners to ultimately only invest in the cheapest PoW.

Our result show that more interesting combinations of $W_1, \ldots, W_k$ are possible. The first draws inspiration from automated-market-makers[5] and is defined as

$$\Gamma(W_1, \ldots, W_k) = \Pi_{i=1}^{k} W_i^{1/k}.$$

To maximize this weight function (for a given budget), one would have to invest into mining hardware for all PoWs at a similar rate.

Another option is the Leontief utilities function[6]

$$\Gamma(W_1, \ldots, W_k) = \min\{W_1, \ldots, W_k\}$$

which ensures that all PoWs must significantly contribute.

Our work just classifies the weight functions that are secure in the sense that we get security against private double-spending whenever the honest parties control resources of higher weight (as specified by the weight function). A question that is mostly orthogonal to this work is to investigate which of those weight functions are also interesting from a practical perspective, say because they incentivize decentralization or other desirable properties. Let us observe that the class of secure weight functions does contain functions that make little sense in practice, for example the function $\Gamma(V) = V$ which simply counts the number of VDF steps. A blockchain based on this weight function would be secure assuming some honest party holds a VDF that is faster than the VDF held by the adversary.

1.3 Model and Modelling Rationale

1.3.1 Modelling Resources

Our model captures resources that are *external* to the chain, i.e., *physical resources*. In particular, we consider disk space S, sequential computation V, and parallelizable computation W where we allow multiple resources per type, e.g., W_1 and W_2.[7] Each resource is modelled as a function mapping time $\mathbb{R}_{\geq 0}$ to an amount $\mathbb{R}_{>0}$. This is expressive enough to capture, e.g., Bitcoin, any other PoW-based blockchain, or Chia.

In practice, cryptographic primitives are used to track these resources, usually *Proof-of-Space* (PoSpace) [15], *Verifiable Delay Functions* (VDFs) [9, 41, 36], and *Proof-of-Work* (PoW). Our modelling essentially assumes a perfect primitive, glossing over implementation details and any probabilistic nature of the resource (similar to [39]).

[5] `https://en.wikipedia.org/wiki/Constant_function_market_maker`
[6] `https://en.wikipedia.org/wiki/Leontief_utilities`
[7] These are three fundamental resources in computation, and also the most popular physical resources used for blockchains. Nevertheless, we believe our model could be extended to other external resources.

In practice parallel work W as captured by a PoW, and sequential work V as captured by a VDF are very different. W is a quantitative resource in the sense that one can double it by investing twice as much, while V is a qualitative resource as it measures the speed of the fastest available VDF. From the perspective of our Theorem on the other hand, W and V behave the same, the only thing that matters in the (proof of the) Theorem is that W and V are "timed" resources in the sense that their unit is something "per second". W and S on the other hand are both quantitative resources, but behave very differently.

1.3.2 Reasons for Omitting Stake

First, as described by Roughgarden and Lewis-Pye [30], stake-based blockchains do not operate in the fully-permissionless setting. Therefore, since our result targets this setting, modelling stake is not possible.

Another reason is that in our modeling we assume that parties hold some resources at some given time, for an on-chain resource like stake this is not well defined as the resource is only defined with respect to some particular chain, i.e., for different forks the parties would hold different resources at the same time. To make issues even more tricky, with stake it's possible to obtain old keys that no longer hold any value, but still can be used in an attack [2].

1.3.3 The Continuous Chain Model

Towards Thm. 1.1, we will first consider the *Continuous Chain Model*. While it is a very abstract model, it is rich enough to already yield Conditions 1 and 2. In a nutshell, we assume that a chain *continuously* and *exactly* reflects the resources that were expended to create it.

Assume that the honest parties at time t hold resources $S(t), W(t), V(t)$, then the chain they can create in a time window $[t_0, t_1]$ will have weight $\int_{t_0}^{t_1} \Gamma(S(t), V(t), W(t))\, dt$.

This continuous model avoids some issues of actual blockchains, like their probabilistic nature or network delays. For example in Bitcoin, a so called 51% attack can actually be conducted with less, say, 41% of the hashing power if network delays are sufficiently large. Moreover, as a Bitcoin block is found every 10 minutes *in expectation*, it frequently happens that no block is found in an hour at all. For this reason a block is only considered confirmed if it's sufficiently deep in the chain. These factors only have a *quantitative* impact on the concrete security threshold of longest-chain blockchains and are well understood [20, 14]. The goal of this paper, however, is *qualitative* in nature. That is, we want to describe which weight functions are secure against private double-spending attacks as long as honest parties have sufficiently more resources than the adversary. Precisely quantifying how much security is lost due to the fact that resources are only approximately recorded, due to network delays or other aspects like double dipping attacks is not our goal.

So far we considered a strongly idealized setting where the blockchain recorded the available resources *continuously* and *exactly*, both can not be met in a practical blockchain[8] where the quantitative resource S or W is distributed over an a priori unlimited number of miners, but for practical reasons we only want a bounded number to actually give input to a block. In Bitcoin and Chia, it is just a single miner that finds a proof that passes some difficulty, and the frequency at which such proofs are found gives an indication of the total

[8] At least not if they use a quantitative resource S or W, which only leaves V, but speed alone will not make a good chain.

resource. We can get a good approximation of the resources by waiting for sufficiently many blocks or using a design where multiple miners contribute to a block, say we record some $k > 1$ best proofs found since the last block in every block. In this work, we will not further deal with the fact that resources are not *exactly* recorded as it is not very informative for the ideal perspective we are taking. In actual constructions like Bitcoin one deals with this by requiring some time before considering blocks as confirmed. The number of blocks to wait is computed using a tail inequality, it depends on the probability of failure one can accept and on the quantitative gap one assumes between honest and adversarial resources.

1.3.4 Private Double-Spending Attack (PDS)

1.3.4.1 Why We Focus on PDS

We analyze the security of weight functions against a specific attack, the *Private Double-Spending* (PDS) attack. So we do not prove security against arbitrary attacks and cannot rule out that a worse attack than PDS exists.

The works of Dembo et al. [14] and Gaži et al. [20] used different techniques to show that PDS is the worst attack against PoW and Proof-of-Stake-based longest-chain protocols. The former [14] analysis also extends to Chia. Concretely, they show that if an adversary has sufficient resources to perform some attack against one of these blockchains, they could also perform a private double-spending attack instead. This verifies the intuition of Nakamoto, who only considered the double-spending attack when arguing about Bitcoin's security [31].

The above results [14, 20] are not general enough to imply the same (i.e., that only consider PDS attacks is sufficient) in our more general setting. For example, their analyses do not capture a blockchain design relying on two different PoWs – a design which our model allows. Nevertheless, these works give evidence that focusing on PDS attacks is enough, and we believe that the results of [14] should generalize to our setting. We leave proving or refuting this intuition to future work.

Note that our intuition is based on the technical details of Dembo et al.'s analysis [14]. It relies on a connection between PDS and any general attack strategy. That is, one "can view *any* attack as a race between adversary and honest chains, not just the private attack. However, unlike the private attack, a general attack may send many adversary chains to simultaneously race with the honest chain." [14, p. 2].

1.3.4.2 Explanation of PDS

In a PDS attack, the adversary forks the chain at some point in time, privately extends its own fork (while honest parties continue to extend the main chain), and releases its fork later on. The attack is successful if the adversary's fork is at least as heavy as the honest chain since this would allow the adversary to double-spend a transaction.

We let the adversary choose the resources available to honest parties and itself during this time. The only condition is that we disallow the adversary to trivially perform a successful attack. That is, at every point in time the adversary resources have at most as much weight as the resources of honest parties, and, to avoid a draw, in some interval strictly less.

The honest chain directly corresponds to the resources of the honest parties. The adversary, however, may cheat since it is mining in private. In particular, it can pretend to have created the chain in a shorter or longer amount of time by stretching/squeezing time. This time manipulation affects the resources recorded on the chain. For example, consider W as the hash rate, then a chain records the total number of hashes. If the adversary now pretends to have created this chain in $1/2$ time, then its hashrate must be $2 \cdot W$ since the total amount of hashes does not change.

Given such an attack, e.g., the weight function $\Gamma(W) = W^2$ is insecure. Indeed, consider an adversary with resource $W(t) = 1$ and honest parties with $W(t) = 2$. Honest parties mining for 1 time create a chain of weight $1 \cdot 2^2 = 4$. In the same timespan, the adversary creates a chain of weight $1 \cdot 1^2 = 1$. However, if it pretends to have mined this chain privately in $1/8$ time, then its chain records the weight $1/8 \cdot (8 \cdot 1)^2 = 8$ instead, beating the honest chain.

We defer more precise definitions and figures exemplifying this time manipulation to § 3.

1.3.5 The Discrete Chain Model

So far, we discussed an abstract model where the chain *continuously* and *exactly* records resource expenditure.

In §4 we discuss a model closer to a real blockchain, where blocks arrive in discrete time slots. We still assume the block exactly records the resources W and V. In particular, a block produced during some timespan $[a, b)$ records $W_{\blacksquare} = \int_a^b W(t)\,dt$ and $V_{\blacksquare} = \int_a^b V(t)\,dt$. For the space we assume that the block records $S(t)$ at some point $a \leq t < b$. The reason for this difference is that W and V are resources that are measured *per second* (e.g., hashes/s or steps/s), so integration over time is well-defined. One the other hand, a proof of space gives a snapshot of the space $S(t)$ available at some point t during block creation. To be on the safe side, we simply assume that the adversary can choose the time t where its space was maximal, while for the honest parties we assume t is the time when $S(t)$ was minimal.

We show (Theorem 4.9) that the classification of secure weight functions basically carries over to this discrete setting as long as the resources don't vary by too much within the block arrival time. But our main motivation to consider the discrete model is to discuss the issue of replotting attacks in §4.3, which only make sense in a discrete setting.

1.4 Future Work

Our work opens multiple new questions for future work. We already mentioned identifying weight functions that are not only secure, but also interesting at the end of section §1.2. At the end of section §4.3 we will discuss an open question concerning replotting attacks. Some other open questions include:

First, modelling on-chain resources, most notably, stake. While stake somewhat behaves like disk space[9], it is different and difficult to model since it is an on-chain resource. For example, one modelling challenge is capturing long range attacks in which parties sell old keys that controlled a lot of stake at some point. This is similar to a bootstraping attack for disk space, but the difference is that the adversary can perform this attack for free (after having bought the keys).

Second, considering chain-selection rules other than the heaviest-chain rule. For example, in the stake setting, Ourboros Genesis [3] operating in dynamically available setting achieves security against PDS using a different chain selection rule.

Third, considering different degrees of permissionless, such as the dynamically-available or quasi-permissionless setting described by [30]. While our results rule out solely using disk space in the fully-permissionless setting, this impossibility does not carry over to other models. For example, Filecoin [17] only uses disk space, but is secure against PDS because it operates in the quasi-permissionless model.

[9] There exist proposals similar to Chia that use stake instead of disk space, i.e., where the weight is Stake $\cdot V$ [13].

1.5 Related Work

1.5.1 Abstract Resource Models

Recently, Roughgarden and Lewis-Pye [30] (an updated version of [29]) presented many (im)possibility results about permissionless consensus. They consider a resource-restricted adversary where resources can be external or on-chain resources. External resources are modelled by so-called *permitter oracles*, whose outputs depend on the amount of resources the querying party has at the time of the query. An important part of their work is a classification of the permissionlessness of consensus protocols:

- *Fully-permissionless* protocols are oblivious to its participants (e.g., Bitcoin).
- *Dynamically-available* protocols know a dynamic list of parties, which may be a function of the past protocol execution (e.g., parties who staked coins), the participants are a subset of this list, and *at least one* honest member of this list participates.
- *Quasi-permissionless* protocols are similar to dynamically-available protocols, but make the stronger requirement that *all* honest members of the list participate. Note that such protocols differ from *permissioned* ones, which also have a list of parties, but where the list *cannot* depend on the past protocol execution.

For example, a result of theirs states that no deterministic protocol solves Byzantine agreement in the fully-permissionless setting, even with resource restrictions.

Two preceding works modelling abstract resources are Terner [39] and Azouvi et al. [2]. Terner [39] considers an abstract resource that essentially is a black-box governing participant selection. They give a consensus protocol that can be instantiated with any such resource satisfying certain properties (e.g., resource generation must be rate-limited relative to the maximum message delay). Both [39, 2] consider only a single resource and not combination of multiple resources.

Azouvi et al. [2] use the abstraction of resource allocators (similar to permitter oracles in [30]) to build a total-ordered broadcast protocol. They describe the properties a resource allocator must fulfill (e.g., honest majority), and construct resource allocators for the resources stake, space[10], and work. As part of this, they classify resources as external vs. on-chain (they call it virtual), and burnable vs. reusable (space and stake are reusable whereas work is not) and discuss trade-offs between different types of resources, e.g., on-chain resources are susceptible to long-range attacks. A limitation of [2] is that total resource is *a priori* known and fixed. In our model all resources can vary and are known only when the blocks are created.

1.5.2 Blockchain Designs

We give a selection of well-known permissionless blockchain designs, describing the weight function or – for non-Nakamoto-like protocols – resource (S, V and W as before, and stake St) and degree of permissionlessness (**f**ully-**p**ermissionless, **d**ynamically-**a**vailable, or **q**uasi-**p**ermissionless): Bitcoin [31] (fp, W), Chia [10] (fp, $S \cdot V$), Filecoin [17] (qp, S), Ethereum [42] (da/qp,[11] St), Algorand [24] (qp, St), Ouroboros [27, 12, 3] (da/qp, St), Snow White [11] (da, St).

[10] Their space allocator lies in the quasi-permissionless model, thus not conflicting with our results.
[11] Depending on whether the network is synchronous/partially-synchronous [30].

Multiple combinations of proof-of-stake (PoStake) and PoW, e.g., [28, 8, 18] exist (all either da or qp). [18] is the only *fungible* protocol, i.e., it is secure as long as the adversary controls less than half of all stake and work *cumulatively* (essentially mapping to $\Gamma(St, W) = St+W$). Their protocol handles multiple PoStake and multiple PoW resources, thus capturing $\Gamma(W_1, W_2) = W_1 + W_2$, which we discussed in § 1.2.

[13] combine PoStake with sequential computation to create a dynamically-available protocol.

Ignoring difficulty, Bitcoin assigns unit weight to every block whose hash surpasses a threshold. [26] analyze other functions assigning weight to block hashes. They suggest using a function that grows exponentially, but is capped at a certain value, which depends on the maximum network delay.

1.5.3 Analyses of Blockchain Protocols

Various works analyze specific blockchains, mostly Bitcoin (or similar PoW chains) [19, 34, 35, 37, 20], but also longest-chain protocols in general [14]. These generally give quantitative security thresholds (i.e., what fraction of adversarial resources is tolerable) depending on, e.g., maximum message delay. We again remark that our work has a different aim, namely a qualitative description of weight functions, disregarding precise security thresholds.

2 Preliminaries

Let $[n] = \{1, \ldots, n\}$. Vectors are typeset as bold-face, e.g., $\boldsymbol{x}$. $\mathbb{R}_{>0}$ and $\mathbb{R}_{\geq 0}$ denote the set of positive real numbers excluding and including 0, respectively. Given two tuples $(x_1, \ldots, x_n), (x'_1, \ldots, x'_n) \in \mathbb{R}^n_{>0}$ we say $(x_1, \ldots, x_n) \leq (x'_1, \ldots, x'_n)$ if $x_i \leq x'_i$ for all $i \in [n]$ with equality holding if and only if $x_i = x'_i$ for all $i \in [n]$.

We denote the time by $t \in \mathbb{R}_{\geq 0}$. For $T_0, T_1 \in \mathbb{R}_{\geq 0}$ where $T_0 < T_1$, $[T_0, T_1]$ denotes the time interval starting at T_0 and ending at T_1. The open interval $(T_0, T_1]$ denotes the time interval $[T_0, T_1]$ excluding T_0. $[T_0, T_1)$ is defined analogously.

▶ **Definition 2.1** (Monotonicity). *A function $f\colon \mathbb{R}^n_{>0} \to \mathbb{R}_{>0}$ is monotonically increasing if*

$$(x_1, \ldots, x_n) \leq (x'_1, \ldots, x'_n) \implies f(x_1, \ldots, x_n) \leq f(x'_1, \ldots, x'_n).$$

▶ **Definition 2.2** (Homogeneity). *A function $f\colon \mathbb{R}^n_{>0} \to \mathbb{R}_{>0}$ is homogeneous*[12] *in $x_j, \ldots, x_n$ with $1 \leq j \leq n$ if, for all $(x_1, \ldots, x_n) \in \mathbb{R}^n_{>0}$ and $\alpha > 0$,*

$$f(x_1, \ldots, x_{j-1}, \alpha \cdot x_j, \ldots, \alpha \cdot x_n) = \alpha \cdot f(x_1, \ldots, x_{j-1}, x_j, \ldots, x_n)$$

▶ **Definition 2.3** (Subhomogeneity). *A function $f\colon \mathbb{R}^n_{>0} \to \mathbb{R}_{>0}$ is subhomogeneous in $x_1, \ldots, x_j$ with $1 \leq j \leq n$ if, for all $(x_1, \ldots, x_n) \in \mathbb{R}^n_{>0}$ and $\alpha \geq 1$,*

$$f(\alpha \cdot x_1, \ldots, \alpha \cdot x_j, x_{j+1}, \ldots, x_n) \leq \alpha \cdot f(x_1, \ldots, x_j, x_{j+1}, \ldots, x_n).$$

3 Continuous Chain Model

To characterize which weight functions provide security against private double-spending (PDS) attacks, we will first introduce the *Continuous Chain Model*. It models physical resources, how resources are turned into an idealized blockchain, and PDS attacks. As the name suggests, the continuous model views the blockchain as one continuous object, instead of consisting of multiple discrete blocks.

[12] More precisely, f is a positively homogeneous function of degree 1. However, we will not need homogeneity of higher degree, so we simply call it "homogeneous".

3.1 Modelling Resources

The model captures physical resources, which are external to the chain, and allows for multiple resources per type. The resources are disk space $\boldsymbol{S} \coloneqq (S_1, \ldots, S_{k_1})$, sequential work $\boldsymbol{V} \coloneqq (V_1, \ldots, V_{k_2})$, and parallel work $\boldsymbol{W} \coloneqq (W_1, \ldots, W_{k_3})$.[13] It allows for multiple resources per type. We will omit k_1, k_2, and k_3 unless needed for clarity.

A *resource profile* records the amount of each resource available at any point in time. Time is modelled as a continuous variable $t \in \mathbb{R}_{\geq 0}$, and we restrict our attention to the time interval $[0, T]$ for some $T > 0$. We take each resource to be a function mapping this interval to $\mathbb{R}_{>0}$, e.g., $W_1 \colon [0, T] \to \mathbb{R}_{>0}$.

▶ **Definition 3.1** (Resource Profile). *A resource profile $\mathcal{R}$ is a 3-tuple of tuple of functions*

$$\mathcal{R} \coloneqq (\boldsymbol{S}(t), \boldsymbol{V}(t), \boldsymbol{W}(t))_{[0,T]}$$

where each tuples of functions is composed of functions with domain $t \in [0, T]$ with $T > 0$ and range $\mathbb{R}_{>0}$, and where each function is Lebesgue integrable.

▶ Remark 3.2. The requirement that each resource is non-zero at every point in time is a minor technical condition. Note that it is always fulfilled in practice since interaction with the blockchain requires a general-purpose computer, and even a low-powered one provides a non-zero amount of S, V, and W.

3.2 Idealized Chain

Ideally, a blockchain should record the amount of resources that were expended to create it, and blockchain protocols are generally designed to approximate this as closely as possible. In our idealized model, we assume that a blockchain *continuously* and *exactly* reflects the resources expended to create it.

▶ **Definition 3.3** (Continuous Chain Profile). *A continuous chain profile $\mathcal{CC}$ is a 3-tuple of tuple of functions*

$$\mathcal{CC} \coloneqq (\boldsymbol{S}(t), \boldsymbol{V}(t), \boldsymbol{W}(t))_{[0,T]}$$

where each tuples of functions is composed of functions with domain $t \in [0, T]$ where $T > 0$ and range $\mathbb{R}_{>0}$, and where each function is Lebesgue integrable.

▶ Remark 3.4. Resource and chain profiles are syntactically identical. The difference lies in semantics: A resource profile describes the resources available to a party (or a set of parties). Meanwhile, a chain profile describes the resources that the chain reflects.

▶ Remark 3.5. In practice, blockchains do not *exactly* record the amount of resources, but only approximate them. For example, in Bitcoin, finding blocks is a probabilistic process, so blocks do not record the actual work invested to create them, but only the *expected* amount of work. Additionally, network delays cause miners to waste time (and thereby work) trying to extend an out-of-date block, in the worst case leading to orphaned blocks. In spite of these issues, the ideal model is still meaningful because these issues introduce *quantitative* gaps (e.g., [14, 19, 20]).

[13] These are three fundamental resources in computation and also the most popular physical resources used for blockchains. Nevertheless, we believe our model could be extended to other external resources.

To capture the heaviest-chain rule, the model assigns each chain a weight. To this end, we first introduce the *weight function* Γ, which assigns a weight to a triple of resources. In other words, it assigns a weight to one point in time.

▶ **Definition 3.6** (Weight Function). *A weight function is a non-constant function given by*

$$\Gamma\colon \mathbb{R}_{>0}^{k_1} \times \mathbb{R}_{>0}^{k_2} \times \mathbb{R}_{>0}^{k_3} \to \mathbb{R}_{>0}.$$

▶ Remark 3.7. The requirement that Γ is non-constant is natural in practice. We explicitly require it because such functions would be vacuously secure against PDS. Looking ahead, the security definition only considers adversaries with a resource disadvantage, which is measured using weight. But if the weight is constant, no such adversary exists, so the function would always be secure against PDS.

As said, Γ takes in three resources and outputs the weight of a specific point in time. In the next step, we extend Γ to compute the weight of a whole chain. We denote this function by $\overline{\Gamma}$. It takes a continuous chain profile as input and outputs the weight of it. Overloading notation slightly, we also allow inputting a resource profile to $\overline{\Gamma}$ since it is syntactically identical to a chain profile.

▶ **Definition 3.8** (Weight of a Chain or Resource Profile). *Consider a weight function Γ and a continuous chain $\mathcal{CC} = (\boldsymbol{S}(t), \boldsymbol{V}(t), \boldsymbol{W}(t))_{[0,T]}$ or resource profile $\mathcal{R} = (\boldsymbol{S}'(t), \boldsymbol{V}'(t), \boldsymbol{W}'(t))_{[0,T]}$. The chain weight function $\overline{\Gamma}$ is defined as*

$$\overline{\Gamma}(\mathcal{CC}) \coloneqq \int_0^T \Gamma(\boldsymbol{S}(t), \boldsymbol{V}(t), \boldsymbol{W}(t))\, dt$$

and

$$\overline{\Gamma}(\mathcal{R}) \coloneqq \int_0^T \Gamma(\boldsymbol{S}'(t), \boldsymbol{V}'(t), \boldsymbol{W}'(t))\, dt.$$

3.3 The Private Double-Spending Attack

In a private double-spending (PDS) attack, the adversary forks the chain at some point in time, extends this fork in private, before releasing the private fork to the public. The attack is successful if the adversary's fork is heavier than the honest chain, because the adversarial fork replaces the honest chain, effectively reverting past transactions. We focus on the PDS attack because it is the prototypical attack against blockchains; we refer back to § 1.3.4 for an in-depth discussion.

3.3.1 Modelling the Attack

To model this attack, we first consider the time frame of the attack. The attack starts (i.e., the adversary forks the chain) at time 0, and the adversary publicly publishes its private chain at time T_{end}. So the attack spans the time interval $[0, T_{\mathsf{end}}]$. During this time, the resources available to the honest parties are given by the resource profile $\mathcal{R}^{\mathcal{H}}$, and they use them to build the honest chain profile $\mathcal{CC}^{\mathcal{H}}$ in the following way:

▶ **Definition 3.9** (Honest Chain Profile). *Consider a resource profile $\mathcal{R}^{\mathcal{H}} = (\boldsymbol{S}^{\mathcal{H}}(t), \boldsymbol{V}^{\mathcal{H}}(t), \boldsymbol{W}^{\mathcal{H}}(t))_{[0,T_{\mathsf{end}}]}$ The corresponding honest chain profile is $\mathcal{CC}^{\mathcal{H}} \coloneqq \mathcal{R}^{\mathcal{H}}$.*

That is, the honest chain $\mathcal{CC}^{\mathcal{H}}$ correctly reflects the resources available to honest parties, and also precisely keeps track at which point in the resources were available.[14]

The adversary's resources are $\mathcal{R}^{\mathcal{A}}$, and they use them to build the chain profile $\mathcal{CC}^{\mathcal{A}}$.[15] In contrast to the honest parties, the adversary may deviate from the protocol and cheat. First, they may simply not use some of the resources available to them. Second, and more importantly, since the adversary creates the fork in private, the chain $\mathcal{CC}^{\mathcal{A}}$ may not correctly reflect *at what time* the resources were available. In essence, the adversary can *alter* the time by *stretching* and *squeezing* it. For example, in Bitcoin the adversary may mine a block in 100 minutes but pretend to have mined it within 10 minutes.

We model this time manipulation by a function $\phi(t)$ describing the squeezing/stretching factor at any point in time. At time t, $\phi(t) > 1$ represents squeezing, $\phi(t) < 1$ stretching, and $\phi(t) = 1$ no alteration. Altering the time affects, e.g., the length of the interval $[0, T_{\mathsf{end}}]$. To this end, we introduce the *altered time* function AT (and its inverse AT^{-1})[16] to translate between time before and after squeezing/stretching. For example, $\widetilde{T}_{\mathsf{end}} = \mathsf{AT}(T_{\mathsf{end}})$, resulting in the interval $[0, \widetilde{T}_{\mathsf{end}}]$.[17]

Altering time affects how $\mathcal{CC}^{\mathcal{A}}$, which covers the time interval $[0, \widetilde{T}_{\mathsf{end}}]$, reflects resources. For the resources V and W, altering time cannot change the cumulative amount (e.g., in Bitcoin it cannot change the number of found blocks and thus work performed). Therefore, V and W must be multiplied by ϕ. That is, at altered time $\tilde{t} \in [0, \widetilde{T}_{\mathsf{end}}]$, $\mathcal{CC}^{\mathcal{A}}$ records the resource $\phi(t)V^{\mathcal{A}}(t)$ and $\phi(t)W^{\mathcal{A}}(t)$. The disk space S behaves differently. As long as it is available, it can be reused [2], so it does not accumulate (unlike V and W). As a consequence, altering time just changes when space was available. Thus, at altered time $\tilde{t} \in [0, \widetilde{T}_{\mathsf{end}}]$, $\mathcal{CC}^{\mathcal{A}}$ records $S(\mathsf{AT}^{-1}(\tilde{t}))$.

▶ **Definition 3.10** (Adversarial Chain Profile). *Consider a resource profile* $\mathcal{R}^{\mathcal{A}} = (\boldsymbol{S}^{\mathcal{A}}(t), \boldsymbol{V}^{\mathcal{A}}(t), \boldsymbol{W}^{\mathcal{A}}(t))_{[0,T_{\mathsf{end}}]}$ *and some function* $\phi(t) \colon [0, T_{\mathsf{end}}] \to \mathbb{R}_{>0}$. *Define* $\mathsf{AT}(t) \coloneqq \int_0^t \frac{1}{\phi(u)}\, du$ *and its inverse* $\mathsf{AT}^{-1}(\cdot)$.

Let $\widetilde{T}_{\mathsf{end}} \coloneqq \mathsf{AT}(T_{\mathsf{end}})$. *An adversarial chain profile is any chain profile*

$$\mathcal{CC}^{\mathcal{A}} = (\widetilde{\boldsymbol{S}}^{\mathcal{A}}(\widetilde{t}), \widetilde{\boldsymbol{V}}^{\mathcal{A}}(\widetilde{t}), \widetilde{\boldsymbol{W}}^{\mathcal{A}}(\widetilde{t}))_{[0,\widetilde{T}_{\mathsf{end}}]}$$

where $\widetilde{S}_i^{\mathcal{A}}(\cdot)$, $\widetilde{V}_i^{\mathcal{A}}(\cdot)$ *and* $\widetilde{W}_i^{\mathcal{A}}(\cdot)$ *are Lebesgue integrable, and satisfy*

$$0 < \widetilde{\boldsymbol{S}}^{\mathcal{A}}(\widetilde{t}) \leq \boldsymbol{S}^{\mathcal{A}}(t)$$
$$0 < \widetilde{\boldsymbol{V}}^{\mathcal{A}}(\widetilde{t}) \leq \phi(t) \cdot \boldsymbol{V}^{\mathcal{A}}(t)$$
$$0 < \widetilde{\boldsymbol{W}}^{\mathcal{A}}(\widetilde{t}) \leq \phi(t) \cdot \boldsymbol{W}^{\mathcal{A}}(t)$$

for all $\widetilde{t} \in [0, \widetilde{T}_{\mathsf{end}}]$ *with* $t = \mathsf{AT}^{-1}(\widetilde{t})$.

Let us illustrate the stretching and squeezing from Def. 3.10 by the example of Bitcoin and Chia in Figures 1 and 2.

[14] While this is by construction in our idealized model, timestamps are generally accurate in longest-chain blockchains – even if a not-too-powerful adversary tries to disrupt them [40].
[15] In general, we use the superscripts $^{\mathcal{H}}$ and $^{\mathcal{A}}$ to denote the honest parties and the adversary, respectively.
[16] AT^{-1} exists because $\frac{1}{\phi(u)} > 0$ for all u, so $\int_0^t \frac{1}{\phi(u)}\, du$ is a monotonically increasing function of t with co-domain $[0, \mathsf{AT}(T_{\mathsf{end}})]$.
[17] We use $\widetilde{\ }$ to denote time after squeezing/stretching.

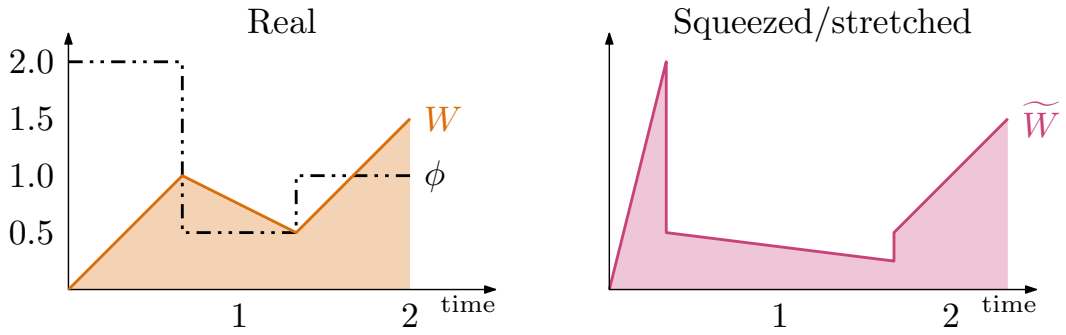


Figure 1 Bitcoin's weight function W and how it reacts to stretching and squeezing. The shaded area is the weight.

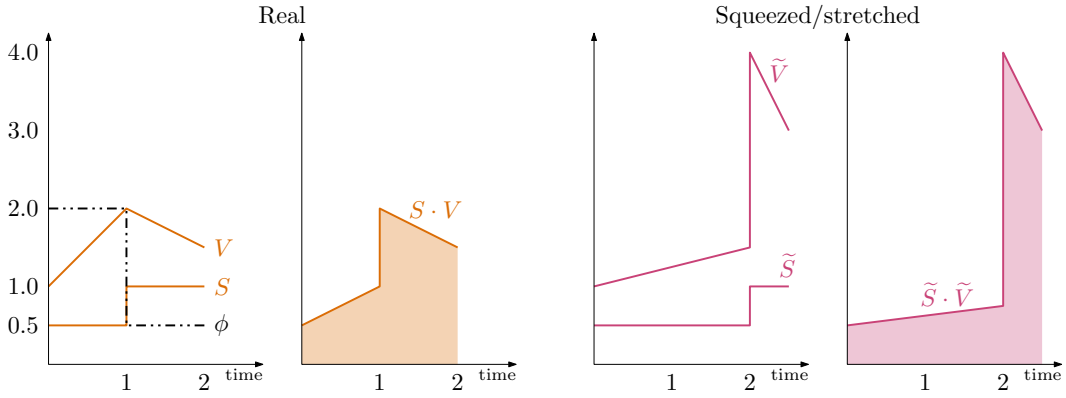


Figure 2 Chia's weight function $S \cdot V$ and how it reacts to stretching and squeezing. The shaded are is the weight.

We now have all ingredients to define when a weight function is secure against PDS attacks. On a high level, the definition states that an adversary having resources of less weight than the honest parties[18] cannot create a private chain that is heavier than the honest parties one – even by manipulating time. In more detail, "less weight" means that the adversary has at most equal weight at every point in time (Equation (1)), and in some interval it has strictly less (Equation (2)).

▶ **Definition 3.11** (Weight Function Security Against PDS, Continuous Model). *A weight function* Γ *is* secure against private double-spending attack *in the* continuous model *if for all* $\mathcal{R}^{\mathcal{H}} = (\boldsymbol{S}^{\mathcal{H}}(t), \boldsymbol{V}^{\mathcal{H}}(t), \boldsymbol{W}^{\mathcal{H}}(t))_{[0,T_{\mathsf{end}}]}$ *and* $\mathcal{R}^{\mathcal{A}} = (\boldsymbol{S}^{\mathcal{A}}(t), \boldsymbol{V}^{\mathcal{A}}(t), \boldsymbol{W}^{\mathcal{A}}(t))_{[0,T_{\mathsf{end}}]}$ *such that*

$$\Gamma(\boldsymbol{S}^{\mathcal{A}}(t), \boldsymbol{V}^{\mathcal{A}}(t), \boldsymbol{W}^{\mathcal{A}}(t)) \leq \Gamma(\boldsymbol{S}^{\mathcal{H}}(t), \boldsymbol{V}^{\mathcal{H}}(t), \boldsymbol{W}^{\mathcal{H}}(t)) \; \forall t \in [0, T_{\mathsf{end}}] \tag{1}$$

and for a time interval $[T_0, T_1]$

$$\Gamma(\boldsymbol{S}^{\mathcal{A}}(t), \boldsymbol{V}^{\mathcal{A}}(t), \boldsymbol{W}^{\mathcal{A}}(t)) < \Gamma(\boldsymbol{S}^{\mathcal{H}}(t), \boldsymbol{V}^{\mathcal{H}}(t), \boldsymbol{W}^{\mathcal{H}}(t)) \; \forall t \in [T_0, T_1] \tag{2}$$

it holds that

$$\overline{\Gamma}(\mathcal{CC}^{\mathcal{H}}) > \overline{\Gamma}(\mathcal{CC}^{\mathcal{A}})$$

where $\mathcal{CC}^{\mathcal{H}} \coloneqq \mathcal{R}^{\mathcal{H}}$ *and* $\mathcal{CC}^{\mathcal{A}}$ *satisfies Def. 3.10 for* $\mathcal{R}^{\mathcal{A}}$ *and any* $\phi(t)$.

[18] Clearly, a PDS attack always works when the adversary has a resource advantage.

▶ Remark 3.12. An alternative to the precondition (Equations (1) and (2)) on resource profiles in Def. 3.11 is that adversarial resources must be strictly smaller than the honest ones at every point in time (instead of just for an interval). Looking ahead, Thm. 3.13 would be true in the if-direction (monotonically increasing and homogeneous implies secure against PDS), but not in the only-if direction. The reason is that not every non-homogeneous function can be attacked (e.g., a function that is $S \cdot \max(V, W)$ when each resource is below some constant threshold c and that is constant c^2 after that). If we additionally put the natural constraint that a weight function Γ is not eventually constant (i.e., for any point $(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{w})$ there exists $(\boldsymbol{s}', \boldsymbol{v}', \boldsymbol{w}')$ such that $(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{w}) < (\boldsymbol{s}', \boldsymbol{v}', \boldsymbol{w}')$ and $\Gamma(\boldsymbol{s}, \boldsymbol{v}, \boldsymbol{w}) < \Gamma(\boldsymbol{s}', \boldsymbol{v}', \boldsymbol{w}')$), then only-if direction also holds (by an adaption of our proof). Either way, the main takeaway is that monotonically increasing and homogeneous functions are the ones secure against PDS, and they are the only ones that should be used to construct Nakamoto-like blockchains using multiple resources.

3.4 Main Theorem in the Continuous Model

There are many possible choices for Γ, but not all are secure against PDS, i.e., a bad choice for a blockchain. For example, Figure 3 shows that $W_1 \cdot W_2$ is insecure, but that $\sqrt{W_1} \cdot \sqrt{W_2}$ seems secure – at least in the example. The following theorem shows that it is secure against PDS in general, because it is monotone (cf. Def. 2.1) and homogeneous in $\boldsymbol{W}$ and $\boldsymbol{V}$ (i.e., $\alpha\Gamma(\boldsymbol{S}, \boldsymbol{V}, \boldsymbol{W}) = \Gamma(\boldsymbol{S}, \alpha\boldsymbol{V}, \alpha\boldsymbol{W})$, cf. Def. 2.2). These are sufficient, but also necessary conditions for security against PDS in the continuous chain model.

▶ **Theorem 3.13** (Secure Weight Functions, Continuous Model)**.** *A weight function Γ is* secure against private double-spending attacks *in the* continuous model *if and only if $\Gamma(\boldsymbol{S}, \boldsymbol{V}, \boldsymbol{W})$ is monotonically increasing (Def. 2.1) and homogeneous in $\boldsymbol{V}, \boldsymbol{W}$ (Def. 2.2).*

We omit the proof due to space constraints. Please refer to the full version of the paper [5, Appendix A].

4 Discrete Chain Model

The continuous chain model is rather abstract, so we also consider a discretized version using blocks. A block reflects the *total* amount of resources that were expended to create it. Honest users create blocks according to some prescribed rule, e.g., in fixed time intervals, but the adversary may not adhere to this rule.

Like in the continuous model, we will describe which weight function Γ leads to a discrete blockchain that is secure against PDS. Compared to the continuous model the security statement introduces quantitative factors. The reason is that resources can fluctuate within a block. The quantitative parameters essentially state: The higher the magnitude of fluctuations within blocks, the larger the resource disadvantage of the adversary must be.

In principle, this statement also needs to quantitatively depend on how Γ depends on $\boldsymbol{S}$. The reason is that in our modelling a block reflects $\boldsymbol{S}$ available *at some point in time* during the block creation. Since $\boldsymbol{S}$ can fluctuate within a block, we pessimistically assume that the adversary always gets the maximum and the honest parties only the minimum. So, e.g., the function $S^2 \cdot V$ requires a larger disadvantage than $S \cdot V$. To not carry around another parameter, we restrict our attention to natural choices for Γ, namely, Γ that are subhomogeneous in $\boldsymbol{S}$ (cf. Def. 2.3) – think of this as "at most linear in $\boldsymbol{S}$".

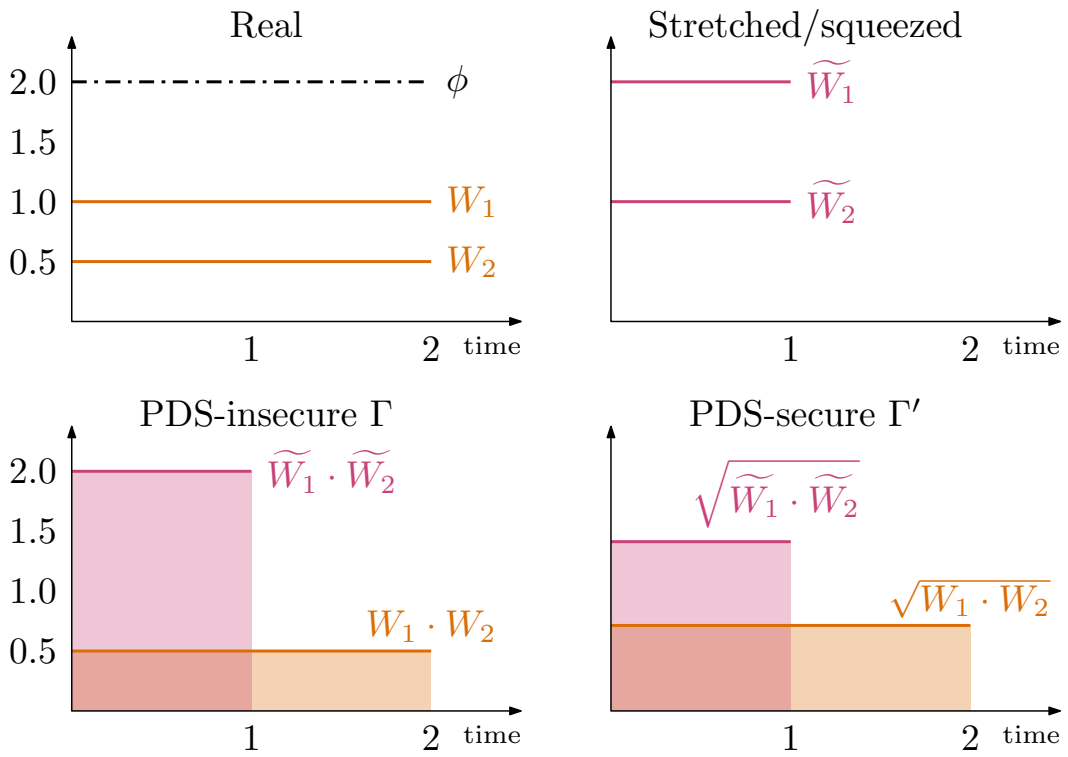


Figure 3 Consider two PoWs W_1, W_2, and two weight functions $\Gamma(W_1, W_2) = W_1 \cdot W_2$ and $\Gamma'(W_1, W_2) = \sqrt{W_1 \cdot W_2}$. The top row show the real resources W_1, W_2 (left) and how squeezing them by $\phi(\cdot) = 2$ (left) results in $\widetilde{W_1}, \widetilde{W_2}$ (right). The bottom row shows that Γ is not secure because $\int_0^2 W \cdot V < \int_0^1 \widetilde{W} \cdot \widetilde{V}$, i.e., squeezing increases the weight. In contrast, Γ' is not affected by the squeezing.

4.1 Definitions

We define a blockchain $\mathcal{BC}$ as the discretization of a resource profile $\mathcal{R}$. Let us first define a *block*.

▶ **Definition 4.1** (Blocks). *Let $\mathcal{R} = (\boldsymbol{S}(t), \boldsymbol{V}(t), \boldsymbol{W}(t))_{[0,T]}$ be a resource profile. A block b_i is defined by a timespan (t_i, t_i') with $0 \leq t_i < t_i' \leq T$. The resources reflected by the block are denoted by $\boldsymbol{S}_{\blacksquare}(b_i)$, $\boldsymbol{V}_{\blacksquare}(b_i)$, and $\boldsymbol{W}_{\blacksquare}(b_i)$.*

Timed resources $\boldsymbol{V}_{\blacksquare}$ and $\boldsymbol{W}_{\blacksquare}$ are reflected by

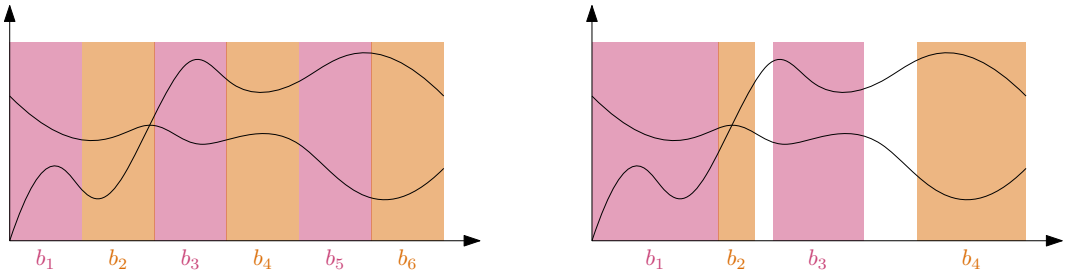
$$\boldsymbol{V}_{\blacksquare}(b_i) = \int_{t_i}^{t_i'} \boldsymbol{V}(t)\, dt \quad \text{and} \quad \boldsymbol{W}_{\blacksquare}(b_i) = \int_{t_i}^{t_i'} \boldsymbol{W}(t)\, dt.$$

The constraint on $\boldsymbol{S}_{\blacksquare}$ is that

$$\inf_{t_i < t < t_i'} \boldsymbol{S}(t) \leq \boldsymbol{S}_{\blacksquare}(b_i) < \sup_{t_i < t < t_i'} \boldsymbol{S}(t). \tag{3}$$

The weight of a block b is $\Gamma(\boldsymbol{S}_{\blacksquare}(b), \boldsymbol{V}_{\blacksquare}(b), \boldsymbol{W}_{\blacksquare}(b))$.

The resources $\boldsymbol{V}$ and $\boldsymbol{W}$ accumulate within a block (e.g., a Bitcoin block reflects the expected number of hashes). $\boldsymbol{S}$ is different ([2] called it "reusable"), so a block can only reflect *some* amount of space that was available within the block's timespan.



(a) Honest parties. **(b)** Adversary (Example).

Figure 4 Discretization of parties. Here, honest parties discretize in fixed time intervals, while the adversary may construct blocks in any non-overlapping fashion.

In the sequel, we will often make use of minima and maxima of resources within a block. For technical reasons, they are defined via infimum and supremum, but think of them as minimum and maximum.

▶ **Definition 4.2** (Minima and Maxima of Resources). *For a resource profile* $\mathcal{R} = (\boldsymbol{S}(t), \boldsymbol{V}(t), \boldsymbol{W}(t))_{[0,T]}$ *we denote the minima/maxima of resources in a block* b *with timespan* (t', t'') *by*

$$\boldsymbol{S}_{\min}(b) = \inf_{t'<t<t''} \boldsymbol{S}(t) \qquad \boldsymbol{S}_{\max}(b) = \sup_{t'<t<t''} \boldsymbol{S}(t)$$
$$\boldsymbol{V}_{\min}(b) = \inf_{t'<t<t''} \boldsymbol{V}(t) \qquad \boldsymbol{V}_{\max}(b) = \sup_{t'<t<t''} \boldsymbol{V}(t)$$
$$\boldsymbol{W}_{\min}(b) = \inf_{t'<t<t''} \boldsymbol{W}(t) \qquad \boldsymbol{W}_{\max}(b) = \sup_{t'<t<t''} \boldsymbol{W}(t)$$

where inf *and* sup *are applied element-wise over the whole vector.*

Now, a blockchain is a sequence of non-overlapping blocks. Its weight $\overline{\Gamma}_{\blacksquare}$ is the sum of the blocks' weights.

▶ **Definition 4.3** (Discrete Blockchain). *A discrete blockchain is a sequence of blocks* $\mathcal{BC} = (b_0, \ldots b_B)$ *whose timespans do not overlap. The weight of a blockchain is*

$$\overline{\Gamma}_{\blacksquare}(\mathcal{BC}) = \sum_{b_i \in \mathcal{BC}} \Gamma(\boldsymbol{S}_{\blacksquare}(b_i), \boldsymbol{V}_{\blacksquare}(b_i), \boldsymbol{W}_{\blacksquare}(b_i)) \tag{4}$$

For honest parties, chain profile and resource profile are identical. Honest parties discretize their resource profile $\mathcal{R}^{\mathcal{H}}$ by following some prescribed rules to create blocks (e.g., in fixed one unit time intervals as depicted in Figure 4a). The resulting blocks are non-overlapping and cover the whole timespan without gaps. The latter requirement is reasonable since honest parties generally do not waste resources. Without loss of generality, we assume the time interval to create a block is 1.

▶ **Definition 4.4** (Honest Discretization). *Let the honest parties' resource profile be* $\mathcal{R}^{\mathcal{H}} = (\boldsymbol{S}^{\mathcal{H}}(t), \boldsymbol{V}^{\mathcal{H}}(t), \boldsymbol{W}^{\mathcal{H}}(t))_{[0,T]}$. *Consider a blockchain* $\mathcal{BC}^{\mathcal{H}} = (b_0^{\mathcal{H}}, \ldots, b_T^{\mathcal{H}})$ *where each block* $b_i^{\mathcal{H}}$ *spans the time* (t_i, t_i'). $\mathcal{BC}^{\mathcal{H}}$ *is an honest blockchain arising from* $\mathcal{R}^{\mathcal{H}}$ *if* $t_0 = 0$, $t_T' = T$, *and* $t_i' = i + 1$ *for all* $i \in [T-1]$.

The adversary also starts from a resource profile $\mathcal{R}^{\mathcal{A}}$, but they may cheat when deriving the blockchain from the resources. In terms of discretization, the adversary may not necessarily follow the prescribed rule. It may create blocks covering varying timespans, or it might leave gaps between blocks. The only condition is that blocks don't overlap (as shown in Figure 4b).

▶ **Definition 4.5** (Adversarial Discretization). *Let the adversary's resource profile* $\mathcal{R}^{\mathcal{A}} = (\boldsymbol{S}^{\mathcal{A}}(t), \boldsymbol{V}^{\mathcal{A}}(t), \boldsymbol{W}^{\mathcal{A}}(t))_{[0,T]}$. *Consider a blockchain* $\mathcal{BC}^{\mathcal{A}} = (b_0^{\mathcal{A}}, \ldots, b_B^{\mathcal{A}})$ *where each block* $b_i^{\mathcal{A}}$ *spans the time* (t_i, t_i'). $\mathcal{BC}^{\mathcal{A}}$ *is an adversarial blockchain arising from* $\mathcal{R}^{\mathcal{A}}$ *if* $0 \leq t_0$, $t_B' \leq T$ *and* $t_i' \leq t_{i+1}$ *for all* $i \in [B-1]$.

Looking ahead, the security of the discrete blockchain quantitatively depends on the maximum fluctuation of resources within blocks. We quantify this fluctuation by the ξ-*Smoothness* of resources. Essentially, $\xi \geq 1$ bounds the absolute change of resources within a block.

▶ **Definition 4.6** (ξ-Smoothness). *Let* $\xi \geq 1$. *A blockchain* $\mathcal{BC}$ *arising from* $\mathcal{R}$ *satisfies* ξ*-smoothness if, for all blocks* $0 \leq i \leq B$, *it holds that*

$$\begin{aligned} \boldsymbol{S}_{\max}(b_i) &\leq \xi \cdot \boldsymbol{S}_{\min}(b_i) \\ \boldsymbol{V}_{\max}(b_i) &\leq \xi \cdot \boldsymbol{V}_{\min}(b_i) \\ \boldsymbol{W}_{\max}(b_i) &\leq \xi \cdot \boldsymbol{W}_{\min}(b_i). \end{aligned}$$

▶ Remark 4.7. In practice, blockchains generally ensure that resources within a block are relatively smooth, i.e, ξ is small. They do so by imposing an upper bound on the resources within a block. For example, Bitcoin's difficulty mechanism essentially puts an upper and lower bound on the work within a block ([26] proposes an alternative way to record work; it has no lower bound, yet still an upper bound). This effectively limits the time span a block takes. Since physical resources are not very elastic – especially at the quantities that are consumed by blockchains – fluctuation is effectively limited.

4.2 Security Statement

Intuitively, the security statement in the continuous model was: If the adversary starts out with fewer resources than the honest parties, then the weight of the adversarial chain is lower than that of the honest one. In the discrete model, the result is a bit weaker because we require a quantitative gap, denoted by $\delta \geq 1$, between honest and adversarial resources. Together with ξ-Smoothness, this leads to the definition of (δ, ξ)-security below.

▶ **Definition 4.8** (Weight Function Security Against PDS, Discrete Model). *A weight function* Γ *is* (δ, ξ)*-secure against private-double spending in the discrete model if, for all honest and adversarial resource profiles* $\mathcal{R}^{\mathcal{H}}$ *and* $\mathcal{R}^{\mathcal{A}}$ *such that*

$$\delta \cdot \Gamma(\boldsymbol{S}^{\mathcal{A}}(t), \boldsymbol{V}^{\mathcal{A}}(t), \boldsymbol{W}^{\mathcal{A}}(t)) < \Gamma(\boldsymbol{S}^{\mathcal{H}}(t), \boldsymbol{V}^{\mathcal{H}}(t), \boldsymbol{W}^{\mathcal{H}}(t)) \; \forall t \in [0, T] \tag{5}$$

the following holds:

For any ξ*-smooth (Def. 4.6) blockchains* $\mathcal{BC}^{\mathcal{H}}$ *and* $\mathcal{BC}^{\mathcal{A}}$, *respectively arising from* $\mathcal{R}^{\mathcal{H}}$ *and* $\mathcal{R}^{\mathcal{A}}$ *according to Defs. 4.4 and 4.5, it holds that*

$$\overline{\Gamma}_{\blacksquare}(\mathcal{BC}^{\mathcal{H}}) > \overline{\Gamma}_{\blacksquare}(\mathcal{BC}^{\mathcal{A}}).$$

We remark that the adversary is more powerful if δ is small (i.e., the gap is small) and ξ is large (i.e., resources may fluctuate by a large magnitude). The following theorem expresses ξ as a function of δ, namely $\xi = \sqrt[4]{\delta}$. This means that if the gap δ is small, then only small fluctuations of resources within blocks may be tolerated.

▶ **Theorem 4.9** (Secure Weight Functions, Discrete Model). *For any* $\delta \geq 1$, *a weight function is* $\Gamma(\boldsymbol{S}, \boldsymbol{V}, \boldsymbol{W})$ *is* $(\delta, \sqrt[4]{\delta})$*-secure against private-double spending (Def. 4.8) if it is*

1. *monotonically increasing;*
2. *homogeneous in* $\boldsymbol{V}$ *and* $\boldsymbol{W}$*; and*
3. *subhomogeneous in* $\boldsymbol{S}$.

We omit the proof due to space constraints. Please refer to the full version of the paper [5, Appendix B].

4.3 Replotting Attacks

In the discrete model, we also have to consider *replotting attacks*. Such attacks were first discussed in the Spacemint [33] paper under the term "space reuse". The Chia green paper [32] discusses them in more detail.

Replotting attacks are easiest understood when one assumes that disk space is bound to some public key of a wallet. Then, in a replotting attack, the adversary repeatedly *replots* (i.e., re-initializes) its space using different keys within the time span of a block. This effectively increases the adversary's space within a block at the cost of extra computation to perform the replotting. Concretely, we assume replotting takes $\rho > 1$ time (usually, blockchains ensure that this ρ large), and an adversary with N space that replots m times appears to have $(m+1) \cdot N$ space.

▶ Remark 4.10. No matter the concrete cryptographic primitive used to track space, such attacks seem unavoidable in the fully-permissionless model. In other settings, e.g., the quasi-permissionless model, replotting can be disincentivized. For example, Filecoin [17] requires parties to commit to space, and then the parties must continuously prove that they are storing the committed space. This prevents replotting if the gap between the required proofs is smaller than the replotting time. In practice, this gap might be too small and hence inefficient, so a more delicate security argument is needed; see [23] for details.

4.3.1 Extra Assumptions are Necessary

Without any extra assumptions, replotting leads to attacks in the discrete model. For the sake of example, consider Chia's weight function $S \cdot V$, which is secure according to Thm. 4.9. Assume replotting takes time $\rho = 2$, $S^{\mathcal{A}} = V^{\mathcal{A}} = 1$ and $S^{\mathcal{H}} = V^{\mathcal{H}} = 1.1$ (this gap suffices since both profiles are 1-smooth), and consider the timespan $[0, 6]$. The honest parties create 5 blocks with a cumulative weight of $6 \cdot (1.1 \cdot 1.1) \approx 7.3$. The adversary creates one block in which it replots once. Assuming that the adversary cannot do anything else while replotting (i.e., it can only gain V for 4 time), the weight of the block is $4 \cdot (2 \cdot 1) = 8$. Note that this attack generalizes to other weight functions Γ and other values of ρ.

4.3.2 A Solution using Difficulty

One way to disincentive replotting in the discrete model is bounding the total weight of a block b. Consider that the protocol keeps track of a difficulty D that is periodically adjusted so that roughly one block is created per time unit (e.g., in Bitcoin the difficulty is reset once every two weeks so blocks arrive roughly every 10 minutes). Further, let $\eta \geq 1$ be a protocol parameter (to be set later). Then, the protocol bounds the weight of blocks as $D \leq \Gamma(b) \leq \eta \cdot D$ (abusing notation of Γ slightly).

If we now set $\eta < \rho$, it is not hard to see that replotting does not help: Replotting even once requires ρ time, and the resulting adversarial block has at most $\eta \cdot D < \rho \cdot D$ weight. Meanwhile, the honest parties produce around ρ blocks, each of weight at least D; so in total $\rho \cdot D$.

Note that this argument requires D to stay fixed, an attacker might still be able to create a heavier chain over a long period of time that spans several epochs (where the difficulty is reset once every epoch).

Chia [32] with its weight function $\Gamma(S, V) = S \cdot V$ uses a similar idea, but it tracks the difficulty of the space and VDF separately. The block arrival time is only determined by the VDF difficulty, which is nice as VDF speed hardly fluctuates at all over time.

One can generalize this idea to any weight function that can be written as $\Gamma(\boldsymbol{S}, \boldsymbol{V}, \boldsymbol{W}) = \Gamma_1(\boldsymbol{S}) \cdot \Gamma_2(\boldsymbol{V}, \boldsymbol{W})$. Now one would require that each block that records resources $\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{s}$ must satisfy $\Gamma_2(\boldsymbol{v}, \boldsymbol{w}) = D$. One doesn't need to put an additional upper bound on the space $\Gamma_1(\boldsymbol{s})$ if Γ_1 is subhomogenous, i.e., for any $\alpha > 1$ we have $\Gamma_1(\alpha \boldsymbol{s}) \leq \alpha \Gamma_1(\boldsymbol{s})$ (Chia does both, it is (sub)homogenous and has an upper bound).

4.3.3 Future Work on Replotting

As mentioned, the above solutions don't formally prevent replotting attacks that range over many epochs. In practice that might not be such a big issue, as extremely long range attacks are not really practical: they require a lot of resources for a long period of time, and thus are expensive to launch. Moreover it might be difficult to convince honest parties to accept a very long fork as it's such an obvious attack. It still would be interesting to understand whether it's possible to formally achieve security against replotting attacks, we leave this for future work.

Financial Conflicts-of-Interest

Krzysztof Pietrzak is a scientific advisor to Chia Network Inc.

References

1 Steven Allen, Masih H. Derkani, Jie Hou, Henrique Moniz, Alex North, Matej Pavlovic, Aayush Rajasekaran, Alejandro Ranchal-Pedrosa, Jorge M. Soares, Jakub Sztandera, Marko Vukolic, and Jennifer Wang. Fast Finality in Filecoin (F3). https://github.com/filecoin-project/FIPs/blob/master/FIPS/fip-0086.md, 2023.

2 Sarah Azouvi, Christian Cachin, Duc V. Le, Marko Vukolić, and Luca Zanolini. Modeling Resources in Permissionless Longest-Chain Total-Order Broadcast. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems (OPODIS 2022)*, volume 253 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:23, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.OPODIS.2022.19.

3 Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 913–930. ACM, 2018. doi:10.1145/3243734.3243848.

4 Vivek Kumar Bagaria, Amir Dembo, Sreeram Kannan, Sewoong Oh, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Proof-of-stake longest chain protocols: Security vs predictability. In Jorge M. Soares, Dawn Song, and Marko Vukolic, editors, *Proceedings of the 2022 ACM Workshop on Developments in Consensus, ConsensusDay 2022, Los Angeles, CA, USA, 7 November 2022*, pages 29–42. ACM, 2022. doi:10.1145/3560829.3563559.

5 Mirza Ahad Baig, Christoph U. Günther, and Krzysztof Pietrzak. Nakamoto consensus from multiple resources, 2025. arXiv:2508.01448.

6 Mirza Ahad Baig, Christoph Ullrich Günther, and Krzysztof Pietrzak. Nakamoto consensus from multiple resources. Cryptology ePrint Archive, Paper 2025/1410, 2025. URL: https://eprint.iacr.org/2025/1410.

7 Mirza Ahad Baig and Krzysztof Pietrzak. On the (in)security of proofs-of-space based longest-chain blockchains. Financial Cryptography and Data Security, 2025.

8 Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]y. *SIGMETRICS Perform. Eval. Rev.*, 42(3):34–37, December 2014. doi:10.1145/2695533.2695545.

9 Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Cham, August 2018. doi:10.1007/978-3-319-96884-1_25.

10 Bram Cohen and Krzysztof Pietrzak. The chia network blockchain, 2019. This is an early proposal and differs significantly from the implemented version [32]. URL: https://docs.chia.net/files/Precursor-ChiaGreenPaper.pdf.

11 Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 23–41. Springer, Cham, February 2019. doi:10.1007/978-3-030-32101-7_2.

12 Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78375-8_3.

13 Soubhik Deb, Sreeram Kannan, and David Tse. PoSAT: Proof-of-work availability and unpredictability, without the work. In Nikita Borisov and Claudia Díaz, editors, *FC 2021, Part II*, volume 12675 of *LNCS*, pages 104–128. Springer, Berlin, Heidelberg, March 2021. doi:10.1007/978-3-662-64331-0_6.

14 Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and nakamoto always wins. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 859–878. ACM Press, November 2020. doi:10.1145/3372297.3417290.

15 Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Berlin, Heidelberg, August 2015. doi:10.1007/978-3-662-48000-7_29.

16 Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454. Springer, Berlin, Heidelberg, March 2014. doi:10.1007/978-3-662-45472-5_28.

17 Filecoin. Filecoin, 2024. URL: https://filecoin.io.

18 Matthias Fitzi, Xuechao Wang, Sreeram Kannan, Aggelos Kiayias, Nikos Leonardos, Pramod Viswanath, and Gerui Wang. Minotaur: Multi-resource blockchain consensus. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pages 1095–1108, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3548606.3559356.

19 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Berlin, Heidelberg, April 2015. doi:10.1007/978-3-662-46803-6_10.

20 Peter Gazi, Aggelos Kiayias, and Alexander Russell. Tight consistency bounds for bitcoin. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 819–838. ACM Press, November 2020. doi:10.1145/3372297.3423365.

21 Peter Gazi, Ling Ren, and Alexander Russell. Practical settlement bounds for proof-of-work blockchains. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1217–1230. ACM Press, November 2022. doi:10.1145/3548606.3559368.

22 Peter Gazi, Ling Ren, and Alexander Russell. Practical settlement bounds for longest-chain consensus. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 107–138. Springer, Cham, August 2023. doi:10.1007/978-3-031-38557-5_4.

23 Irene Giacomelli and Luca Nizzardo. Filecoin proof of useful space – Technical report. URL: https://drive.usercontent.google.com/download?id=1notObdkPT1BCztgspIpzSUAzWSrM8h81.

24 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3132747.3132757.

25 Dongning Guo and Ling Ren. Bitcoin's latency-security analysis made simple. In Maurice Herlihy and Neha Narula, editors, *Proceedings of the 4th ACM Conference on Advances in Financial Technologies, AFT 2022, Cambridge, MA, USA, September 19-21, 2022*, pages 244–253. ACM, 2022. doi:10.1145/3558535.3559791.

26 Simon Holmgaard Kamp, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, Søren Eller Thomsen, and Daniel Tschudi. Weight-based nakamoto-style blockchains. In Patrick Longa and Carla Ràfols, editors, *Progress in Cryptology – LATINCRYPT 2021*, pages 299–319, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-88238-9_15.

27 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Cham, August 2017. doi:10.1007/978-3-319-63688-7_12.

28 Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake, 2012. URL: https://people.cs.georgetown.edu/~clay/classes/fall2017/835/papers/peercoin-paper.pdf.

29 Andrew Lewis-Pye and Tim Roughgarden. Byzantine generals in the permissionless setting. In Foteini Baldimtsi and Christian Cachin, editors, *FC 2023, Part I*, volume 13950 of *LNCS*, pages 21–37. Springer, Cham, May 2023. doi:10.1007/978-3-031-47754-6_2.

30 Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus, 2024. arXiv:2304.14701.

31 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

32 Chia Network. The chia network blockchain, 2019. URL: https://docs.chia.net/green-paper-abstract/.

33 Sunoo Park, Albert Kwon, Georg Fuchsbauer, Peter Gazi, Joël Alwen, and Krzysztof Pietrzak. SpaceMint: A cryptocurrency based on proofs of space. In Sarah Meiklejohn and Kazue Sako, editors, *FC 2018*, volume 10957 of *LNCS*, pages 480–499. Springer, Berlin, Heidelberg, February / March 2018. doi:10.1007/978-3-662-58387-6_26.

34 Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Cham, April / May 2017. doi:10.1007/978-3-319-56614-6_22.

35 Rafael Pass and Elaine Shi. Rethinking large-scale consensus. In Boris Köpf and Steve Chong, editors, *CSF 2017 Computer Security Foundations Symposium*, pages 115–129. IEEE Computer Society Press, 2017. doi:10.1109/CSF.2017.37.

36 Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019. doi:10.4230/LIPIcs.ITCS.2019.60.

37 Ling Ren. Analysis of Nakamoto consensus. Cryptology ePrint Archive, Report 2019/943, 2019. URL: https://eprint.iacr.org/2019/943.

38 Yonatan Sompolinsky and Aviv Zohar. Accelerating Bitcoin's transaction processing. Fast money grows on trees, not chains. Cryptology ePrint Archive, Report 2013/881, 2013. URL: https://eprint.iacr.org/2013/881.

39 Benjamin Terner. Permissionless consensus in the resource model. In Ittay Eyal and Juan Garay, editors, *Financial Cryptography and Data Security*, pages 577–593, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-18283-9_29.

40 Apostolos Tzinas, Srivatsan Sridhar, and Dionysis Zindros. On-chain timestamps are accurate. Cryptology ePrint Archive, Report 2023/1648, 2023. URL: `https://eprint.iacr.org/2023/1648`.

41 Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Cham, May 2019. `doi:10.1007/978-3-030-17659-4_13`.

42 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. EIP-150 Revision.

Fully-Fluctuating Participation in Sleepy Consensus

Yuval Efron
Columbia University, New York, NY, USA

Joachim Neu
a16z Crypto Research, New York, NY, USA

Toniann Pitassi
Columbia University, New York, NY, USA

Abstract

Proof-of-work allows Bitcoin to boast security amidst arbitrary fluctuations in participation of miners throughout time, so long as, at any point in time, a majority of hash power is honest. In recent years, however, the pendulum has shifted in favor of proof-of-stake-based consensus protocols. There, the sleepy model is the most prominent model for handling fluctuating participation of nodes. However, to date, no protocol in the sleepy model rivals Bitcoin in its robustness to drastic fluctuations in participation levels, with state-of-the-art protocols making various restrictive assumptions. In this work, we present a new adversary model, called *external adversary*. Intuitively, in our model, corrupt nodes do not divulge information about their secret keys. In this model, we show that protocols in the sleepy model can meaningfully claim to remain secure against fully fluctuating participation, without compromising efficiency or corruption resilience. Our adversary model is quite natural, and arguably naturally captures the process via which malicious behavior arises in protocols, as opposed to traditional worst-case modeling. On top of which, the model is also theoretically appealing, circumventing a barrier established in a recent work of Malkhi, Momose, and Ren.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Security and privacy → Distributed systems security

Keywords and phrases Sleepy Consensus, fully-fluctuating dynamic Participation

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.17

Acknowledgements We thank Joseph Bonneau, Javier Nieto, and Ling Ren for fruitful discussions.

1 Introduction

Byzantine-fault tolerant (BFT) consensus, the problem of reaching agreement on a value amongst n nodes, of which a fraction are corrupted and controlled by an *adversary*, is one of the most fundamental problems in distributed computing, with research spanning over four decades [19, 26]. The last decade has seen a significant growth in work and interest in the problem, mainly motivated by the pivotal role that consensus protocols play in the design and analysis of blockchains [23, 17, 7, 1, 25, 13].

Traditionally, it is assumed that apart from the corrupt nodes, all other nodes follow the protocol at all times. This assumption, while justified for applications such as data centers, is arguably optimistic in the context of blockchains, where the participants in the protocol may come and go at arbitrary points in time. Specifically, the desiderata for many blockchain protocols, such as Bitcoin and Ethereum, requires maintaining security even in the presence of *many unexpected temporary crash faults*, in which some subset of the nodes may go offline for an undetermined interval of time. In particular, the protocol may experience alternating periods of high participation and low participation.[1] The seminal

[1] Throughout, we assume a fixed set of *potential* participants *known* to the protocol; i.e., the set of nodes allowed to partake is fixed, and fluctuation in participation occurs w.r.t. that set. *Reconfiguration* of the nodes partaking in the protocol is orthogonal to the *unexpected* and *wide* fluctuations in participation considered here, and reconfiguration is expressly *not* considered in this work.

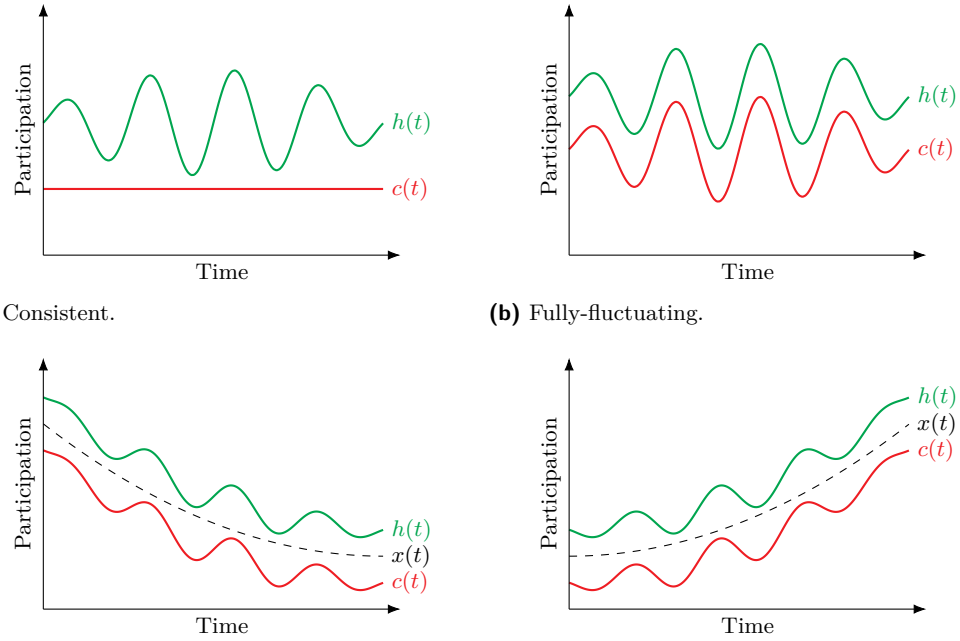
© Yuval Efron, Joachim Neu, and Toniann Pitassi;
licensed under Creative Commons License CC-BY 4.0

Bitcoin protocol [23], also known as proof-of-work Nakamoto consensus, was the first protocol to achieve secure consensus under such circumstances, allowing participants in the protocol to come and go as they please so long as an honest majority (of hash-based mining power) is maintained at any point in time. Its deftly formalized proof-of-work framework allowed for protocol design which is completely agnostic to the total number of participating nodes (corrupt or honest!) *across* present, past, or future. For example, if tomorrow the honest participation in Bitcoin drops to a level below the corrupt participation *today*, it would not render Bitcoin insecure (so long as corrupt participation decreases accordingly). The same holds for if corrupt participation *tomorrow* exceeds honest participation *today* (assuming honest participation increases accordingly).

As the years went on, proof-of-work ceded its spot in the limelight to proof-of-stake, which presently is the dominant approach to Sybil-resistance among blockchain protocols. In the translation, however, resilience to fluctuation in participation was lost. In the context of proof-of-stake, a formal model that captures fluctuating participation is the *sleepy* consensus model, introduced in [24] by Pass and Shi. This model is the focus of our work. In the sleepy model, there is a fixed set of n nodes, identified via a public key infrastructure (PKI). Besides being classified as *honest* or *corrupt*, nodes can alternate (adversarially) between two states at each point in time: *awake*, or *asleep*. Awake nodes can participate in the protocol normally, i.e., perform local computation, send and receive messages. On the other hand, asleep nodes cannot perform any computation and cannot engage in the protocol in any way, until they wake up. While a proof-of-stake instantiation of Nakamoto's protocol was proven secure [24] in the sleepy model, the proof assumes significant restrictions on fluctuating participation. Namely, that honest participation at *any* time exceeds corrupt participation at *all* times. Such a concession perhaps felt inherent, with many follow-up works in the sleepy model making the same assumption [24, 16, 12, 2, 25, 8, 13]. This intuition for the necessity of such an assumption was formalized in [21], which showed that indeed any consensus protocol that does not employ time-based cryptography (e.g., verifiable delay functions [5], henceforth VDF) must heavily restrict fluctuating participation.

The impossibility [21] is established by the following *key transfer* attack. Consider a consensus protocol that makes use of a PKI setup, where it is assumed that prior to the initiation of the protocol, all nodes were privately handed secret-key/public-key pairs. Pondering on the PKI assumption for a moment, however, one observes that the notion of an *asleep corrupt node* makes little sense, as any corrupt node that is awake for even a single point in time, and whose behaviour is completely controlled by the adversary, can simply broadcast its secret key to all other corrupt nodes, and then be put to sleep. Thus allowing essentially a *single* awake corrupt node to simulate a behaviour indistinguishable from all corrupt nodes being awake. Such an attack clearly renders a protocol vulnerable to the case where honest participation decreases over time.

What remains is perhaps an unfortunate state of affairs, in which proof-of-stake protocols seem to inherently carry the burden of rigid participation constraints. The goal of this work is to challenge this status quo. We do so by revisiting the adversary model. Specifically, [21] has already observed that such a strong adversary model capable of executing the key transfer attack may be a tad unrealistic: "The (standard) adversary assumption, where nodes can extract all corrupt nodes' private states, is too strong in practice. In particular in the proof-of-stake protocols, it is highly unlikely that corrupt nodes hand off their secret keys to the adversary (or other corrupt nodes) at the risk of losing their entire stake." [21] They conclude by raising the problem of achieving sleepy consensus under fully-fluctuating corrupt nodes in a "more realistic adversarial model."



(a) Consistent.

(b) Fully-fluctuating.

(c) Decreasing.

(d) Increasing.

Figure 1 Examples of possible participation rates for each of the discussed settings. The graphs depict the number of participating nodes at any point in time, with green curves depicting honest participation, and red curves depicting corrupt participation. Note that in all four settings, honest participation exceeds corrupt participation at any given point in time. On top of that, in the consistent case, note that *any* point on the honest curve dominates the corrupt curve at *all* points in time. Denoting by $h(t)$ and $c(t)$ the honest and corrupt participation at time t, the increasing (decreasing) setting can be characterized by the existence of a monotone increasing (decreasing) function $x(t)$ such that $c(t) < x(t) < h(t)$ holds at all times. The fully-fluctuating case poses no additional restrictions.

Motivated by similar principles, we consider in this work a mildly relaxed adversary model, which we refer to as an *external adversary*. Informally, an external adversary is external to the network and the protocol; the adversary can observe the protocol from a birds eye view. In turn, corrupt nodes are not inherently malicious, but opportunistic: they are recruited by the external adversary to perform arbitrary behavior in the protocol, but won't divulge information about their own secret key to any other node. We believe it is much more likely that a malicious entity would be able to "rent" keys in the protocol, by means of recruiting nodes to engage in Byzantine behavior, rather than convince nodes to give up their secret keys. Stating a formal model that accurately captures the above intuition turns out to be quite simple, but does requires some care, and we tend to it in Sec. 2. Not only does this model still capture a rich plethora of attack vectors pertinent in practice, but it is also theoretically appealing: Disallowing corrupt nodes to share information about their secret keys with one another is precisely the minimal relaxation required to render the key transfer attack infeasible.

The settings. Pondering on the notion of fluctuating participation of nodes in the protocol, one can identify several interesting settings to consider. See also Fig. 1.

1. **Consistent participation.** (Fig. 1a) This is the most common assumption in the context of the sleepy model [24, 16, 12, 2, 25, 8, 13], where it is assumed that new (i.e., after time 0) corrupt nodes may never wake up, and corrupt nodes may never go to sleep. Additionally, an honest majority is assumed to hold at all times. In other words, for *any* point time, honest participation exceeds corrupt participation at *all* times.
2. **Increasing participation.** (Fig. 1d) In this setting, participation of corrupt nodes is guaranteed to be non decreasing as time progresses. In other words, new corrupt nodes may wake up throughout the protocol, but may no go to sleep once woken up. Note that in this setting, even assuming an honest majority at any point in time, it could be that as time goes on, corrupt participation exceeds honest participation in *earlier* points in time. As a matter of fact, this setting has received a fair amount of attention recently [22, 21, 9, 10, 11], in several works showcasing efficient consensus protocols for this setting.
3. **Decaying participation.** (Fig. 1c) In this setting, participation of corrupt nodes is guaranteed to be non increasing as time progresses. In other words, corrupt nodes may go to sleep throughout the protocol, but can never be newly woken up. In this setting, even with an honest majority at any point in time, it could be that as time goes on, corrupt participation in the *past* exceeds *current* honest participation.
4. **Fully-fluctuating.** (Fig. 1b) This is the most general setting in the context of the sleepy model. In this setting, participation of both honest and corrupt nodes can both increase and decrease over time, to the discretion of the adversary. In particular, any of the scenarios mentioned in the previous settings can also occur in this setting.

1.1 Main results

Our results establish that assuming an external adversary allows protocols in the sleepy model to be endowed with resilience to fluctuating participation of nodes, rivaling Nakamoto in its graceful handling of unknown participation patterns.

(1) Sleepy consensus under decaying participation. Our first result establishes the utility of the external adversary model by showcasing a separation from the standard adversary model. Namely, consider the case where participation of both honest and corrupt nodes decreases over time. In particular, for any point in time, *future* honest participation may be eclipsed by *current* corrupt participation. Recent work [21] showed that in this setting, no protocol can be secure in the presence of an adversary that can carry out key transfer.

Our first result shows that against an external adversary, a consensus protocol resilient to decaying participation is not only feasible, but can be made highly efficient and requires only standard cryptographic assumptions.

▶ **Theorem** (Informal, see Thm. 6)**.** *Assuming a PKI, a VRF, and an external adversary, we present a randomized protocol solving consensus in the decaying participation setting with expected constant latency.*

(2) Consensus under fully-fluctuating participation. Thm. 6 establishes the strength of the external adversary model from the lens of protocol design, allowing us to obtain provable security guarantees known to be impossible under the standard notion of an adversary, that is able to carry out the key transfer attack. We now turn our attention to the more general setting of *fully-fluctuating* participation, in which participation of honest and corrupt nodes can both increase and decrease over time, to the discretion of the adversary. Designing a

protocol secure in the fully-fluctuating setting in a relaxed adversary model was explicitly stated as an open problem in [21]. We show that with the help of VDFs, one can design an efficient consensus protocol for the fully-fluctuating participation setting, secure against an external adversary.

▶ **Theorem** (Informal, see Thm. 8)**.** *Assuming a PKI, a VRF, a VDF, and an external adversary, we present a randomized protocol solving consensus that tolerates fully-fluctuating participation with expected constant latency.*

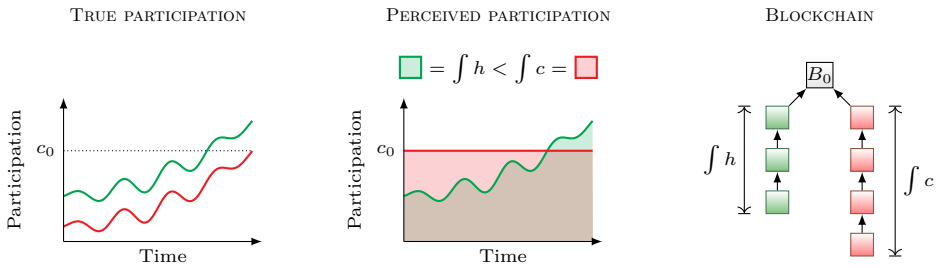
Tab. 1 summarizes our contributions and gives full context of previous work using the formal notation we introduce in Sec. 2. See Sec. 6 for an in-depth overview of related work.

1.2 Key techniques

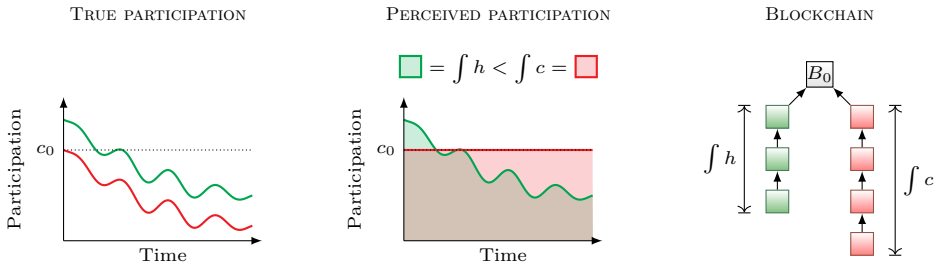
The challenge. Even in non-sleepy synchrony settings [27], any consensus protocol must assume that the adversary is limited in corrupting only a minority of the participants. The sleepy model, being a generalization of the traditional setting, clearly must also abide by an assumption of similar flavour. Note further that in the sleepy model, the awake/asleep status of every node at every point in time is also dictated by the adversary. Thus the changing nature of node participation in the sleepy model makes such an assumption non-trivial to state. Arguably the minimal assumption one could make, is that at each point in time, honest participation exceeds corrupt participation. E.g., this is the assumption for which Nakamoto consensus [23] is proven secure in the proof-of-work setting. In the translation to proof-of-stake, however, such a minimal assumption forces any purported protocol to be resilient against the following two attack vectors. We emphasize that these attacks remain a concern in the external adversary model, and are described to go through within the confines of the model. To envisage the attack vectors in a tangible manner, we consider here as a running example the proof-of-stake version of Nakamoto consensus.

- **Backward simulation.** (Fig. 2a) Consider the following. The adversary commences the protocol at time 0 with low participation of both corrupt and honest nodes. As time goes on, the adversary increases participation of corrupt (and respectively, honest) nodes until at some point in time t, corrupt participation at time t exceeds *honest* participation at timeslot 0 by a sufficiently large amount. At this point, the corrupt nodes, via a costless simulation attack, can construct a private chain whose length exceeds the length of the honest chain constructed thus far. From the point of view of a newly woken up honest node at time t, the longest chain rule will cause it to build on top of the corrupt chain, violating safety. Such an attack is a concern in periods of *increasing participation.*
- **Forward simulation.** (Fig. 2b) Alternatively, consider the following scenario. Participation starts out high, and as time goes on, the adversary puts corrupt (and honest, respectively) nodes to sleep until at some time t, the number of awake corrupt nodes at time 0 exceeds *honest* participation at time t by a sufficiently large amount. Assume further that the leader order is *predictable* from time 0. The corrupt nodes awake at time 0 can simulate an execution from time 0 to time t with a consistent participation rate equal to the number of corrupt nodes *awake at time* 0. From the point of view of a newly woken up honest node at time t, the longest chain rule will cause it to build on top of the corrupt chain, violating safety. Such an attack is a concern in periods of *decreasing participation.*

In this work, we devise several tools to deal with such attack vectors.



(a) Backward simulation attack.



(b) Forward simulation attack.

Figure 2 Examples of backward and forward simulation attacks. (b) Consider the presented *decaying* participation pattern amongst nodes. Let $c_0 = c(0)$, and let t be a timeslot such that $\int_0^t c_0 > \int_0^t h(t)$. Assume that corrupt players awake at time 0 can predict the leader order up to timeslot t. Then the corrupt players awake at time 0 can costlessly simulate a valid execution of the protocol with chain length $\int_0^t c_0$. As $\int_0^t c_0 > \int_0^t h(t)$, newly woken up honest nodes at timeslot $t+1$ will confirm blocks from the corrupt chain, thus violating safety. (a) A similar attack can be carried out by an adversary under *increasing* participation, but "in reverse", using the players awake at a time slot t in which c_0 corrupt players are awake such that $\int_0^t c_0 > \int_0^t h(t)$ holds, this time costlessly simulating a chain as if they were all awake from time 0.

Wakeness vectors. We define a new primitive in the context of the sleepy model, which might be of independent interest, which we call *wakeness vectors*. Intuitively, wakeness vectors allow nodes to have an estimate on the timeslots in which other nodes were awake throughout the execution a protocol. A similar idea was explored by Bar-Joseph, Keidar, and Lynch [4] in the context of crash failures. Both of our protocols (for the decaying setting, and the fully-fluctuating setting) begin with establishing that the respective models we consider allow the nodes to implement wakeness vectors of sufficient quality. The construction of these vectors employs different techniques in each of the settings. See Sec. 4 and Sec. 5 for an in-depth overview of the main ideas behind the construction of wakeness vectors for the decaying and fully-fluctuating settings, respectively, along with the formal proofs. Armed with this primitive, we then show how can be used to to secure protocol against forward/backward simulation attacks, and enhance protocols in the sleepy model to remain secure amidst decaying/fully-fluctuating participation.

General compiler. For our protocols (Thms. 6 and 8) we actually prove a more general statement than discussed above. We identify key properties, which most existing protocols in the literature satisfy, that are necessary for a protocol to be compatible with our techniques, and then prove that any such protocol can be augmented to be secure amidst fully-fluctuating

Table 1 Overview of previous work and our contributions for atomic broadcast protocols in the synchronous sleepy setting, along with the corresponding properties/assumptions. Below, resilience refers to the fraction of corrupt nodes, f^* refers to the number of *actual* corruptions in a given execution, γ refers to the participation rate of all nodes, and λ is a security parameter; $O(\cdot)^*$ refers to $O(\cdot)$ latency in the optimistic case. The specific adversary restrictions in each of the above results are as follows. See Sec. 6 for further details.

Paper	T_f	T_b	Resilience	VDF	Latency	Adversary restrictions
[23, 15]	0	0	$1/2$	✗	$O(\frac{1}{\gamma}\lambda\Delta)$	Proof of work
[24, 8, 2]	∞	∞	$1/2$	✗	$O(\frac{1}{\gamma}\lambda\Delta)$	–
[16]	∞	∞	$1/2$	✗	$O(\lambda\Delta)$	–
[3]	∞	∞	$1/2$	✗	$O(\frac{\Delta}{\gamma})^*$	–
[18]	∞	∞	$1/3$	✗	$O(n)$	Nodes announce going to sleep
[4]	0	0	$1/3$	✗	$O(f^*)$	Crash failures
[14]	0	0	$1/2$	✗	$O(\Delta)$	Time-stamped messages
[21, 10]	∞	$O(\Delta)$	$1/2$	✗	$O(\Delta)$	–
[21]	$O(\Delta)$	$O(\Delta)$	$1/3$	✗	$O(\Delta)$	Message timeout
[11]	$O(\frac{\Delta}{\gamma})$	$O(\frac{\Delta}{\gamma})$	$\frac{1}{1+e} \simeq 0.268$	✓	$O(\frac{1}{\gamma}\lambda\Delta)$	External adversary
[9]	∞	$O(\Delta)$	$1/2$	✗	$O(\Delta)^*$	–
This paper	$O(\Delta \log^2 \lambda)$	∞	$1/2$	✗	$O(\Delta)$	External adversary
This paper	$O(\Delta)$	$O(\Delta)$	$1/2$	✓	$O(\Delta)$	External adversary

or decaying participation, in the presence of an external adversary. In the next section (Sec. 2), we pour rigor into above discussion by formally defining the models and problems we consider in this work.

2 Preliminaries

Sleepy model. We work in the extended sleepy model that allows fully-fluctuating participation of corrupt nodes as defined in [21]. Specifically, we operate in the permissioned setting, i.e., there exists a set $\mathcal{P}$ of size n which contains the identifiers of all the nodes, which are public and known to all nodes. Without loss of generality, we will assume that $\mathcal{P} = [n]$. At each timeslot t, any node can exist in one of two states, *awake* or *asleep*. The number of awake nodes at timeslot t is denoted by $0 < n_t \leq n$. Asleep nodes at time t are prohibited from sending messages or executing any code at time t, and all messages delivered to them at time t are buffered until the next time step $t' > t$ in which they are awake. Given a protocol Π, there are two types of nodes, *honest* and *corrupt*. Corrupt nodes are controlled by the *adversary* (defined shortly) and can behave in any manner under the restrictions imposed on a given adversary. Honest nodes are the nodes not controlled by the adversary, and they execute Π at all timeslots. As for message delivery, we assume a synchronous network, with synchrony parameter Δ that is fixed and known to all nodes: for every timeslot t, if a node is awake at timeslot t, then we assume that they have received all messages sent to them prior to timeslot $t - \Delta$. We assume that the adversary has control over message delivery timing so long as it abides by the Δ-synchrony assumption. As to not over encumber notation, we omit Δ from the list of inputs to protocols and algorithms discussed in this work. As mentioned earlier, asleep nodes receive all messages sent to them by timeslot $\max(t', t + \Delta)$, where $t' > t$ is the first timeslot when they are awake after timeslot $t + \Delta$. Throughout this paper, we assume that communication is point-to-point.

The environment. We consider an external party to the protocol, *the environment*, which has the role of providing inputs to nodes from some universe I of inputs.

Oracles. We adopt the notion of oracles to model ideal versions of cryptographic primitives, formally defined in [20]. Such a modeling choice assists us in easily formalizing external adversary model we consider in this work, and to more cleanly state our results. Specifically, the description of a protocol also specifies a set of oracles $\mathcal{O} = \{O_1, ..., O_z\}$. At each timeslot t, each node p_i may issue a finite amount of *queries* to an oracle O_j, to which it receives a response at some timestep. An oracle's behaviour may depend on the nodes identity i, the current timeslot t, the input query q, and the internal randomness of the oracle, if the oracle models a random function. More formally, a deterministic oracle O is modeled by a function mapping tuples $(p, m, t) \to (m, t)$ such that $O(p, m, t) = (m', t')$ implies that if p queried O with input m at timeslot t, then p obtained a response m' at timeslot t'. A random oracle is modeled by a distribution over deterministic oracles. Intuitively, in this work, oracles are going to model trusted hardware access to cryptographic primitives. From here on, we denote protocols by the tuple $(\Pi, \mathcal{O})$ where Π is an algorithm executed by honest nodes, and $\mathcal{O}$ is the set of oracles employed by Π. When $\mathcal{O}$ is clear from context, we refer to a protocol as Π.

Adversary and fluctuating participation. Throughout this paper, we focus on the *static* adversary model, i.e., the identities of corrupt nodes are chosen by the adversary prior to the initiation of the protocol, and prior to any randomness being drawn, and in particular, nodes that begin the protocol as honest, cannot be corrupted later. Denote by f the total number of corrupt nodes, and by $\mathcal{F}$ the set of all corrupt nodes. To formalize the notion of fully-fluctuating participation of corrupt nodes, we adopt the terminology introduced in [21]. Specifically, let $\mathcal{F}_t$ be the set of corrupt nodes awake at timeslot t, and let $f(t, T_\mathrm{f}, T_\mathrm{b})$ be defined as follows.

$$f(t, T_\mathrm{f}, T_\mathrm{b}) = |\bigcup_{t-T_\mathrm{f} \leq \tau \leq t+T_\mathrm{b}} \mathcal{F}_\tau|$$

When $T_\mathrm{f}, T_\mathrm{b}$ are clear from context, we at times refer to $f(t, T_\mathrm{f}, T_\mathrm{b})$ by f_t. To provide some intuition for this notation, the parameters $T_\mathrm{f}, T_\mathrm{b}$ indicate how resilient the protocol has to be to forward simulation and backward simulation, respectively. In particular, for a given corrupt node q, if q is awake at time t, then is it considered awake for T_b time-steps *prior* to being awake, and for T_f steps *after* being awake. For a given $T_\mathrm{b}, T_\mathrm{f}$, unless otherwise stated, we always assume that $f(t, T_\mathrm{f}, T_\mathrm{b}) < \frac{n}{2}$ for all t.

Furthermore, the adversary has complete control over the environment and over the set of nodes awake at any timeslot t. While the choice of corrupt nodes occurs at the beginning of the protocol and is thus static, the same does not hold for the awake/asleep status. Namely, the set of nodes awake at each timeslot t can be adaptively chosen by the adversary based on the current transcript (i.e., the contents of all messages sent by honest nodes up to, but not including, the current timeslot) of the protocol and internal state of the corrupt nodes.

External adversary. The view of cryptography via oracles (introduced in [20]) allows us to define the external adversary model in a clean manner, in which (polynomial time) corrupt nodes do not know their secret key, and in particular, due to idealized cryptographic guarantees of the primitives implemented by the oracles, can not produce signatures on behalf of other *asleep corrupt nodes*, except for with negligible probability. This is achieved by not distributing the secret keys of nodes to them, but only allowing them oracle access

to cryptographic functionalities that make use of said keys. More formally, in the external adversary model, given a protocol $(\Pi, \mathcal{O})$, each node p (honest or corrupt) can only issue queries to any oracle $O \in \mathcal{O}$ of the form $(p, \cdot, \cdot)$ throughout the execution of Π. In particular, corrupt nodes, even when colluding, can not obtain fresh oracle queries using the keys of *asleep* corrupt nodes.

This is in contrast to the *standard* adversary in which each node $p \in \mathcal{F}$ can access all the secret of corrupt nodes. Rephrased in oracle terms: Every $p \in \mathcal{F}$ can query any $O \in \mathcal{O}$ with queries of the form $(q, \cdot, \cdot)$, for all $q \in \mathcal{F}$, regardless of the asleep/awake status of q.

Protocols and executions. We are now ready to define the notion of a *protocol* and an *execution* of a protocol. A protocol is specified by a pair $(\Pi, \mathcal{O})$, where Π is the algorithm run by the honest nodes, and $\mathcal{O}$ is the set of oracles employed by Π. An execution of a protocol $(\Pi, \mathcal{O})$ is determined by a 4-tuple, $E = (\Pi, \mathcal{O}, \mathcal{A}, r)$, where $\mathcal{A}$ is an adversary, and the vector r specifies the random coins of all nodes and the adversary. An execution refers further to the contents of all messages sent and received by nodes at each timeslot. Given an execution E, we denote thus by E^t the contents of the execution up to timeslot t, i.e., all random coins and messages received by nodes up to timeslot t. Given an execution E, a timeslot t, and a node i awake at timeslot t, we denote by E_i^t the *view* of node i of the execution E, which includes the internal state of i, the protocol $(\Pi, \mathcal{O})$, and all messages received by i and the timeslots in which they were received. For a given adversary $\mathcal{A}$, we define the *transcript of* Π *at time* t to be all the messages sent by honest nodes up to time t, and the internal state of the adversary $\mathcal{A}$ at time t, and we denote it by $\mathrm{Tr}_{\mathcal{A}}^t$. Throughout the entire paper, we consider a finite execution horizon of $\mathrm{T}_{\mathrm{horizon}} = \mathrm{poly}(n, \lambda)$. We assume that $\mathrm{T}_{\mathrm{horizon}}$ is known in advance to all nodes and in particular can be taken into account in the protocol design. We say that an execution is *admissible* in the $(T_{\mathrm{f}}, T_{\mathrm{b}}, \rho)$-sleepy model iff for all timesteps $t \geq 0$ it holds that $f(t, T_{\mathrm{f}}, T_{\mathrm{b}}) < \rho n_t$.

Atomic broadcast. We consider the atomic broadcast (AB) problem, in which the nodes are required to agree on a linearizable log of their inputs. For every timeslot t, every honest node i reports a *log*, denoted by $\mathcal{L}_i^t$, of its *decided values* $[x_0, x_1, x_2, ...]$. Honest nodes make decisions about the contents of their log based on their internal state, Π, and the messages they received so far in any given execution. More formally, given the random variable $\mathbf{E}$ over executions defined by Π and $\mathcal{A}$, $\mathcal{L}_i^t(\mathbf{E}_i^t)$ is the random variable that denotes the contents of the log of node i at timeslot t, and it depends only on $\mathbf{E}_i^t$. When i and t are clear from context, we omit them from the superscript and subscript of $\mathcal{L}$.

We say that a protocol Π solves the atomic broadcast problem with probability $1 - \epsilon$ in a given execution if the following is maintained throughout the entire execution horizon.

1. ϵ-*Safety.* For every timeslot t, and every i, j such that p_i and p_j are honest with logs $\mathcal{L}(\mathbf{E}_i^t), \mathcal{L}(\mathbf{E}_j^t)$, with probability 1 we have: $\mathcal{L}(\mathbf{E}_i^t)[0 : m] = \mathcal{L}(\mathbf{E}_j^t)[0 : m]$ where $m = \min\{|\mathcal{L}(\mathbf{E}_i^t)|, |\mathcal{L}(\mathbf{E}_j^t)|\}$.
2. (ϵ, ℓ)-*Liveness.* For all $t \in [\mathrm{T}_{\mathrm{horizon}}]$ the following holds: If input x was sent to an honest node p_i by the environment at timeslot t, then $\Pr[\forall t'' \geq t + \ell \colon x \in \mathcal{L}(\mathbf{E}_j^{t''})] \geq 1 - \epsilon$ holds for all honest nodes p_j.

Cryptographic primitives. Let λ be a security parameter. Throughout the paper, we make use of three cryptographic primitives. Namely, a PKI (for signatures), a VRF, and a VDF. We employ a cryptographic hash function (known and computable by all nodes), which we

model as a random oracle, and denote by H. Similarly to previous work, we denote by $\langle m \rangle_p$ the fact that the message m was signed by node p. We model all three of these primitive in an *idealized* fashion, using oracles $O_{\mathrm{S}}, O_{\mathrm{V}}, O_{\mathrm{D}}$ that are defined as follows:

1. Oracle O_{S} on a given input (p, m, t) where p is a node, t is a timeslot, and m is a message is defined to be $(\langle m \rangle_p, t)$. In other words, p receives $\langle m \rangle_p$ in the same timeslot in which it issued the query. From here on in, unless explicitly said otherwise, we assume that every message being sent in any protocol by an honest node is signed by its sender.
2. Oracle O_{V} is defined as $O_{\mathrm{V}}(p, m, t) = (\mathrm{VRF}_{sk_p}(m), \pi_m, t)$. In other words, p receives $\mathrm{VRF}_{sk_p}(m), \pi_m$ in the same timeslot in which it issued the query. Here, $\mathrm{VRF}_{sk_p}(\cdot)$ is a VRF scheme based on p's secret key sk_p, known by the oracle O_{V}, and π_m is the accompanying proof as per the VRF primitive. Again, to keep the focus of the paper on protocol design, we model the functionality of all the VRFs as random oracles. We further assume that the VRF functions accessible to each node all have the same domain ($\{0,1\}^*$), and the same range ($\{0,1\}^\lambda$).
3. Oracle O_{D} is defined as follows when p sends queries $m_1, \ldots, m_k$ to the oracle:

   $$O_{\mathrm{D}}(p, m_1, \ldots, m_k, t) = (\mathrm{VDF}(m_1), \pi_{m_1}, \ldots, \mathrm{VDF}(m_k), \pi_{m_k}, t).$$

   In other words, p receives $\mathrm{VDF}(m_i), \pi_{m_i}$ at timeslot t. Here, $\mathrm{VDF}(m)$ is the VDF computation result on input string m, and π_m is the accompanying proof as per the VDF primitive. We stipulate the following restriction on O_{D}: For every round t and every node p, p can call O_{D} *at most once* at timestep t. In other words, we capture the delay property of O_{D} by not allowing adaptive queries to O_{D} in any single timestep. This restriction applies to both honest and corrupt nodes, thus in particular, the adversary has no advantage in computing the VDF. We consider this modeling as an idealized model for a VDF. We model our VDF as a random function for the purposes of the analysis. We assume that the VDF accessible to each node has domain $\{0,1\}^*$, and range $\{0,1\}^\lambda$.

This would be the place to mention, that even though we model our cryptographic primitives using oracles that execute their ideal functionality, we still assume that all corrupt nodes are computationally bounded in the sense that there exists a polynomial $z(n)$ that depends on the total number of nodes such that the total number of queries Q_p to oracles performed by any node p in a single timeslot satisfies $Q_p \leq z(n)$. Finally, for an event E we say that E holds *with high probability* (w.h.p.) if $\Pr[E] \geq 1 - \mathsf{negl}(\lambda)$.

3 Tools and definitions

3.1 Wakeness vectors

Consider vectors $v^1, ..., v^n$ whose dimension is $\mathrm{T}_{\mathrm{horizon}}$. These vectors are defined as follows for each timeslot t: For all $i \in [n]$ and all $j \in [t]$, $v^i[j] = 1$ if and only if node p_i was awake at timeslot j, and 0 otherwise. Endowing honest nodes with such vectors should intuitively make protocol design in the sleepy model substantially easier. Our upper bounds are constructed by having nodes implement subroutines that allow them to emulate an approximate version of this ideal functionality. The formal definition is as follows.

▶ **Definition 1.** *Let $(\Pi, \mathcal{O})$ be a protocol and let $\mathbf{E}_{\Pi,\mathcal{A}}$ be the random variable indicating a sample from the distribution over executions of Π defined by adversary $\mathcal{A}$. For a given T, we say that $v_p \in \{0,1\}^T$ is a* d-valid wakeness vector *of node p with respect to $\mathbf{E}$ iff, except with negligible probability, $v_p[j] = 1$ implies that p is awake at a timeslot $j' \geq j - d$. Furthermore, if p is honest, then p being awake at timeslot j implies $v_p[j] = 1$. When v is a 0-valid wakeness vector, we simply refer to it as valid.*

Throughout the paper, we assume without loss of generality (w.l.o.g.) that all honest nodes keep a *local view* of wakeness vectors. I.e., if $(\Pi, \mathcal{O})$ is a protocol, then in any execution E of Π, each honest node p stores $v_p^1, ..., v_p^n$, where $v_p^j \in \{0,1\}^{\mathrm{T_{horizon}}}$ for all j. We say that p *deems* node i to be awake at timeslot t iff $v_p^i[t] = 1$. For an interval I of timeslots, we at times abuse notation and use $v_p^i[I]$ to refer to all coordinates in I in v_p^i. The main two upper bounds we prove in this work can be formalized as a general statement about augmenting atomic broadcast protocols with wakeness vectors. We present the formal lemmas in the appropriate sections.

Both our upper bounds are general compiler results, that augment protocols working in the (∞, ∞, ρ)-sleepy model with wakeness vectors of sufficient quality to support fully-fluctuating participation. These augmentations however don't work for arbitrary protocols. Specifically, there is a key set of properties regarding the structure of a protocol that we must assume in order for our results to go through. It turns out, however, that the vast majority of literature on protocols for the sleepy model already posses these properties. Specifically, we define notions that we refer to as *Statelessness* and *Unpredictability*.

Stateless protocols. Intuitively, a protocol is said to be stateless if an honest nodes action depends on messages it received from nodes it has *deemed* to be awake in recent timeslots. We assume w.l.o.g. that in any protocol Π, honest node keep a local view of wakeness vectors. Note that these vectors are not necessarily d-valid for any d. Such a property depends on the model and protocol in question. We further assume w.l.o.g. that honest nodes always attach the current timeslot to their messages.

▶ **Definition 2.** *Let $(\Pi, \mathcal{O})$ be a protocol. We say that Π is* T-stateless *if the following holds for all honest nodes p and timeslots t and pair of executions E_0, E_1. Let $v_p^1, ..., v_p^n$ be the wakeness vectors viewed by p at timeslot t. Let $T' \leq T$, chosen by p, and consider the set $S = \left\{i \in [n] \mid \exists j \in [t - T', t - 1], v_p^i[j] = 1\right\}$. Then if S is the same in both E_0, E_1, and furthermore, the messages sent from nodes in S to p are the same in both E_0, E_1, then the distribution over actions of p is the same at timeslot t in both E_0, E_1. Furthermore, Π solves atomic broadcast in any execution in which for all honest nodes p and timeslots t, S has a strict honest majority.*

Unpredictability. Intuitively, this property captures the adversary's ability to predict the contents of the log of honest nodes at some future timeslot.

▶ **Definition 3.** *Let $(\Pi, \mathcal{O})$ be a protocol that solves atomic broadcast. We say that Π has* α-unpredictability *if the following holds for all t, and for all adversaries $\mathcal{A}$. Let $\mathrm{Tr}_{\mathcal{A}}^t$ be the transcript of the protocol up to timeslot t, and let $\mathcal{L}_{\mathcal{A}} \leftarrow \mathcal{A}(\Pi, \mathrm{Tr}_{\mathcal{A}}^t, \alpha)$ be an adversarial log, which we refer to as the adversary's* guess. *Then for all honest nodes p_i, it holds that $\Pr[\mathcal{L}(\mathbf{E}_i^t) \prec \mathcal{L}_{\mathcal{A}} \preceq \mathcal{L}(\mathbf{E}_i^{t+\alpha})] = \mathsf{negl}(\lambda)$.*

4 Decaying participation

Efficient protocols for the $T_\mathrm{f} = \infty$ regime (i.e., increasing participation) can be found in [22, 21, 9], with T_b being nearly optimal, namely with $T_\mathrm{b} = O(\Delta)$. In this section, we tackle the yet unstudied $T_\mathrm{b} = \infty$ case, in which participation is decaying. Namely, in the decaying participation model, corrupt nodes are allowed to go to sleep, but may never wake up after timeslot 0. We show that that PKI, VRF, and an external adversary suffice to design protocols in this setting, with $T_\mathrm{f} = O(\Delta\lambda)$, where λ is the security parameter, with error

probability of $O(\frac{1}{2^\lambda})$. This result establishes the benefit of considering an external adversary, as an impossibility result by [21] established that atomic broadcast is impossible to solve in the $T_b < \infty$ case against a standard adversary that can carry out the key transfer attack.

Blocks. We employ the notion of a *block*. Discussing blocks allows our black box protocols to be compatible with previous protocols in the literature that employ such a notion. We think of blocks as being built from the inputs sent by the environment to the nodes in the following way. In general, each block B added to the chain is going to have the following structure: Each block has the form $B = (x, \mathsf{H}(B'), v, e, \mathsf{seed})$, where x is the content of the block. You should think of x as all of the inputs sent by the environment to nodes that the block constructor has seen thus far that are not already decided. Here, $\mathsf{H}(B')$ is a hash of the previous block in the chain, with all honest nodes initializing their chain with the *genesis block*, denoted by $B_0 = (\bot, \bot, 0)$. At times, protocols induce a partition of the timeslots into intervals which are referred to as *views* and *epochs*, when that is the case, v indicates the *view* in which the block was created, and e is the *epoch* in which the block was created. When the protocols we discuss do not make use of views and epochs, we simply omit these coordinates from the description of a block. Lastly seed is an ancillary string. Similarly, when we discuss protocols in which the seed entry is not used, we simply omit it from the block description. We denote by $\mathsf{view}(B)$ and $\mathsf{epoch}(B)$ the view and epoch, respectively, in which block B is was (or claimed to be) created. We define the *height* of a block to be its distance in the chain from the genesis block, and we denoted it by $h(B)$.

Note that any block B defines a unique path from B back to the genesis block, and thus defines a unique log. We say that a block B extends a block B' if $B = B'$ if or B' is an ancestor of B. We say that two block B, B' *conflict* with each other if neither one of them extends the other. We say that a block B is valid if its ancestor block B' is valid and $\mathsf{epoch}(B') < \mathsf{epoch}(B)$ or ($\mathsf{view}(B') < \mathsf{view}(B)$ and $\mathsf{epoch}(B') = \mathsf{epoch}(B)$). If the view or epoch are omitted from the block, the above conditions hold vacuously.

Permissible block. On top of being valid, a protocol may impose additional constraints on blocks in order for them to be considered for confirmation. This usually takes the form a *locally computable* (i.e., no communication required) predicate ξ, computable efficiently and locally by every node, so that $\xi(B) = 1$ if B is considered a permissible block, and 0 otherwise. Messages involving impermissible blocks simply get ignored by honest nodes throughout the execution of the protocol.

4.1 Graded proposal election

To obtain the result we require the observation that the protocol of [21] can be endowed with the unpredictability property with a slight modification which we discuss formally later in the section. This modification concerns a procedure called *graded proposal election*, which we define here for completeness.

▶ **Definition 4.** *In* graded proposal election *(GPE), each node p has an input block B_p, and outputs a single pair (B, g) of a block and a grade ($g \in \{0, 1\}$) with the following properties.*

1. Consistency. *If two honest nodes p, q output blocks B, B' respectively, so that $B \neq \bot, B' \neq \bot$, then $B = B'$.*
2. Graded Delivery. *If an honest node p outputs $(B, 1)$, then all honest nodes output $(B, *)$.*
3. Validity. *With probability (w.p.) at least $\frac{1}{2}$, all honest nodes output $(B, 1)$ for some block B that was the input of some honest node p.*
4. Integrity. *If an honest node p outputs $(B, *)$, then either $B = B_0$, or there exists an honest node p' that received the* content *of B from the environment.*

In their paper, Malkhi, Momose, and Ren [21] exhibit a protocol solving GPE in $O(\Delta)$ timesteps in the $(\infty, \infty, \frac{1}{2})$-sleepy model.

4.2 Protocol and proof

We now have all the tools to state our results formally. The following result employs the notions of log unpredictability (Def. 3), and the notion of a stateless protocol (Def. 2). We can now formally state the following lemma, which intuitively says that assuming an external adversary, one can turn any protocol with non-trivial unpredictability and statelessness, into a protocol that can tolerate decaying participation.

▶ **Lemma 5.** *Let $(\Pi, \mathcal{O})$ be a protocol solving atomic broadcast w.p. $1-\epsilon$ and liveness parameter ℓ in the (∞, ∞, ρ)-sleepy model, and assume that:*

- *Π has α-unpredictability, and*
- *Π is α-stateless.*

Then assuming an external adversary, $\rho < \frac{1}{2}$, there is a protocol $(\Pi', \mathcal{O})$ that solves the atomic broadcast problem in the $(O(\alpha), \infty, \rho)$-sleepy model w.p. $1-\epsilon-\mathsf{negl}(\lambda)$ and liveness parameter ℓ . Furthermore, Π' expected latency is upper bounded by that of Π', and incurs an additive $O(\lambda n^2)$ factor of communication per timeslot. In particular, assuming an external adversary, one can implement $O(\alpha)$-valid wakeness vectors in the $(O(\alpha), \infty, \rho)$-sleepy model.

Lem. 5 is established via a reduction, depicted in Alg. 1, that constructs Π' given blackbox access to the given protocol Π in the premise of the lemma. Instantiating Π in the reduction of Lem. 5 with a modification (detailed in Alg. 2) of the atomic broadcast protocol of [21, Alg. 3], we get the following result:

▶ **Theorem 6.** *Assuming PKI, VRF, and an external adversary, there is a protocol Π, solving the atomic broadcast problem w.p. $1-\frac{1}{2^\lambda}$ in the $(O(\Delta\lambda), \infty, \frac{1}{2})$-sleepy model with $O(\Delta)$ expected latency, and liveness parameter $\ell = O(\Delta\lambda)$.*

We next provide an overview of the main ideas that go into the proof of Lem. 5. The key observation for the decaying participation setting against an external adversary is that honest nodes can *always* rely on the messages *claimed* to have been sent in the earlier timeslots, due to the $T_\mathrm{b} = \infty$ assumption. The rough idea then is two-fold.

1. **Random log.** The main challenge in the decaying participation setting is dealing with forward simulation attacks, similarly to Fig. 2b. Our idea for handling such attacks is by introducing *nested* randomness to the log decided by honest nodes. Specifically, along with a proposal for a block, the proposal must include a VRF output, which is computed on the VRF output attached to the parent of the proposed block. In case the proposer is honest, this randomness is unknown to the corrupt nodes until the moment of broadcasting the proposal. As such, corrupt nodes attempting to construct a corrupt chain will not be able to do so with the randomness of honest nodes beyond the latest available block/proposal sent by honest nodes. Thus, the key is to ensure that blocks proposed by *honest* nodes are decided often enough. This endows the log with an unpredictability property of the contents of the log with corrupt nodes awake in the early timeslots, but not in the current timeslot. We show that we can ensure that w.h.p., a block proposed by an honest node gets decided once every $O(\Delta\lambda)$ timeslots, hence our choice of $T_\mathrm{f} = O(\Delta\lambda)$.
2. **Tolerating forward simulation.** In the $T_\mathrm{f} < \infty$ model, we are susceptible to forward simulation attacks, in which corrupt nodes active in early rounds send messages to honest nodes awake far in the future, pretending to have been awake all along. With the

Algorithm 1 Protocol Π' for Lem. 5.

// Each node p, if it is awake, executes the following at all timeslots t. Time is divided into epochs, each of length α, starting from timeslot 0. Denote the current epoch by $e(t)$. Each node initializes a local variable $\mathcal{L}_p^* = B_0$. Initialize $\mathcal{L}_i^0 \leftarrow B_0$. Epoch -1 is the same as epoch 0. Initialize $v_p^1 = ... = v_p^n = 0^{\mathrm{T}_{\mathrm{horizon}}}$.
// Update log
for $e = 0, ..., e(t) - 1$
  $D^* \leftarrow$ # of nodes q so that p received $\langle decide, *, e\rangle$
  for all blocks B *extending a block of epoch* $e-1$ *in* $\mathcal{L}^*$
    $v_p^i[e] = 1^\alpha$ for every p_i that sent a block B extending a block of epoch $e-1$ in $\mathcal{L}_p^*$
    $D(B) \leftarrow$ # of nodes q so that p received $\langle decide, B', e\rangle$ for any B' extending B
    if $D(B) > D^*/2$
      Add B and all its ancestors to $\mathcal{L}_p^*$
// Executing Π
Execute Π as instructed at timeslot t, ignoring messages from nodes p_i that do not satisfy $v_p^i[e(t)-1] = 1^\alpha$
upon a block B is *decided* according to Π
  Decide B and all its ancestors.
  $\mathcal{L}_p^* \leftarrow B$ and all its ancestors.
// Decision messages
Multicast $\langle decide, B, e(t)\rangle_p$, where B is the most recently decided block

Algorithm 2 Augmenting [21, Alg. 3] with unpredictablility.

// $\mathsf{GPE_v}$ invocation
if $r \in [0, 4\Delta]$
  $B \leftarrow (x, H(\mathsf{candidate}), v(t), e(t), O_{\mathrm{V}}(\mathsf{candidate}, t))$ for some $x \in I_p$
  Execute $\mathsf{GPE_v}$ with input B. A block B is *permissible* if it extends lock and has $\mathsf{view}(B) = v(t)$, $\mathsf{epoch}(B) = e(t)$, and $O_{\mathrm{V}}(\mathsf{candidate}, t)$ consists of a valid output of O_{V} for input of the form $(p, \cdot, \cdot)$.

randomness introduced in the first bullet point, honest nodes joining the protocol at a late time can recover the honest log by *inductively* building it, starting from the first block. The inductive argument then intuitively goes as follows.

- The correct recovery of the *first* block in the log is guaranteed by the $T_{\mathrm{b}} = \infty$ assumption. Namely, the number of honest nodes awake at the commencement of the protocol is greater than the total number of corrupt nodes that are *ever* going to be awake during the execution. This is the base step of the reduction.
- The randomness introduced to the log and the choice of T_{f} allows honest nodes to safely deduce the next block in the log, given correct recovery up to block k, for any k. Roughly, this follows from two observations. First, any forward simulation of the protocol constructing a corrupt log up to $O(\Delta\lambda)$ timeslots forward is going to get outvoted by honest nodes, due to the $T_{\mathrm{f}} = O(\Delta\lambda)$ assumption. Any attempt by corrupt nodes to forward simulate the protocol for $\omega(\Delta\lambda)$ timeslots is foiled by the fact that any adversarial log is almost certainly different than the honest log, as its suffix contains no blocks proposed by honest nodes, and thus the randomness attached to it is unpredictable to corrupt nodes. Hence, honest nodes awake during the following $O(\Delta\lambda)$

timeslots are still going to outvote the fake log, due to the $T_f = O(\Delta\lambda)$ assumption. This allows honest nodes to safely deduce block $k+1$ of the log by deciding on the block for which they observe a majority of votes.

The proof of Lem. 5 involves augmenting a general protocol with resilience to forward simulation, this augmentation is presented in Alg. 1. Afterwards, in order to obtain Thm. 6, we explain how to augment the protocol of [21, Alg. 3] with randomness and thus endow it with $O(\Delta \log^2 \lambda)$-unpredictability (see Def. 3).

Proof of Lem. 5. Let Π be the protocol given in the premise of Lem. 5. The protocol we construct is described in Alg. 1. The protocol is divided into epochs, each of length α. Each node that wakes up, reconstructs the contents of the log inductively starting from the first epoch. The inductive step is then implemented by each honest node ignoring all messages regarding blocks of epoch e that do not extend a block of epoch $e-1$ confirmed by the node. This is formalized as described in Alg. 1. In Alg. 1, the notation $v_p^i[e]$ corresponds to all coordinates of v_p^i corresponding to epoch e. First we prove the following property of Alg. 1.

▶ **Lemma 7.** *Except for with negligible probability, it holds that for any timeslot t and honest node p, $v_p^1, ..., v_p^n$ are 3α-valid wakeness vectors in all $(T_f = 10\alpha, \infty, \rho)$-sleepy executions. Furthermore, it holds that at the end of iteration k of* update log *(See line 2 of Alg. 1), that $\mathcal{L}_p^*$ contains a block of epoch k of $\mathcal{L}(\mathbf{E}_p^t)$, and contains no blocks that were not decided by at least one honest node.*

Proof. We prove this by induction over the epochs. Denote by t the current timeslot. For $e = 0$, the wakeness vector statement is trivial, i.e., whenever $v_q^i[e=0] = 1^\alpha$ for an honest node q for some i, then node i was awake at timeslot at least 0. For $\mathcal{L}_q^*$, Let B be a block in $\mathcal{L}^*$ that extends the genesis block, this implies that q observed a majority of *decide* messages for B amongst all decide messages it received for blocks extending the genesis block. This includes all the honest nodes awake during epoch 0, by line 16 of Alg. 1. By the $(10\alpha, \infty, \rho)$-sleepy model, the honest nodes awake during epoch 0 exceed the total number of corrupt nodes that are ever going to be awake during the execution. This implies that p received a *decide* message for B from at least one honest node p, i.e., $B \in \mathcal{L}(\mathbf{E}_p^{t'})$ for some timeslot $t' < t$, as required. The log $\mathcal{L}_q*$ contains the genesis block, and thus contains a block of $\mathcal{L}(\mathbf{E}_p^t)$ of epoch 0. For the induction step, consider the claim to be true for all epochs at most $\leq e$, we prove the claim for epoch $e+1$. Let q be an honest node awake at timeslot t of epoch $\geq e+1$ and let i be a node such that $v_q^i[e] = 1^\alpha$. This implies that q received a *decide* message from i extending a block in $\mathcal{L}_q^*$ of epoch $e-1$. Consider the event E in which p_i was last awake during epoch $e' \leq e-5$. By the induction hypothesis, up to epoch $e-1$, all wakeness vectors stored by all honest nodes are 3α valid. In particular, due to the $(T_f = 10\alpha, \infty, \rho)$-sleepy assumption, and due to the α-stateless property of Π, we get that at least up to the end of epoch $e-1$, Π correctly solves the atomic broadcast problem, as by line 11 of Alg. 1, all honest nodes consider messages from a set that has an honest majority. This allows us to invoke the α-unpredictability of Π, together with the second induction assumption, that assures us that $\mathcal{L}_q^*$ contains a block in $\mathcal{L}(\mathbf{E}_q^t)$ of epoch $e-1$ to claim that the probability that i could send to q a block extending a block of epoch $e+1$ in $\mathcal{L}_q^*$ is $\mathsf{negl}(\lambda, n)$. This invokes the external adversary assumption, meaning that a message observed to be from i can be sent only by i (or other corrupt node) at a timeslot in which i is awake. Thus E occurs with at most negligible probability. Applying a union bound over all corrupt nodes and over all timeslots during the execution horizon (both polynomial in n, λ) maintains this negligible probability, and proves the claim for the wakeness vectors. This

guarantees the correctness of Π for epoch $e+1$ by its α-stateless property. This implies that $\mathcal{L}$ is consistent between all honest nodes at timeslot t. Now we must prove that at iteration e, a block of epoch e in $\mathcal{L}(\mathbf{E}_q^t)$ is added to $\mathcal{L}_q^*$, and no blocks that are not decided by some honest node are added.

Assume that for an honest node q no block of epoch e is added to $\mathcal{L}_q^*$ by the end of iteration e. This implies that there exists an honest node p, awake at timeslot $t-1$, that doesn't have any blocks of epoch e in $\mathcal{L}(\mathbf{E}_p^{t-1})$. This again holds by the induction assumption of $\mathcal{L}^*$ for iteration $e-1$, the 3α-validity of the wakeness vectors, the α-unpredictability of Π, and the $(10\alpha, \infty, \rho)$-sleepy execution. All these imply that q is hearing from an honest majority, even when restricted to honest nodes awake at timeslot $t-1$. So if all honest nodes awake at timeslot $t-1$ had a block of epoch e decided, consistency of Π would imply that it is added to $\mathcal{L}^*$ for q at line 9 of Alg. 1.

This event can only happen with negligible probability, as otherwise we contradict the α-unpredictability of Π, as the adversary at round $t-\alpha-1$ had already seen all blocks of epoch at most $e-1$ communicated between honest nodes, including the transcript of the protocol, and all messages received by p, and could thus compute $\mathcal{L}(\mathbf{E}_p^{t-1})$ with non negligible confidence. Thus except for with negligible probability, all honest nodes awake at timeslot $t-1$ have a block of epoch e decided, and thus this block is added to $\mathcal{L}^*$ by q at timeslot t by the behaviour of the protocol and the induction assumption. Lastly, again since q hears from an honest majority, we get that every block B added to $\mathcal{L}_q^*$ was decided by at least one honest node, i.e., added to $\mathcal{L}(\mathbf{E}_p^{t'})$ for some honest node p at a timeslot $t' < t$. ◀

The 3α-validity of the wakeness vectors given by Lem. 7, combined with the $(10\alpha, \infty, \rho)$-sleepy executions, and the α-statelessness of Π, allow us to deduce that Π' (Alg. 1) solves the atomic broadcast correctly as required. Notice that block confirmation in Π' coincides with block confirmation in Π, and thus they have the same latency properties. The only added communication occurs in *decision messages* part of the protocol, and each message is of length $O(\lambda)$, thus the total communication blowup per timeslot is $O(\lambda n^2)$, as required. ◀

Proof of Thm. 6. All that is left to obtain Thm. 6, is to observe that the protocol (specifically, Algorithm 3) of [21] (henceforth [21, Alg. 3]) is $O(\Delta)$-stateless, works in the $(\infty, \infty, \frac{1}{2})$-sleepy model (see Thm. 11). It can be augmented to be $O(\log^2 \lambda)$-unpredictable with an incredibly simple change to the *GPE invocation* part of the protocol, which we describe in Alg. 2. In words, each honest node, when inputting a block B to $\mathsf{GPE_v}$, simply includes in its seed a VRF computation on the current timeslot and the parent block of B. This ensures that the contents of each block contain a component with high entropy, thus making the logs of honest nodes unpredictable.

More formally, the augmented algorithm is described in Alg. 2, with the red lines indicating changes from the original pseudo-code. Due to the simplicity of the change, and not to burden the write up with the technical details of [21, Alg. 3], we provide below a sketch of the proof argument. [21, Alg. 3] is divided into *views*, each consisting of $O(\Delta)$ timeslots, during which an attempt to agree on this next block in the log is executed. This attempt is captured by the GPE procedure, which is executed once every view. By the properties (in particular, validity) of the GPE subroutine (see Def. 4) and the behaviour of [21, Alg. 3], w.p. at least $\frac{1}{2}$, at the end of the GPE subroutine, *all* honest nodes decide a block proposed by an honest node into their log. Conditioned on this event, the contents of $O_{\mathrm{V}}(\mathsf{candidate}, t)$ are uniformly random and *unpredictable* to any adversary not awake before the beginning of the current view. In particular, the adversary $\mathcal{A}$ attempting at timeslot t to guess the contents of the log of *any* honest node, succeeds w.p. at most $\frac{1}{2^\lambda}$. As different GPE executions are

independent, we get that after $O(\lambda)$ views, condition 3 in Def. 4 is invoked except w.p. $\frac{1}{2^\lambda}$. Thus in total, the probability that an adversary at timeslot t can correctly guess the contents of the log of *any* honest node at timeslot $t + O(\lambda\Delta)$ is at most $\frac{1}{2^\lambda}$, as required. ◀

5 Fully-fluctuating participation

Finally, we move on to consider fully-fluctuating participation regime, where both $T_\mathrm{f}, T_\mathrm{b} < \infty$. We show that assuming an external adversary and a VDF, one can design efficient protocols in this setting.

▶ **Theorem 8.** *Assuming a PKI, a VRF, a VDF, and an external adversary, there exists a randomized protocol that with high probability (w.h.p.) solves atomic broadcast with expected latency of $O(\Delta)$, and liveness parameter $\ell = O(\Delta \log \frac{1}{\epsilon})$ in all admissible $(O(\Delta), O(\Delta), \frac{1}{2})$-sleepy executions.*

The proof of Thm. 8 is driven by two lemmas. The first lemma (Lem. 9, stated below) is to show that there is an efficient protocol that implements valid wakeness vectors, assuming PKI, VRF, and VDF oracles, and an external adversary:

▶ **Lemma 9.** *Assuming a PKI, a VRF, a VDF, and an external adversary, there is a protocol with $O(\lambda n^2 \log^2 n)$ communication complexity per timeslot that w.h.p. implements valid wakeness vectors for all timeslots t over an execution horizon of length $\mathrm{T}_{\mathrm{horizon}}$.*

The second lemma (Lem. 10, stated below) is a general result that gives a black-box simulation showing how to convert *any* stateless protocol for atomic broadcast in the $(\infty, T_\mathrm{b}, \rho)$-sleepy model into an atomic broadcast protocol in the stronger $(T_\mathrm{b}, T_\mathrm{b}, \rho)$-sleepy model:

▶ **Lemma 10.** *Let $(\Pi, \mathcal{O})$ be a protocol solving atomic broadcast w.p. $1 - \epsilon$ in the $(\infty, T_\mathrm{b}, \rho)$-sleepy model with liveness parameter ℓ. Furthermore, assume that Π is T_b-stateless (Def. 2). Then, assuming a PKI, a VRF, an external adversary, $\rho < \frac{1}{2}$, and valid wakeness vectors, there exists a protocol $(\Pi', \mathcal{O})$ solving atomic broadcast in the $(T_\mathrm{b}, T_\mathrm{b}, \rho)$-sleepy model w.p. $1 - \epsilon$ and liveness parameter ℓ. Furthermore, Π' has expected latency upper bounded by that of Π and incurs an additive $O(\lambda n^2 \log^2 n)$ communication cost per timeslot.*

We prove Lem. 10 and Lem. 9 by presenting a protocol (Alg. 3) that both implements wakeness vectors using VDFs in the external adversary model, and uses a given protocol Π as a black box. In a high level, every node at each timeslot has three responsibilities:

1. Extracting decided log so far from received messages. This computation is internal, i.e. involves no communication.
2. Extension of wakeness vectors.
3. Running Π.

The idea is to build, along with the blockchain of inputs of nodes, an object we refer to as *VDF chains*, from which one can extract valid wakeness vectors for honest nodes to convince other honest nodes of them being awake at certain timeslots. This allows the honest nodes that wake up after a long period of sleepiness to only consider messages from nodes that were awake in recent timeslots to realize the current state of the blockchain; the T_b interval in both forward simulation and backwards simulation give us the needed majority to make sure the honest nodes can perform these tasks successfully.

The proofs of wakeness allow honest nodes to determine the identities of of nodes that were awake in the previous view, and take only their messages into account in deciding on the state of the log, using the $(T_\mathrm{b}, T_\mathrm{b}, \frac{1}{2})$-sleepy assumption regarding the permissible executions.

Algorithm 3 Constructing protocol Π' from Π.

// The following is executed by every honest node p at every timeslot t in which it is awake. Let $T = \text{poly}(\lambda, n)$ be the execution horizon. Initialize $v_p^i \leftarrow 0^T$ for all $i \in [n]$. $\mathcal{L}_i^0 \leftarrow B_0$.
// Wakeness vector updates
Let $(v_p^1, ..., v_p^n)$ be the local view of p from the last timeslot r in which p was awake
for $r < t' \leq t$
  for all $i \in [n]$
    if p observes a valid t'-depth value with prover p_i
      $v_p^i[t'] \leftarrow 1$ // p deems p_i awake at timeslot t'
    else
      $v_p^i[t'] \leftarrow 0$ // p deems p_i asleep at timeslot t'
// VDF chain extension
$P \leftarrow$ set of nodes q so that p received a valid depth-$(t-1)$ value val_q from q
$S \leftarrow O(\log^2(n \cdot t))$ uniformly random elements (q, val_q) sampled from P
for all $(q, \mathsf{val}_q) \in S$
  Compute $O_{\text{D}}(p, \mathsf{val}_q, t)$ and multicast $\langle \text{VDF}(\mathsf{val}_q) \rangle_p$
// Running the protocol
Execute Π as instructed, and in particular, decide when Π decides, while ignoring messages from nodes p_i for which $v_p^i[t - T_{\text{b}} : t - 1] = 0$

Thm. 8 follows from Lem. 9 and Lem. 10, by instantiating the protocol Π to be the protocol from [21]. Specifically, [21, Alg. 3] proves the following theorem, despite not stating the stateless property explicitly.

▶ **Theorem 11** (Theorem 1, [21]). *Assuming a PKI and a VRF, there exists a $O(\Delta)$-stateless protocol Π solving atomic broadcast w.p. 1 with expected latency $O(\Delta)$, and liveness parameter $\ell = O(\Delta \log \frac{1}{\epsilon})$, in any admissible $(\infty, O(\Delta), \frac{1}{2})$-sleepy execution.*

Sec. 5.1 is devoted to the proofs of Lem. 9, and Lem. 10.

5.1 Constructing wakeness vectors from VDFs

This section is devoted to proving Lems. 9 and 10. Thm. 8 then follows. In the external adversary model, where corrupt nodes may only access their secret keys via oracles, a VDF assumption allows any honest node p to prove w.h.p. the following statement for any timeslot t, if it is true: "node p was awake at some timeslot $t' \geq t$". In order to prove this statement with VDF, we need to define a few notions first.

▶ **Definition 12.** *Let val be some value in the range of the VDF function, and let p be some node. We say that val is a valid* depth-v *value w.r.t. p if there exists values $\mathsf{val}_0, ..., \mathsf{val}_{v-1}, \mathsf{val}_v$, where $\mathsf{val}_v = \mathsf{val}$, and nodes $p_0, ..., p_{v-1}, p_v$, where $p_v = p$, such that following holds.*

1. *For all $i \in [v]$, $\text{VDF}(\mathsf{val}_i) = \mathsf{val}_{i+1}$.*
2. *Node p has received $\langle O_{\text{D}}(p_i, \mathsf{val}_i, \cdot) \rangle_{p_i}$ for all $i \in [v]$.*

We call p_{v-1} the prover, and val the v-proof.

We first note that if p received a valid depth-v value val from p_{v-1}, then p knows that w.h.p. p_{v-1} sent that value at a timeslot $v' \geq v$, and in particular was awake at a timeslot $v' \geq v$. We give a formal proof of this below.

▷ Claim 13. In the presence of a external adversary, if a node p receives a valid depth-t value val from a node p_{t-1}, then except with negligible probability, p_{t-1} sent val at timeslot $t' \geq t$, and was *awake* at a timeslot $t'' \geq t-1$.

Proof. We prove this by induction on t. For $t = 0$, the claim is trivial since a node that wasn't awake at any time ≥ 0 can not send any messages. Assume that node p received a valid depth-t value val from q, for q that was not awake at any timeslot $\geq t$. Denote the corresponding nodes and values by $p_1, ..., p_{t-1} = q$, $\mathsf{val}_0, ..., \mathsf{val}_{t-1}$. Note that in particular, val_{t-1} is a valid depth-$(t-1)$ value w.r.t. p, sent by p_{t-2}, and thus by the induction assumption, it was sent by p_{t-2} at timeslot $t'' \geq t-1$.

Now assume that val was sent by p_{t-1} at a timeslot $r < t$, this means that p_{t-1} had not yet received val_{t-1} from p_{t-2}. Denote by Q the total number of queries to the VDF oracle made by any node by timeslot r, that number is bounded by $r \cdot n \cdot p(n, \Pi)$, in particular, the probability that val_{t-1} appeared as the valid output of a VDF oracle at timeslot $\leq r$ is at most $\frac{r \cdot n \cdot p(n,\Pi)}{2^n}$, and thus as long as $r = o(2^n)$, w.h.p. p_{t-1} could not have computed $O_{\mathrm{D}}(p_{t-1}, \mathsf{val}_{t-1}, \cdot)$ before timeslot r, for any $r < t$. Thus w.h.p. p_{t-1} could have computed $O_{\mathrm{D}}(p_{t-1}, \mathsf{val}_{t-1}, \cdot) = \mathsf{val}$ only after p_{t-2} sent val_{t-1}, which occurred at timestep at least $t-1$, i.e., p_{t-1} could have only queried $O_{\mathrm{D}}(p_{t-1}, \mathsf{val}_{t-1}, \cdot)$ at timestep at least $t-1$, and received a response at timestep at least t due to the behavior of O_{D}, thus p_{t-1} sent val at timestep at least t, and was thus awake at timestep at least $t-1$, as required. ◁

Proof of Lem. 9. The full algorithm is depicted in Alg. 3, specifically in the *wakeness vector updates* and *VDF chain extension* parts.

With Clm. 13 in mind, the main observation is that to prove the lemma, it suffices to prove that at every timeslot, every honest node extends a valid VDF chain of at least one other *honest* node. Notice that this is necessary as well, as otherwise, consider the following case involving 3 nodes p_1, p_2, p_3.

1. Node p_3 is corrupt and awake at timeslot 1, node p_1 is honest and awake at timeslot 2, node p_2 is honest and awake at timeslot 3.
2. Node p_3 computes $\mathsf{val}_1 = O_{\mathrm{D}}(p_3, \mathsf{val}_0, 1)$, and sends it to p_1, but not to p_2.
3. By instructions of some protocol Π, p_1 computes $\mathsf{val}_2 = O_{\mathrm{D}}(p_1, \mathsf{val}_1, 2)$ and broadcasts it.

Note now that p_2 by timeslot 3 did not receive $\mathsf{val}_1 = O_{\mathrm{D}}(p_3, \mathsf{val}_0, 1)$, and thus in particular, w.h.p. val_2 is not considered a valid depth-2 value w.r.t. to p, even though p_1 was indeed awake at timeslot 2.

The trivial solution to this would be to instruct each honest node p to extend the VDF chains it received from all nodes, thus extending all honest node's chains. However, this incurs a multiplicative n cost in communication. A better approach, and the one we implement in Alg. 3, is to let each honest node p sample $O(\log^2(n \cdot t))$ of the nodes it received valid depth-t VDF computations from, and extend those chains, thus guaranteeing that except with negligible probability, at least one of the chosen nodes is honest (due to the honest majority assumption in any admissible execution) and that for all t, except with negligible probability all honest nodes at all timeslots t observe valid chains from all honest nodes for the appropriate timeslots they were awake.

We prove this by induction on the timeslot. The base case of $t = 0$ clearly holds as all honest nodes have valid wakeness vectors. Assume correctness for t and consider time $t+1$, let p be an honest node awake at time t. By the induction assumption, p had valid wakeness vectors, and so p set $v_p^i[t-1] = 1$ for all honest nodes i awake at time $t-1$. In particular, by the behavior of the protocol, this implies that p observed valid depth-$(t-1)$ w.r.t. all honest nodes i awake at time $t-1$. Furthermore, the validness of the wakeness vectors of p

at timeslot t, the set of nodes i for which $v_p^i[t-1] = 1$ has an honest majority. As such, by lines 11, 12 of Alg. 3, when p chooses $O(\log^2 n)$ random nodes amongst these, the probability of an honest node *not* being chosen is at most $2^{\log^2 n} = \mathsf{negl}(n)$. Thus w.h.p. p extends in round t a valid depth-$(t-1)$ chain sent by an honest node, which implies that at round $t+1$, all honest nodes see a valid depth-t honest chain from p, and thus set p as being awake in their wakeness vectors, as required. This proves the completeness of the wakeness vectors at time $t+1$, i.e., that for all honest q awake at time $t+1$, and all $t' < t+1$ and all honest p awake at time t', we have that $v_q^p[t'] = 1$. The soundness of the wakeness vectors at time $t+1$ is implied by Clm. 13. This completes the induction argument and concludes the proof. ◀

Proof of Lem. 10. Denote the given protocol in the premise of Lem. 10 by Π. Lem. 9 proves that *wakeness vector updates* and *VDF chain extension* correctly implement valid wakeness vectors. Which in turns implies, due to the $(T_\mathrm{b}, T_\mathrm{b}, \rho)$-sleepy model, that Π is executed by every honest node by only considering messages from an honest majority at every timeslot. By the T-stateless property of Π, we thus have that Π solves the atomic broadcast problem correctly. Thus Π' inherits ϵ-Safety, and (ϵ, ℓ)-Liveness from Π. Protocol Π' decides blocks when Π decides them, thus the expected latency of Π' can not be worse than that of Π, and implementing *wakeness vector updates* and *VDF chain extension* requires an additive $O(\lambda n^2 \log^2 n)$ overhead in communication at every timeslot, as required. ◀

6 Additional related work

The early protocols designed for the synchronous sleepy model [24, 8, 2, 13] follow the longest chain approach of Nakamoto [23]. While inheriting its simplicity and elegance, they tend to suffer from high latency in the worst case. Furthermore, all of these protocols assume fixed and non-fluctuating participation of corrupt nodes. In this setting, the works of [3, 16] are geared at breaking the $\Omega(\frac{\lambda}{\gamma})$ latency barrier. Prism [3] designs a protocol whose latency is independent of λ under a certain optimistic condition, while [16] designs a protocol whose latency is independent of the participation rate γ (see Tab. 1 for the concrete bounds). Momose and Ren [22] designed the first protocol to achieve near optimal latency, namely $O(\Delta)$ in expectation. This was followed up by [21] in which the latency was further improved, as well as modifying the protocol to work in the *growing* participation setting [9]. The protocols of [22, 21] differ from the earlier protocols in the sleepy model, by emulating the PBFT [6, 28] approach to consensus, as opposed to longest chain based techniques.

Adversary restrictions. On the road to improve latency and supporting fluctuating participation of corrupt nodes, many works have imposed restrictions on adversary capabilities. The early work of [4] designs a protocol that can be instantiated in the fully-fluctuating sleepy model which is secure only against crash failures, with the *early decision* property. Namely, the latency of the protocol is proportional to the *actual* number of crash failures in the network. The work of [14] designs a $O(\Delta)$ expected latency protocol in the sleepy model that remains secure against fully-fluctuating participation, but restricts the adversary to only send time-stamped messages, i.e., each message sent by the adversary includes the *real* timeslot in which it was sent. In particular, this implies that the adversary is not allowed to equivocate (send conflicting messages to honest nodes in a given timeslot). The work of [18] considers a model where honest nodes know whether or not they will be awake or asleep in the next timeslot, and as such may use this information during the execution of the protocol (e.g., to announce their impending sleepiness to the rest of the nodes). The paper of [11] does not discuss fully-fluctuating participation of corrupt nodes, but their

protocol, when instantiated in the external adversary model we introduce in this paper, is secure against fully-fluctuating participation, and makes use of a VDF [5]. An early version of [21] showcased a protocol that supports fully-fluctuating participation and has optimal latency. They assume authenticated channels (i.e., identity of message sender is attached to each message), and further assume that messages that do not get delivered after Δ timeslots (because the recipients are asleep) get dropped from the network and never get delivered. This essentially prevents the adversary from performing forward simulation attacks.

Permissionless setting. In this paper we consider the setting where the total set of nodes allowed to participate in the protocol is fixed and known to all nodes. The work of Lewis-Pye and Roughgarden [20] explores a generalized version of the sleepy model in which the set of allowed participants in the protocol changes over time (via stake), and their identities are not known in advance to nodes.

PoSAT. It is worth noting that an existing consensus protocol is already resilient against fully-fluctuating node participation, under our external-adversary model! Specifically, despite the model not being discussed or defined directly in the paper, the PoSAT protocol [11], which makes use of time-based cryptography (specifically, VDFs) to achieve consensus in the growing participation setting, can be proven secure in the fully-fluctuating participation setting against an external adversary. Both the resilience and latency of the protocol are sub-optimal, however, with resilience $\rho = \frac{1}{1+e} \simeq 0.268$ and latency $O(\frac{1}{\gamma}\lambda\Delta)$, where γ is the participation rate of nodes, i.e., $\frac{1}{\gamma} > 1$, and λ is the security parameter. Our protocol obtains optimal resilience $\frac{1}{2}$ and expected constant latency, respectively. See Sec. 5 for details.

References

1. Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solida: A blockchain protocol based on reconfigurable byzantine consensus. In *OPODIS*, volume 95 of *LIPIcs*, pages 25:1–25:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.OPODIS.2017.25.
2. Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *CCS*, pages 913–930. ACM, 2018. doi:10.1145/3243734.3243848.
3. Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *CCS*, pages 585–602. ACM, 2019. doi:10.1145/3319535.3363213.
4. Ziv Bar-Joseph, Idit Keidar, and Nancy A. Lynch. Early-delivery dynamic atomic broadcast. In *DISC*, volume 2508 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2002. doi:10.1007/3-540-36108-1_1.
5. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO (1)*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018. doi:10.1007/978-3-319-96884-1_25.
6. Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002. doi:10.1145/571637.571640.
7. Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: efficient leaderless byzantine consensus and its application to blockchains. In *NCA*, pages 1–8. IEEE, 2018. doi:10.1109/NCA.2018.8548057.
8. Phil Daian, Rafael Pass, and Elaine Shi. Snow White: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography*, volume 11598 of *Lecture Notes in Computer Science*, pages 23–41. Springer, 2019. doi:10.1007/978-3-030-32101-7_2.

9 Francesco D'Amato, Joachim Neu, Ertem Nusret Tas, and David Tse. Goldfish: No more attacks on ethereum?! In *FC (1)*, volume 14744 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2024. doi:10.1007/978-3-031-78676-1_1.

10 Francesco D'Amato and Luca Zanolini. Streamlining sleepy consensus: Total-order broadcast with single-vote decisions in the sleepy model. *CoRR*, abs/2310.11331, 2023. doi:10.48550/arXiv.2310.11331.

11 Soubhik Deb, Sreeram Kannan, and David Tse. Posat: Proof-of-work availability and unpredictability, without the work. In *Financial Cryptography (2)*, volume 12675 of *Lecture Notes in Computer Science*, pages 104–128. Springer, 2021. doi:10.1007/978-3-662-64331-0_6.

12 Yuval Efron and Ertem Nusret Tas. Dynamically available common subset. Cryptology ePrint Archive, Paper 2025/016, 2025. URL: https://eprint.iacr.org/2025/016.

13 Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. Cryptology ePrint Archive, Paper 2018/1119, 2018. URL: https://eprint.iacr.org/2018/1119.

14 Eli Gafni and Giuliano Losa. Brief announcement: Byzantine consensus under dynamic participation with a well-behaved majority. In *DISC*, volume 281 of *LIPIcs*, pages 41:1–41:7. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.DISC.2023.41.

15 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015. doi:10.1007/978-3-662-46803-6_10.

16 Vipul Goyal, Hanjun Li, and Justin Raizes. Instant block confirmation in the sleepy model. In *Financial Cryptography (2)*, volume 12675 of *Lecture Notes in Computer Science*, pages 65–83. Springer, 2021. doi:10.1007/978-3-662-64331-0_4.

17 Vincent Gramoli. From blockchain consensus back to byzantine consensus. *Future Gener. Comput. Syst.*, 107:760–769, 2020. doi:10.1016/J.FUTURE.2017.09.023.

18 Pankaj Khanchandani and Roger Wattenhofer. Byzantine agreement with unknown participants and failures. In *IPDPS*, pages 952–961. IEEE, 2021. doi:10.1109/IPDPS49936.2021.00104.

19 Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

20 Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus. *CoRR*, abs/2304.14701, 2023. doi:10.48550/arXiv.2304.14701.

21 Dahlia Malkhi, Atsuki Momose, and Ling Ren. Towards practical sleepy BFT. In *CCS*, pages 490–503. ACM, 2023. doi:10.1145/3576915.3623073.

22 Atsuki Momose and Ling Ren. Constant latency in sleepy consensus. In *CCS*, pages 2295–2308. ACM, 2022. doi:10.1145/3548606.3559347.

23 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: https://bitcoin.org/bitcoin.pdf.

24 Rafael Pass and Elaine Shi. The sleepy model of consensus. In *ASIACRYPT (2)*, volume 10625 of *Lecture Notes in Computer Science*, pages 380–409. Springer, 2017. doi:10.1007/978-3-319-70697-9_14.

25 Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2018. doi:10.1007/978-3-319-78375-8_1.

26 Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980. doi:10.1145/322186.322188.

27 Srivatsan Sridhar, Ertem Nusret Tas, Joachim Neu, Dionysis Zindros, and David Tse. Consensus under adversary majority done right. In *Financial Cryptography*, 2025. URL: https://eprint.iacr.org/2024/1799.

28 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356. ACM, 2019. doi:10.1145/3293611.3331591.

From Permissioned to Proof-of-Stake Consensus

Jovan Komatovic
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Andrew Lewis-Pye
London School of Economics (LSE), UK

Joachim Neu
a16z Crypto Research, New York, NY, USA

Tim Roughgarden
Columbia University, New York, NY, USA
a16z Crypto Research, New York, NY, USA

Ertem Nusret Tas
Stanford University, CA, USA

Abstract

This paper presents the first generic compiler that transforms any permissioned consensus protocol into a proof-of-stake permissionless consensus protocol. For each of the following properties, if the initial permissioned protocol satisfies that property in the partially synchronous setting, the consequent proof-of-stake protocol also satisfies that property in the partially synchronous and quasi-permissionless setting (with the same fault-tolerance): consistency; liveness; optimistic responsiveness; every composable log-specific property; and message complexity of a given order. Moreover, our transformation ensures that the output protocol satisfies accountability (identifying culprits in the event of a consistency violation), whether or not the original permissioned protocol satisfied it.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Security and privacy → Distributed systems security

Keywords and phrases Permissioned Consensus, Proof-of-Stake, generic Compiler, Blockchain

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.18

Related Version *See full version for appendix*: `https://eprint.iacr.org/2025/1139` [54]

Funding The work of JK and ENT was conducted in part while at a16z Crypto Research. The research of TR at Columbia University was supported in part by NSF awards CCF-2006737 and CNS-2212745. ENT was further supported by the Stanford Center for Blockchain Research and a Research Hub Collaboration agreement between Stanford University and Input Output Global Inc.

1 Introduction

1.1 Permissioned and Permissionless Consensus

Blockchain protocols are run by large collections of processes that must stay in sync about the current state of the protocol's virtual machine. Keeping a distributed network of processes in agreement in the face of failures, attacks, and a potentially unreliable communication network – the problem of *consensus* – is a hard problem, but one for which there is a large body of research developed over the past four-plus decades. And indeed, many major blockchain protocols achieve their guarantees by building on protocols and ideas developed in the 1980s and 1990s.

Beyond standing on the shoulders of giants, the rise of Internet-scale blockchain protocols has pushed the state-of-the-art of consensus protocols significantly, in two distinct directions. First, 20th-century consensus protocols were typically designed for the *permissioned* setting, in which the protocol is to be run by an a priori known and fixed set of processes (e.g., a bunch

© Jovan Komatovic, Andrew Lewis-Pye, Joachim Neu, Tim Roughgarden, and Ertem Nusret Tas;
licensed under Creative Commons License CC-BY 4.0

of dedicated servers bought by a corporation for that purpose). Blockchain protocols, starting with the Bitcoin protocol, typically aspire to run in the *permissionless* setting in which there is free entry and exit to the set of processes running the protocol (perhaps after the acquisition of a costly resource, such as the protocol's native cryptocurrency). Permissionless consensus is strictly harder than permissioned consensus, due to a combination of additional challenges, including sybil-resistance (most commonly addressed through proof-of-work or proof-of-stake), an ever-changing set of processes running the protocol, and the possibility of non-faulty processes periodically going offline.

Second, even after setting aside the challenges posed by the permissionless setting, the practical deployment of and competition between Internet-scale state machine replication protocols have fueled innovation in the design of permissioned consensus protocols. One good example is the recent rise of DAG-based consensus protocols, which differ from previous protocols in, among other things, their use of simultaneous block proposals by all validators to overcome the bottlenecks typical of "leader-based" protocols in the lineage of PBFT [25].[1] While the formal analysis of DAG-based protocols has been confined to the permissioned setting thus far [50, 79, 78, 8, 34], Sui is an example of a permissionless proof-of-stake (PoS) system using a DAG-based protocol in production (cf. [9]).

1.2 The Dream: A Generic Property-Preserving Compiler

This paper pursues the "automatic tech transfer" of innovations for permissioned consensus protocols to those for permissionless protocols. The holy grail would be a "compiler" that takes as input an arbitrary permissioned protocol and outputs an "equally good" permissionless version. Ideally, such a compiler would obviate the need for any bespoke work to extend the *design* of a permissioned protocol into a permissionless one, and similarly for its *analysis*. Given that we make no assumptions about how the initial permissioned protocol works – i.e., it is given only as a "black box" – such a compiler can only interact with the protocol by executing it directly. Similarly, the analysis of the compiler's output would be able to rely only on the fact that the initial permissioned protocol satisfies the desired properties and not, for example, on any particular design or analysis method by which those properties might be achieved.

Compromise #1: Restriction to the quasi-permissionless setting. The main result of this paper is indeed a "compiler" of the above type, but certain compromises are, or appear to be, unavoidable. First, "sufficiently permissionless" protocols inherently suffer from provable limitations that are not shared by permissioned protocols, and for this reason we focus on the quasi-permissionless setting. In more detail, Lewis-Pye and Roughgarden [62] classify the "permissionlessness of a protocol" – or more accurately, the maximum permissionlessness under which a protocol functions as intended – via a hierarchy with several levels. The permissioned setting is one extreme point of the hierarchy. The next-most restrictive setting is the *quasi-permissionless (QP)* setting; the assumption here, when specialized to proof-of-stake protocols, is that correct processes with a positive amount of locked-up stake are always active. (But unlike the permissioned setting, the set of such processes can be ever-changing, and correct processes without locked-up stake can be periodically inactive.) The next level in the hierarchy is the *dynamically available (DA)* setting in which, similar to the sleepy setting

[1] This feature is not unique to DAG-based protocols; earlier asynchronous consensus protocols – such as [23, 66] – also adopted simultaneous proposals as a fundamental design principle.

of [74], correct processes (including those with locked-up stake) may be periodically offline. Protocols that have even minimal consistency and liveness guarantees in the DA setting cannot, for example, be consistent under asynchrony, or accountable (even in synchrony), or optimistically responsive (even in synchrony) [70, 62, 69]. If the assumptions on the setting are strengthened from DA to QP, all of the properties are achievable [62]. As we are interested in a compiler that preserves properties like consistency and responsiveness, we confine our analysis of permissionless protocols to the QP setting.

Compromise #2: Property-specific analysis. The strongest-possible assertion that the compiler's (permissionless) output protocol shares all the desired properties of its (permissioned) input protocol would be some type of homomorphism between executions (in the spirit of e.g. [32]). Such a homomorphism would map an execution of the permissionless protocol to one or more executions of the permissioned protocol showing that, intuitively, whatever could go wrong in a permissionless execution could have already gone wrong in a permissioned execution. There are several seemingly insurmountable obstacles to achieving this goal. To spell out one just example, such a homomorphism would presumably establish a relationship between the units of participation in the permissionless protocol (e.g., staked coins in a PoS protocol) and those in the original permissioned protocol (e.g., individual processes). But in an execution of the permissionless protocol, coins might well be transferred back and forth between correct and Byzantine processes (even if "locked" for certain durations of the execution), which would seem to translate to an execution of the permissioned protocol with a mobile adversary. Such a homomorphism cannot hope to preserve any property that is achievable with a static adversary but unachievable with a mobile adversary.

Because of the seeming impossibility of a property-preserving mapping between executions of the permissioned and permissionless protocols, we instead prove that our compiler preserves (a long list of) specific properties using direct, property-specific arguments.

Compromise #3: Restrictions on the preservable properties. Because the set of processes in a permissioned protocol is fixed while that of a permissionless protocol must update periodically (e.g., to reflect changes in processes' stakes), and because the permissioned protocol is given only as a closed box, any implementation of the desired compiler must presumably execute the permissioned protocol repeatedly, updating the process set appropriately each time. We refer to each execution of the permissioned protocol as an *epoch*, and this approach to the compiler's design as *epoch-based*. The naive hope would then be that the assumed properties of the permissioned protocol hold in each epoch of the permissionless protocol's execution, and thus also for the overall execution of that protocol.

Neither part of this hope can be carried out without restrictions on the properties we aim to preserve. The first part of the hope breaks down for, for example, properties of the form "eventually, the execution satisfies some predicate φ." The issue here is that while every (infinite) execution of a permissioned protocol might eventually satisfy φ, there may be no finite epoch length for which subexecutions of the protocol are guaranteed to satisfy φ. The second part of the hope breaks down for properties that are not preserved by the "concatenation" of the executions of two consecutive epochs. For a trivial example, the property that "at most n distinct identifiers ever vote" will be true for a PBFT-style permissioned protocol with a fixed set of n processes, but will not generally be true for any corresponding permissionless protocol (for which there is an unbounded number of identifiers, any of which may acquire stake and vote at some point in an execution).

The hope, then, is that an epoch-based compiler might still be capable of preserving all of the specific properties that one would be interested in preserving. This brings us to the main result of the paper.

1.3 The Main Result

The primary contribution of this paper is a generic compiler from permissioned to proof-of-stake permissionless protocols in the quasi-permissionless and partially synchronous setting. The compilation process is well defined for any permissioned protocol. For each of the following properties, if the initial permissioned protocol satisfies that property (in the partially synchronous setting), then the consequent proof-of-stake protocol also satisfies that property (in the partially synchronous and quasi-permissionless setting, and with the same fault-tolerance): consistency; liveness with respect to a liveness (latency) parameter ℓ;[2] optimistic responsiveness; and message complexity of a given order. Furthermore, our construction preserves all composable log-specific safety properties – that is, properties defined solely over logs (the outputs of processes; cf. Sec. 2) and preserved under concatenation (cf. the discussion in the "Compromise #3" paragraph above): roughly speaking, if two consistent logs L_1 and L_2 each satisfy such a property, then their concatenation $L_1 \| L_2$ also satisfies it. Finally, our compiler guarantees that the output protocol satisfies accountability [62, 30, 48, 31, 76, 70, 71] – that is, in the event of a consistency violation, the responsible parties can be correctly identified – regardless of whether the original permissioned protocol possessed this property. For example, applying our transformation to existing permissioned DAG-based protocols (e.g., [50, 79, 78, 8, 34]) yields new quasi-permissionless DAG-based protocols that had not previously been formally analyzed or proven correct.[3]

1.4 Why a Generic Transformation Was Needed

Limitations of existing practical approaches. Several blockchain systems in practice – such as Sui, Aptos, Cosmos, and Celo – already employ partially synchronous, quasi-permissionless consensus protocols. These systems typically adapt or build upon well-known permissioned protocols like HotStuff [84] (in the case of Aptos and Celo), Tendermint [18] (in the case of Cosmos), or Mysticeti [9] (in the case of Sui). While these protocols achieve significant practical performance and are deployed at scale, they rely on *protocol-specific modifications* and often embed *ad-hoc reconfiguration mechanisms* to work in PoS/QP settings. In particular, they are designed and analyzed in a bespoke manner – each new protocol or setting requires revisiting core design choices, reestablishing safety and liveness guarantees, and adapting reconfiguration logic to ensure correctness. To date, there exists *no general-purpose method* for converting an arbitrary permissioned protocol into a quasi-permissionless PoS protocol with well-defined theoretical guarantees.

Theoretical efforts & their shortcomings. Prior theoretical work has investigated how to adapt specific permissioned consensus protocols to PoS or quasi-permissionless settings. However, these approaches, like their practical counterparts, are fundamentally *protocol-specific* and rely on tightly coupled mechanisms that are not general. For example, Lewis-Pye and Roughgarden [62] construct a quasi-permissionless, PoS version of HotStuff [84] by embedding specialized reconfiguration mechanisms directly into the protocol. Budish *et al.* [19] similarly transform Tendermint [18] into a quasi-permissionless, PoS protocol, but

[2] The liveness parameter is passed as input to the compiler and the epoch length of the output protocol will depend on its value (cf. Sec. 4).

[3] To the best of our knowledge, Sui Lutris [16] is the only DAG-based protocol that has been formally analyzed in the quasi-permissionless setting, explicitly addressing the challenge of validator changes through a reconfiguration mechanism.

again this requires bespoke changes tailored to that specific protocol. Sui Lutris [16] is yet another protocol-specific design, featuring a custom hybrid architecture with partial and total ordering, and an ad-hoc epoch-based reconfiguration mechanism. Other systems, such as Hybrid Consensus [73] and PaLa [26], proceed in epochs and incorporate reconfiguration, but are built from the ground up with specific assumptions and analysis for a given protocol architecture. More broadly, reconfiguration – the task of updating the set of validators – has been widely studied [40, 39, 72, 14], but existing solutions embed reconfiguration logic deeply within the protocol's internal machinery. None of these approaches offer a generic, reusable method for reconfiguration mechanisms or for achieving quasi-permissionless PoS protocols.

Why a generic transformation is needed. This disconnect – between *theory*, which mainly focuses on permissioned consensus, and *practice*, which needs permissionless consensus – is precisely what motivates our work. We present the first *generic, closed-box transformation* that converts any partially synchronous permissioned protocol into a quasi-permissionless PoS protocol, while preserving a rich set of properties: consistency, liveness, optimistic responsiveness, accountability, and all composable log-specific safety properties. Our transformation separates the consensus core from reconfiguration logic, enabling a clean and composable analysis. This modularity provides two key benefits:

- For theorists, it allows continued focus on designing and analyzing permissioned protocols, while ensuring their results can be lifted automatically to the permissionless world.
- For practitioners, it offers a principled and reusable path to deploying proven permissioned protocols in permissionless environments, without redesigning reconfiguration or re-establishing correctness from scratch.

Our transformation acts as a "bridge lemma" between permissioned and permissionless worlds, making decades of theoretical work directly applicable to modern PoS systems.

Why designing this transformation was challenging. Naively running an arbitrary permissioned protocol as a subroutine within a PoS protocol – restarting it periodically with a new set of processes reflecting the latest stake amounts – fails to preserve even the most basic properties. Next, we highlight key technical challenges in designing and analyzing our generic compiler, along with the ideas that contribute to our solution.

Quit-enhanced permissioned protocols. At a high level, our compiler follows an epoch-based approach, executing the given permissioned protocol in each epoch for a finite duration. Therefore, the first step in our approach is to analyze, in a general manner, the behavior of permissioned protocols when processes are allowed to quit executing the protocol (which corresponds to the end of an epoch). The challenge here is the fact that we know nothing about how the given permissioned protocol might work, and the worry is that interfering with its execution (e.g., making one process inactive so that a different process can take its place) could affect its properties in unpredictable ways (e.g., with the newly inactive process now viewed as Byzantine by the protocol, violating its assumed fault-tolerance). For instance, consider adapting the Tendermint [18] permissioned consensus protocol to a quasi-permissionless setting using our epoch-based approach. Recall that Tendermint, being a permissioned protocol, assumes that all correct processes participate in the protocol forever, i.e., they never quit executing the protocol. As a result, all of its proven guarantees are built on this crucial assumption. However, when we attempt to transfer Tendermint into a quasi-permissionless setting using our epoch-based approach, this assumption no longer holds. Specifically, if the transition from epoch e to epoch $e + 1$ takes place before the network has stabilized (i.e., before GST; cf. Sec. 2 for the definition of GST), it is possible that all correct

processes except one stop participating in the Tendermint instance associated with epoch e. Therefore, the remaining correct process finds itself isolated, surrounded only by adversarial processes, with no support from other correct participants. This leads to a critical question: can consistency still be preserved in epoch e under such circumstances? In particular, is there a risk that the abandoned correct process could be misled by adversarial processes into violating consistency? Notably, this case is *not* covered by the original Tendermint analysis, as it falls outside the boundaries of the permissioned model assumed in [18], where such an abandoned correct process is never considered. Thus, analyzing the consistency (and other properties) of the resulting quasi-permissionless protocol requires going beyond the guarantees provided by the original Tendermint permissioned protocol, as those guarantees no longer directly apply in this new setting.

To address this challenge, we proceed as follows: (1) we extend the interface of Tendermint – though the idea applies to any permissioned protocol – to allow correct processes to stop participating, and (2) we analyze the guarantees provided by this extended protocol. We refer to these enriched versions of permissioned protocols as *quit-enhanced* permissioned protocols. Given any standard permissioned protocol, in which correct processes are expected to participate forever, the quit-enhanced version behaves identically in terms of internal logic but explicitly permits correct processes to stop executing the protocol. Thus, quit-enhancing a standard permissioned protocol constitutes a closed-box transformation: only the interface is extended, while the internal mechanisms remain unchanged (and untouched). Importantly, quit-enhanced permissioned protocols integrate naturally with our epoch-based structure: the properties of quit-enhanced permissioned protocols extend directly to our setting. With this in mind, we prove that quit-enhanced versions of arbitrary permissioned protocols preserve the key properties of interest (such as consistency and liveness) originally established in the standard permissioned model, and are thus suitable for use within an epoch-based compiler.

Preserving consistency. Even the basic property of consistency will not be preserved in an epoch-based approach to a generic compiler without carefully designing how one epoch transitions into the next. For example, consider a permissioned protocol whose algorithm first tentatively confirms a block – such as upon receiving an initial quorum certificate – and only later finalizes it, either through a follow-up quorum certificate or because the block is extended by blocks that become finalized. (It is important to emphasize that this tentative confirmation is an internal step within the algorithm's operation and does not represent the block's external status.) If an epoch-ending block B is only tentatively confirmed, who is then allowed to extend that block? Which processes should constitute the new validator set? We cannot allow the validator set for the next epoch to be determined based on the transactions in block B and its ancestors, with new validators immediately proposing descendants of B, because B has not been finalized. Since another conflicting block B' might also be tentatively confirmed, this could result in differing views on which validator set should be chosen for the next epoch. If we simply wait for the first directly finalized block and treat it as the genesis block of the next epoch, a key question remains: How can we unambiguously identify which block was directly finalized first?

To overcome this challenge, we introduce the notion of an *epoch-ending block* (akin to epoch transition in [73]). When the validators of an epoch e determine that the time has come to end the epoch (the conditions for this are discussed in the "Preserving liveness" paragraph below), they issue special epoch-ending transactions. They then wait for the first finalized block that, along with its ancestors, finalizes epoch-ending transactions from at least a quorum of the epoch's validators. This uniquely determined block becomes the epoch-ending block and serves as the genesis block for the next epoch. (Any data required to prove finality, such as blocks produced after the epoch-ending one, is retained in the protocol's history but does not contribute to the transactions finalized in that epoch.)

Preserving liveness. Preserving liveness (and optimistic responsiveness, a strengthening of liveness) presents an orthogonal set of challenges. For example, in the partially synchronous model, epoch lengths are most sensibly denominated in blocks (rather than time). The assumed liveness parameter ℓ of the underlying permissioned protocol – stating that newly issued transactions are finalized within ℓ time after the network stabilizes – is, however, defined in terms of time steps. Naturally, this is a desirable property that we aim to preserve. The issue is then that, if epochs complete in fewer than ℓ timesteps (due to an unexpectedly fast network) and every epoch has a fresh set of new validators, there is no guarantee that a given transaction will ever be finalized.

Our compiler addresses this challenge by ensuring that all correct validators overlap in each epoch (after the network stabilizes) for at least ℓ time, which allows for all new transactions to be finalized. To enforce this, each validator of an epoch e locally measures a period of $\ell + \Delta$ time, where Δ denotes the known bound on message delays after the network stabilizes (i.e., after GST; cf. Sec. 2). Given that messages propagate within Δ time, this local timing ensures that correct validators overlap for at least ℓ time during the epoch e (assuming the epoch takes place after stabilization). After this $\ell + \Delta$ interval elapses, a validator knows the epoch has run long enough and issues an epoch-ending transaction (as previously explained in the "Preserving consistency" paragraph). Because each epoch concludes with the finalization of an epoch-ending block – one that includes epoch-ending transactions from a quorum of validators, and thus from at least one correct validator – it is guaranteed that every epoch following network stabilization runs long enough to ensure finalization of new transactions, in accordance with the ℓ-liveness property of the underlying permissioned protocol.

Preserving composable log-specific safety properties. As discussed earlier, the epoch-based approach of our compiler necessitates focusing on properties whose satisfaction in every finite prefix of an (infinite) execution guarantees satisfaction in the entire execution – these are known as *safety* properties [5]. Consistency and optimistic responsiveness are examples of such safety properties. In contrast, eventual liveness is not a safety property, though liveness with respect to a fixed time bound ℓ on time-to-finality is.

This raises the question: how broad is the class of safety properties we can hope to preserve? Given the epoch-based approach, we must limit ourselves not only to safety properties but to those that are "closed under concatenation". To formalize this, we focus on *log-specific* properties – predicates that depend solely on the validators' running logs of finalized transactions (and not on, say, the precise sequence of messages that led to the creation of those logs). For log-specific properties, "safety" then means that a violation of the property must be evident from a finite-length prefix of (possibly unbounded-length) logs, and "composable" means that the property is preserved under unions of sets of logs. A canonical example of a composable log-specific safety property is an external validity property, such as "every finalized transaction is accompanied by appropriate signatures". We prove that our transformation preserves, simultaneously, every composable log-specific safety property.

2 System Model: Overview

We now provide an overview of the system model. A detailed description is in App. A in [54].

Processes, identifiers & adversary. We consider a (potentially infinite) set of processes denoted by Π. Each process $p \in \Pi$ is assigned a non-empty and potentially infinite set of *identifiers*, denoted by $\mathsf{id}(p)$. Intuitively, $\mathsf{id}(p)$ determines the set of public keys for which process p knows the corresponding private key. We denote by IDs the set of all identifiers.

Moreover, each process may or may not be *active* at each timeslot. A *process allocation* is a function specifying, for each process $p \in \Pi$, the timeslots at which process p is active. To accommodate for clock drifts, our model permits processes to be idle even at timeslots at which they are active. Concretely, at each timeslot at which a process is active, the process can either be *waiting* or *not waiting*. Whether a process is waiting or not at a specific timeslot is also determined by the process allocation function. In this work, we focus on protocols assuming a public key infrastructure (PKI) that allows processes to sign their messages and verify messages received from other processes.[4] Finally, we assume a static adversary that corrupts a fraction of all processes at the beginning of each execution. A corrupted process is said to be *faulty*; a non-faulty process is said to be *correct*.

Environment. There exists an *environment* that sends *transactions* to active and non-waiting processes. If the environment sends a transaction to an active and non-waiting process p at a timeslot τ, then p receives the transaction at the timeslot τ (along with messages sent by other processes).

Communication. We assume a message-passing model in which processes communicate by exchanging messages. In particular, any process can *send* a message directly to another process via point-to-point communication. Additionally, we assume the existence of a *gossip* primitive that enables a process to broadcast a message to all processes in the system. This gossip mechanism is "global" – it is not restricted to a specific subset of processes, and messages are intended for the entire network. An important assumption is that messages – whether sent directly or gossiped – are eventually delivered, even if the sender goes offline after sending. Such delivery guarantees are routinely achieved in practice, for instance in gossip networks used in blockchain systems.[5] We stress that our model assumes only guaranteed eventual delivery of messages. It does not preclude faulty processes from equivocating; that is, a faulty process may send different messages to different recipients. A detailed formal description of our network model is provided in App. A.2 in [54].

Partial synchrony. This work focuses on the standard partially synchronous model [41]. In a nutshell, there exists an unknown timeslot GST such that (1) the system behaves asynchronously before GST, and (2) the system behaves synchronously after GST with the known upper-bound Δ on message delays. Moreover, if any correct process p is active at any timeslot $\tau \geq \text{GST}$, then p is not waiting at timeslot τ, i.e., no clock drift occurs after GST. Lastly, each execution is associated with an unknown duration $\delta \leq \Delta$ that denotes the *actual* bound on message delays.

Logs & stake. A *log* is a non-empty ordered list of transactions. Given any log $\mathcal{L}$, the following methods are defined:

- $\mathcal{L}.\mathsf{length}$: the number of transactions in $\mathcal{L}$.
- $\mathcal{L}[i]$, for any $i \in [1, \mathcal{L}.\mathsf{length}]$: the i-th transaction of $\mathcal{L}$.

[4] We discuss how to extend our results in Sec. 5.

[5] We underline that this assumption is not required for dynamic systems with evolving membership. For instance, Carbon [24] avoids it and instead relies on a guaranteed message delivery only when both the sender and receiver remain online. Similar techniques could be employed in our setting to relax the requirement of delivery despite the sender going offline.

Two logs $\mathcal{L}_1$ and $\mathcal{L}_2$ are *consistent* if and only if $\mathcal{L}_1[i] = \mathcal{L}_2[i]$, for every i with $1 \leq i \leq \min(\mathcal{L}_1.\mathsf{length}, \mathcal{L}_2.\mathsf{length})$. Otherwise, the logs are *inconsistent*. Similarly, a log $\mathcal{L}_2$ *extends* a log $\mathcal{L}_1$ if and only if (1) logs $\mathcal{L}_1$ and $\mathcal{L}_2$ are consistent, and (2) $\mathcal{L}_1.\mathsf{length} \leq \mathcal{L}_2.\mathsf{length}$. If a transaction tr belongs to a log $\mathcal{L}$, we write "$\mathsf{tr} \in \mathcal{L}$". We denote by Logs the set of all logs.

Genesis & local log. Each execution is associated with a unique *genesis log* known to all processes. The genesis log generalizes the concept of a "genesis block" and acts as the initial log from which all subsequent logs are built.

Each process maintains its *local log*. Formally, each process p has a special log-register denoted by $\mathsf{log}(p)$. For every correct process p, $\mathsf{log}(p) = \mathcal{L}_g$ at timeslot 0, where $\mathcal{L}_g$ denotes the unique genesis log (of that specific execution). Given any correct process p and any timeslot τ, $\mathsf{log}(p, \tau)$ denotes the value in the $\mathsf{log}(p)$ register at timeslot τ. If $\mathsf{log}(p, \tau) = \mathcal{L}$, we say that p *outputs* $\mathcal{L}$ at timeslot τ.

Stake. Every log defines its *stake distribution*. Formally, there exists a function $\mathsf{S} : \mathsf{Logs} \times \mathsf{IDs} \to \mathbb{N}_{\geq 0}$. Intuitively, stake refers to each identifier's amount of on-chain resources. For each log $\mathcal{L} \in \mathsf{Logs}$, we define its *total stake*:

$$\mathcal{L}.\mathsf{total_stake} = \sum_{id \in \mathsf{IDs}} \mathsf{S}(\mathcal{L}, id).$$

We assume that, for each log $\mathcal{L} \in \mathsf{Logs}$, $\mathcal{L}.\mathsf{total_stake} > 0$. Moreover, we set the following restriction on the considered stake function $\mathsf{S}(\cdot, \cdot)$:

$$\forall (\mathcal{L}_1, \mathcal{L}_2) \in \mathsf{Logs}^2 : \mathcal{L}_1.\mathsf{total_stake} = \mathcal{L}_2.\mathsf{total_stake}.$$

Given this restriction, let $\mathbb{T}$ denote the total stake, i.e., $\mathbb{T} = \mathcal{L}.\mathsf{total_stake}$, for every $\mathcal{L} \in \mathsf{Logs}$. Importantly, we require protocols to be agnostic to the stake distribution function: given *any* stake distribution function $\mathsf{S}(\cdot, \cdot)$ satisfying the conditions above, the protocol must meet its specification to be deemed correct.

Permissioned setting. Here, the set of processes Π is finite and known. Additionally, Π's cardinality n is known. Moreover, each process has a single identifier: $\forall p \in \Pi : \mathsf{id}(p) = \{p\}$. Finally, each process is active at every timeslot.

Static ρ-bounded adversary. A static ρ-bounded adversary, for any $\rho \in [0, 1]$, corruptes at most $\rho \cdot n$ processes at the beginning of each execution.

Known vs. unknown facts. The following facts are known to processes:

- the set of processes Π, its cardinality, and the identifier function $\mathsf{id}(\cdot)$;
- the bound on the power of the adversary ρ, the stake distribution function $\mathsf{S}(\cdot, \cdot)$, the genesis log, and the upper-bound Δ on message delays;
- the process allocation function as every process is active at every timeslot.

In contrast, the following facts are unknown to processes:

- the set of corrupted processes, its cardinality, and GST.

Quasi-permissionless setting. In the quasi-permissionless setting, the set of processes Π is not necessarily finite. Moreover, processes might have more than a single associated identifier. In the quasi-permissionless setting, only processes with non-zero stake are guaranteed to be active. Specifically, for any timeslot τ and any correct process p for which there exist a τ-active correct process q and an identifier $id_p \in \mathsf{id}(p)$ with $\mathsf{S}(\mathsf{log}(q, \tau), id_p) > 0$, process p is active at τ.

Static ρ-bounded adversary. Intuitively, a static ρ-bounded adversary cannot control more than a ρ fraction of the total stake. Formally, for every correct process p and every timeslot τ, at most ρ fraction of $\mathsf{log}(p, \tau)$'s total stake ($\mathsf{log}(p, \tau).\mathsf{total_stake} = \mathbb{T}$) belongs to identifiers associated with faulty processes according to the stake distribution specified by $\mathsf{log}(p, \tau)$.

Known vs. unknown facts. The following facts are known to processes:

- the bound on the power of the adversary ρ, the stake distribution function $\mathsf{S}(\cdot, \cdot)$, and the genesis log;
- the upper-bound on message delays Δ.

The following facts are unknown:

- the set of processes Π, its cardinality, and the identifier function $\mathsf{id}(\cdot)$;
- the set of corrupted processes, its cardinality, and GST;
- the process allocation function.

3 Consensus Properties

This section outlines the consensus properties we aim to translate from the permissioned to the quasi-permissionless setting. These properties are divided into two categories: (1) core properties (Sec. 3.1), including consistency, liveness, optimistic responsiveness, and accountability, and (2) composable log-specific safety properties (Sec. 3.2).

3.1 Core Properties

We begin by defining the *core properties* of (permissioned or quasi-permissionless) consensus protocols, which are found in (almost) all of them.

Consistency. Intuitively, consistency guarantees that logs of correct processes never diverge.

▶ **Definition 1** (Consistency). *A (permissioned or quasi-permissionless) protocol satisfies* consistency *if and only if the following two conditions hold:*

- *No roll-backs: For every correct process $p \in \Pi$ and every two timeslots $\tau_1, \tau_2 \in \mathbb{N}_{\geq 0}$ with $\tau_1 < \tau_2$, $\mathsf{log}(p, \tau_2)$ extends $\mathsf{log}(p, \tau_1)$.*
- *No divergence: For every pair of correct processes $(p_1, p_2) \in \Pi^2$ and every timeslot $\tau \in \mathbb{N}_{\geq 0}$, logs $\mathsf{log}(p_1, \tau)$ and $\mathsf{log}(p_2, \tau)$ are consistent.*

If a protocol satisfies the consistency property against a ρ-bounded static adversary, we say the protocol is *ρ-consistent*.

Liveness. The liveness property ensures that every transaction is finalized within a known time frame after GST.

▶ **Definition 2** (ℓ-Liveness). *A (permissioned or quasi-permissionless) protocol satisfies* ℓ-liveness *if and only if the following condition is satisfied for every timeslot $\tau \in \mathbb{N}_{\geq 1}$. Suppose the following holds:*

- *Let $\tau^* = \max(\tau, GST) + \ell$.*
- *Let a transaction tr be received by a correct process from the environment at some timeslot $\leq \tau$.*
- *Let p be any correct process active (and non-waiting) at a timeslot $\geq \tau^*$ and let τ_a denote the first timeslot $\geq \tau^*$ at which p is active and non-waiting.*

Then, $\mathsf{tr} \in \mathsf{log}(p, \tau_\mathrm{a})$.

If a protocol satisfies the ℓ-liveness property against a ρ-bounded static adversary, we say the protocol is *(ρ, ℓ)-live*.

Optimistic responsiveness. Informally, the optimistic responsiveness property guarantees that transactions are finalized at network speed whenever all processes are correct.

▶ **Definition 3** (ℓ_{or}-Responsiveness). *A (permissioned or quasi-permissionless) protocol satisfies* ℓ_{or}-responsiveness*, where* $\ell_{\text{or}} \in O(\delta)$*, if and only if the following condition is satisfied for every timeslot* $\tau \in \mathbb{N}_{\geq 1}$ *in every execution where all processes are correct. Suppose the following holds:*

- *Let* $\tau^* = \max(\tau, GST) + \ell_{\text{or}}$.
- *Let a transaction* tr *be received by a correct process at some timeslot* $\leq \tau$.
- *Let* p *be any correct process active (and non-waiting) at a timeslot* $\geq \tau^*$ *and let* τ_{a} *denote the first timeslot* $\geq \tau^*$ *at which* p *is active and non-waiting.*

Then, $\mathsf{tr} \in \mathsf{log}(p, \tau_{\text{a}})$.

If a protocol satisfies the ℓ_{or}-optimistic responsiveness property against a ρ-bounded static adversary, we say that the protocol is (ρ, ℓ_{or})-*responsive*. Let us briefly compare our definition of ℓ_{or}-responsiveness (Def. 3) and our definition of ℓ-liveness (Def. 2). As shown, $\ell_{\text{or}} \in O(\delta)$, where δ denotes the actual (and unknown) upper bound on message delays after GST (cf. Sec. 2), while ℓ may depend on the known upper bound Δ and does not need to reflect actual network delays. Thus, while a responsive protocol can finalize transactions at the speed of the network, a live protocol may do so slower, depending on conservative bounds. Importantly, we note that our transformation ensures responsiveness even in non-failure-free executions assuming that all processes do behave correctly after GST.

Accountability. Accountability is a property ensuring that, if consistency is ever violated, a sufficient number of faulty processes are conclusively identified through undeniable proofs of guilt. We begin by introducing the concept of a proof of guilt in quasi-permissionless consensus algorithms; this definition is inspired by that from [63].

▶ **Definition 4** (Proof of guilt). *Let* $\mathcal{P}$ *be any quasi-permissionless protocol, and let* $id \in \mathsf{IDs}$ *be any identifier. Consider a set of messages* $\mathcal{M}$*, each of which is signed by* id *(i.e., using the corresponding private key). The set* $\mathcal{M}$ *constitutes a* proof of guilt *for* id *with respect to* $\mathcal{P}$ *if and only if there exists no execution of* $\mathcal{P}$ *in which (1)* id *(i.e., the corresponding process) sends all the messages from* $\mathcal{M}$*, and (2)* id *(i.e., the corresponding process) is correct.*

We are ready to define the accountability property in quasi-permissionless algorithms. Recall that our transformation guarantees accountability in the resulting quasi-permissionless protocol, regardless of whether the original permissioned protocol satisfies it.[6]

▶ **Definition 5** (ρ_{a}-Accountability). *A quasi-permissionless protocol satisfies* ρ_{a}-accountability *if and only if the following condition holds in every execution where there exist correct processes* p *and* q*, and timeslots* $\tau_p \in \mathbb{N}_{\geq 1}$ *and* $\tau_q \in \mathbb{N}_{\geq 1}$ *such that* $\mathsf{log}(p, \tau_p)$ *is inconsistent with* $\mathsf{log}(q, \tau_q)$*. Let* $\mathcal{M}_p$ *(resp.,* $\mathcal{M}_q$*) be the set of all messages received by process* p *(resp.,* q*) by timeslot* τ_p *(resp.,* τ_q*). Let* $\mathcal{F}$ *be the set of identifiers for which a proof of guilt exists in* $\mathcal{M}_p \cup \mathcal{M}_q$*. Then, there must exist a correct process* z *and a timeslot* $\tau_z \in \mathbb{N}_{\geq 0}$ *such that the identifiers in* $\mathcal{F}$ *collectively hold at least a* ρ_{a}*-fraction of the total stake recorded in* $\mathsf{log}(z, \tau_z)$*:*

$$\sum_{id \in \mathcal{F}} \mathsf{S}(\mathsf{log}(z, \tau_z), id) \geq \rho_{\text{a}} \cdot \mathsf{log}(z, \tau_z).\mathsf{total_stake} = \rho_{\text{a}} \cdot \mathbb{T}.$$

[6] That is why we define accountability solely with respect to quasi-permissionless protocols.

Let us analyze the definition of the ρ_a-accountability property. The property is "triggered" when a consistency violation occurs, meaning that two correct processes p and q output inconsistent logs. In such a case, the definition ensures that p and q collectively hold enough information to correctly identify as faulty a set of participants (i.e., identifiers) whose stake represents at least a ρ_a-fraction of the total stake.

Accountability vs. consistency. Accountability and consistency are inherently at odds. A ρ-consistent protocol guarantees that consistency is preserved as long as the adversary controls at most a ρ-fraction of the total stake; consistency may be violated only if this threshold is exceeded. Ideally, such a protocol would also achieve ρ_a-accountability for some $\rho_a > \rho$, meaning that whenever consistency is violated, the protocol can identify a set of faulty processes holding at least a ρ_a-fraction of the total stake. In other words, if consistency is violated, the adversary controls more than a ρ-fraction of the stake – hence, we seek to hold accountable a stake weight that exceeds the consistency threshold.

3.2 Composable Log-Specific Safety Properties

In this subsection, we define *composable log-specific safety properties*, a generic class of properties we translate from the permissioned to the quasi-permissionless setting.

Definition. A *log-specific property* P is a function $P : \mathbb{P}(\mathsf{Logs}) \to \{true, false\}$, where $\mathbb{P}$ denotes the power set and Logs denotes the set of all logs (cf. Sec. 2). Next, we define log-specific safety properties.

▶ **Definition 6** (Log-specific safety property)**.** *A* log-specific safety property S *is a log-specific property with the following constraint:*

- *Let* $logs \subseteq \mathsf{Logs}$ *be any set of logs such that* $S(logs) = false$*. Then, there exists a finite subset* $logs' \subseteq logs$ *such that, for every set* $logs''$ *with* $logs' \subseteq logs''$*,* $S(logs'') = false$*.*

Intuitively, the constraint specifies that if a set of logs fails to satisfy S, there must exist a finite subset that also does not satisfy S, and none of its supersets satisfy S. We underline that Def. 6 follows the spirit of safety properties as defined by Alpern and Schneider in their seminal work [4]. We are now ready to define composable log-specific safety properties.

▶ **Definition 7** (Composable log-specific safety property)**.** *A log-specific safety property* S *is* composable *if and only if the following constraint holds:*

- *Let* $(logs_1, logs_2) \subseteq \mathsf{Logs}^2$ *be any pair of sets of logs such that* $S(logs_1) = S(logs_2) = true$*. Then, for every set* $logs \subseteq logs_1 \cup logs_2$*,* $S(logs) = true$*.*

In essence, a composable log-specific safety property S indicates that if two sets of logs satisfy S, then every subset of their union also satisfies S. Properties that require correct processes to output only valid logs (according to some predetermined validity condition) are composable log-specific safety properties. These include, for example, the following properties: (1) no log contains futile transactions, and (2) no log contains transactions not signed by their issuers.

Satisfying log-specific safety properties. Lastly, we define what it means for a protocol to satisfy a (composable or not) log-specific safety property. Fix any (permissioned or quasi-permissionless) protocol $\mathcal{P}$. Given any execution $\mathcal{E}$ of the protocol $\mathcal{P}$, let $\mathsf{logs}(\mathcal{E})$ denote the set of all logs held by correct processes in $\mathcal{E}$:

$$\mathsf{logs}(\mathcal{E}) \equiv \{\mathcal{L} \in \mathsf{Logs} \mid \exists p \in \Pi : p \text{ is correct} \wedge p \text{ outputs } \mathcal{L} \text{ in } \mathcal{E}\}.$$

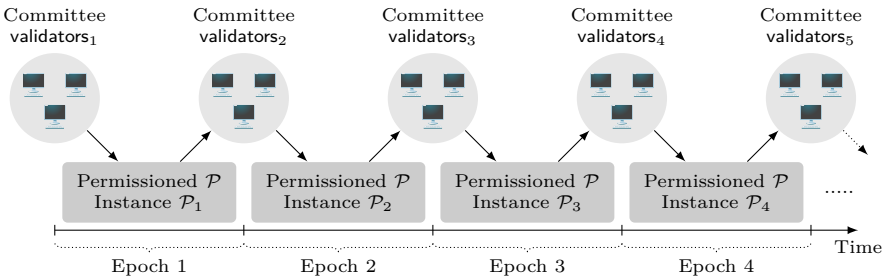


Figure 1 High-level overview of $\mathcal{T}(\mathcal{P})$'s epoch-based structure.

Finally, we present the definition.

▶ **Definition 8** (Satisfying log-specific safety properties). *Let S be any log-specific safety property and let $\mathcal{P}$ be any (permissioned or quasi-permissionless) protocol. We say that $\mathcal{P}$* satisfies *S if and only if, in every execution $\mathcal{E}$ with $S(\{\mathcal{L}_g\}) = true$, where $\mathcal{L}_g$ denotes the genesis log in $\mathcal{E}$, $S\big(\mathsf{logs}(\mathcal{E})\big) = true$.*

A protocol $\mathcal{P}$ satisfies a log-specific safety property S if correct processes output only logs that are allowed by S. Notably, we exclude executions in which the genesis log is not allowed by S; otherwise, no protocol could satisfy S as it is initially violated. We underline that time is irrelevant in the context of satisfying S: for any two (arbitrarily different) executions $\mathcal{E}_1$ and $\mathcal{E}_2$ that have the same set of output logs (i.e., $\mathsf{logs}(\mathcal{E}_1) = \mathsf{logs}(\mathcal{E}_2)$), the property S is either satisfied in *both* executions or in *neither*.

4 Transformation

In this section, we show how to transform any permissioned protocol $\mathcal{P}$ into a quasi-permissionless PoS protocol $\mathcal{T}(\mathcal{P})$. We begin by presenting an overview of our transformation (Sec. 4.1). Then, we introduce its pseudocode (Sec. 4.2). Finally, we provide an informal analysis of $\mathcal{T}(\mathcal{P})$'s correctness (Sec. 4.3). A formal proof is relegated to App. C in [54].

4.1 Overview

We partition the execution of the quasi-permissionless PoS protocol $\mathcal{T}(\mathcal{P})$ into *epochs* (Fig. 1). In each epoch, the permissioned protocol $\mathcal{P}$ is executed, meaning that every epoch is responsible for producing a particular segment of the "global" log of finalized transactions. Each epoch runs for (at least) a predetermined duration greater than the liveness parameter of protocol $\mathcal{P}$. This ensures that, following GST, each epoch runs for sufficiently long to finalize new transactions. Once an epoch completes, the log produced up to (and including) that epoch is treated as the genesis log for the next epoch. Moreover, the identifiers holding stake according to the produced log validate the next epoch; the genesis log $\mathcal{L}_g$ determines the validators of the first epoch. During the entire execution, protocol $\mathcal{T}(\mathcal{P})$ keeps progressing through epochs, with each subsequent epoch building on the output of the previous one, resulting in an ever-growing log of finalized transactions.

Guarantees of $\mathcal{P}$ in $\mathcal{T}(\mathcal{P})$'s epoch-based structure. The correctness of our quasi-permissionless PoS protocol $\mathcal{T}(\mathcal{P})$ crucially relies on the guarantees provided by the permissioned protocol $\mathcal{P}$. However, we observe that, within the epoch-based structure of $\mathcal{T}(\mathcal{P})$, protocol $\mathcal{P}$ might display unexpected behavior as described below. In the standard permissioned

setting, correct processes run $\mathcal{P}$ forever, as this is a fundamental characteristic of the setting. However, the epoch-based structure of $\mathcal{T}(\mathcal{P})$ introduces a different environment for $\mathcal{P}$: correct processes *stop* executing $\mathcal{P}$ associated with epoch e once they transition to epoch $e+1$. This minor change necessitates a re-evaluation of the guarantees offered by $\mathcal{P}$: $\mathcal{P}$'s guarantees in the standard permissioned setting do *not* directly extend to this new context.

To analyze $\mathcal{P}$'s behavior within the epoch-based structure of $\mathcal{T}(\mathcal{P})$, one needs to (1) enrich $\mathcal{P}$'s interface by allowing correct processes to stop executing it, and (2) examine the guarantees provided by this enriched protocol. We achieve this by introducing the concept of *quit-enhanced* permissioned protocols (cf. App. B.3 in [54]): given any standard permissioned protocol $\mathcal{P}'$, where correct processes are expected to participate forever, we define $\mathit{quit}(\mathcal{P}')$ as the quit-enhanced version of $\mathcal{P}'$, where correct processes have the option to stop participating. (We emphasize that $\mathit{quit}(\mathcal{P})$ merely extends the interface of the original permissioned protocol $\mathcal{P}$. In particular, analyzing the behavior of $\mathit{quit}(\mathcal{P})$ relies exclusively on understanding the behavior of $\mathcal{P}$ itself – there is no need to examine the internal mechanisms of the protocol. This makes our quit-based extension inherently "closed-box" in nature.) Using our concept of quit-enhanced protocols, we prove that consistency, liveness, optimistic responsiveness, and all log-specific safety properties translate from $\mathcal{P}$ to $\mathit{quit}(\mathcal{P})$ (cf. App. B.4 in [54]), allowing us to build $\mathcal{T}(\mathcal{P})$ on top of $\mathcal{P}$ that satisfies these properties in the permissioned model.[7]

4.2 Pseudocode

Transformation $\mathcal{T}$ takes a permissioned ρ-consistent protocol $\mathcal{P}$ satisfying liveness with parameter ℓ and a stake distribution function $\mathsf{S}(\cdot,\cdot)$ as input, and outputs a quasi-permissionless PoS protocol $\mathcal{T}(\mathcal{P})$ (cf. Fig. 2 and Alg. 1). Each epoch of $\mathcal{T}(\mathcal{P})$ is associated with a set of active identifiers, denoted by $\mathsf{validators}_e$. At the beginning of each epoch e, these identifiers execute an instance of $\mathcal{P}$ (ln. 24, ln. 31-ln. 90).

Executing $\mathcal{P}$ within an epoch e. As $\mathbb{T}$ denotes the total stake (cf. Sec. 2), identifiers utilized in (executed-in-an-epoch) permissioned protocol $\mathcal{P}$ are integers in the range $[1,\mathbb{T}]$; we refer to these identifiers as "permissioned-ids". Similarly, we refer to identifiers from the set IDs (cf. Sec. 2) as "PoS-ids".

Consider a correct process p executing $\mathcal{P}$ in epoch e. To start executing $\mathcal{P}$ in epoch e, process p invokes the $\mathsf{start_simulation}(\cdot)$ function (ln. 55).[8] There, process p begins by setting the current log of the PoS protocol $\mathcal{T}(\mathcal{P})$, finalized at the end of the previous epoch $e-1$, as the genesis log $\mathcal{L}$ for epoch e (ln. 56-ln. 60). It then maps each permissioned-id (from the $[1,\mathbb{T}]$ range) into one PoS-id (from the IDs set) using the $id_map_p = \mathsf{map_stake}(\mathcal{L})$ map (ln. 62). For instance, if $\mathcal{L}$ assigns 3 tokens to a PoS-id id, then there exist $x_1, x_2, x_3 \in [1,\mathbb{T}]$ such that $id_map_p[x_1] = id_map_p[x_2] = id_map_p[x_3] = id$; intuitively, in epoch e, permissioned-ids x_1, x_2 and x_3 correspond to the PoS-id id, meaning that PoS-id id is responsible for simulating $\mathcal{P}$'s state machines associated with permissioned-ids x_1, x_2 and x_3. This is also crucial because each process must know which identifier to use when forwarding messages related to the permissioned protocol $\mathcal{P}$. Specifically, if a permissioned-id i_1 needs to communicate with

[7] Liveness and optimistic responsiveness are preserved in only some (and not all) executions of $\mathit{quit}(\mathcal{P})$. However, as we show in App. C in [54], only these executions are needed to prove $\mathcal{T}(\mathcal{P})$'s liveness and optimistic responsiveness.

[8] Throughout the remainder of this section and in the pseudocode, we say that processes simulate $\mathcal{P}$ to emphasize that this is the underlying permissioned protocol being executed, and that each PoS-id may be running $\mathcal{P}$'s state machine on behalf of multiple permissioned-ids.

Algorithm 1 Permissioned to PoS transformation $\mathcal{T}$: Pseudocode for process p [part 1/2].

```
1  Inputs:
2      Permissioned ρ-consistent protocol 𝒫 with liveness parameter ℓ        ▷ to be transformed
3  Constants:
4      Log 𝓛_g                                                              ▷ the genesis log; cf. Sec. 2
5      Integer 𝕋                                                            ▷ the total stake; cf. Sec. 2
6  Local variables:
7      Log log(p) ← 𝓛_g                                                     ▷ local log of p, initially set to genesis log
8      Set(Transactions) received_txs_p ← ∅                                 ▷ set of received transactions
9      Epoch epoch_p ← 0                                                    ▷ the current epoch
10     Set(IDs) current_validators_p ← ∅                                    ▷ set of p's identifiers validating the current epoch
11     Map(Log → Boolean) signed_p ← {false, for every log 𝓛 ∈ Logs}

12 at every timeslot τ:
13     received_txs_p ← the set of transactions received by timeslot τ (from the environment and
       other processes)
14     invoke feed(received_txs_p)                                          ▷ forward the transactions to simulated 𝒫
15     ▷ The next line can be optimized: p can only gossip updates not previously gossiped.
16     ▷ For simplicity, we maintain the unoptimized pseudocode.
17     Gossip fully-certified log(p) and received_txs_p

18 upon start:
19     invoke start_simulation(log(p))                                      ▷ start simulating the first epoch; log(p) = 𝓛_g here

20 upon receiving a fully-certified log 𝓛 such that 𝓛 extends log(p):      ▷ hence, 𝓛.epoch ≥ epoch_p
21     log(p) ← 𝓛                                                           ▷ update the local log
22     if 𝓛.epoch > epoch_p or 𝓛.completed = true then                       ▷ check if new epoch should be started
23         invoke stop_simulation                                           ▷ if so, stop simulation associated with the previous epoch
24         invoke start_simulation(log(p))                                  ▷ and start simulation associated with the new epoch

25 upon obtain_log(Log 𝓛):                                                  ▷ 𝓛 is obtained from simulated 𝒫
26     for each id ∈ current_validators_p:
27         for each Log 𝓛′ with 𝓛′.epoch = 𝓛.epoch and 𝓛 extends 𝓛′ and signed_p[𝓛′] = false:
28             if no log inconsistent with 𝓛′ has previously been signed then
29                 Sign 𝓛′ using the private key of id and gossip 𝓛′ accompanied by the signature
30                 signed_p[𝓛′] ← true
```

another permissioned-id i_2 in $\mathcal{P}$ (associated with some epoch), the process p "responsible" for the permissioned-id i_1 must know a PoS-id corresponding to i_2 in order to forward the message correctly (ln. 87). After establishing the aforementioned permissioned-ids to PoS-ids mapping, process p verifies whether it is validating epoch e, i.e., whether any of its identifiers have been assigned a positive stake by $\mathcal{L}$. If there exists a permissioned-id $x \in [1, \mathbb{T}]$ that maps into a p's PoS-id $id \in \mathsf{id}(p)$, process p instantiates $\mathcal{P}$'s state machine initialized with permissioned-id x and genesis log $\mathcal{L}$ (lns. 64 and 65). Then, p updates $simulation_map_p[x]$ to the initialized state machine (ln. 67). Note that id_map_p associates each permissioned-id with its PoS-id counterpart, whereas $simulation_map_p$ maps only p's permissioned-ids into their respective state machines. Finally, if p is indeed validating epoch e, it instructs its timer $epoch_timer_p$ to measure $\mathsf{ED} = \ell + \Delta$ time (ln. 69). Here, $\mathsf{ED} = \ell + \Delta$ is selected large enough so that each simulated instance $\mathcal{P}$ is run sufficiently long to allow new transactions to be finalized after GST. Specifically, the time period $\mathsf{ED} = \ell + \Delta$ ensures that, after GST, all correct validators of some post-GST epoch e overlap in their execution of $\mathcal{P}$ (associated with epoch e) for at least ℓ time. This overlap, together with the ℓ-liveness property of $\mathcal{P}$, guarantees that new transactions are finalized.

Algorithm 1 Permissioned to PoS transformation $\mathcal{T}$: Pseudocode for process p [part 2/2].

```
▷ This section of the pseudocode is dedicated to simulating the permissioned protocol 𝒫.
Local variables:                                                  ▷ variables dedicated to simulating 𝒫
    Map([1,𝕋] → IDs) id_map_p ← empty map                         ▷ permissioned-id to PoS-id
    Map([1,𝕋] → Simulation) simulation_map_p ← empty map          ▷ permissioned-id to simulation
    Set(Simulation) simulations_p ← ∅                             ▷ set of p's currently executed simulations
    Timer epoch_timer_p                                           ▷ for measuring the duration of epochs

Local functions:
    Map([1,𝕋] → IDs) map_stake(Logs ℒ):                           ▷ returns permissioned-id to PoS-id mapping
        Map([1,𝕋] → IDs) map ← empty map
        List(IDs) V ← ℒ.current_ids                               ▷ find identifiers with positive stake according to ℒ
        Sort V in lexicographical order
        Integer counter ← 1
        for each id ∈ V:                                          ▷ iterate through V in the ascending lexicographical order
            for each j ∈ [1, S(ℒ, id)]:                           ▷ associate a number of permissioned-ids equal to the stake
                map[counter] ← id
                counter ← counter + 1
        return map

at every timeslot τ:
    if simulations_p ≠ ∅ then                                     ▷ check if p is validating the current epoch
        for each S ∈ simulations_p:
            Let ℒ ← log(S.identifier, τ)                          ▷ obtain the current log of the simulated instance S
            if ℒ.epoch > epoch_p then                             ▷ check if the current log is "too long"
                ℒ ← ℒ.ep_prefix(epoch_p)
            trigger obtain_log(ℒ)                                 ▷ report the current log

upon start_simulation(Logs ℒ):
    if ℒ.completed = true then
        epoch_p ← ℒ.epoch + 1                                     ▷ ℒ is the genesis log for the epoch
    else                                                          ▷ ℒ is a log from the "middle" of epoch
        epoch_p ← ℒ.epoch                                         ▷ the current epoch is ℒ's epoch
        ℒ ← ℒ.ep_prefix(epoch_p − 1)  ▷ (epoch_p − 1)-prefix of ℒ is the genesis log for the epoch
    current_validators_p ← ℒ.current_ids ∩ id(p)  ▷ set p's identifiers validating the current epoch
    id_map_p ← map_stake(ℒ)                                       ▷ update the permissioned-id to PoS-id map
    for each i ∈ [1,𝕋] such that id_map_p[i] ∈ id(p):
        Let simulation ← initialize 𝒫 with permissioned-id i      ▷ instance of 𝒫 with id i
        Start simulation with genesis log ℒ   ▷ the genesis log for 𝒫 with permissioned-id i is ℒ
        simulations_p ← simulations_p ∪ {simulation}
        simulation_map_p[i] ← simulation
    if current_validators_p ≠ ∅ then
        invoke epoch_timer_p.measure(ED = ℓ + Δ)                  ▷ if validating the epoch, start the timer

upon stop_simulation:
    for each S ∈ simulations_p:
        Stop executing 𝒮                                          ▷ stop each instance S of permissioned protocol 𝒫
    ▷ Reset the variables and cancel the timer
    invoke epoch_timer_p.cancel()
    id_map_p ← empty map
    simulation_map_p ← empty map
    simulations_p ← ∅
    current_validators_p ← ∅

upon epoch_timer_p expires:                                       ▷ the epoch should be completed, i.e., the timer has expired
    for each id ∈ current_validators_p:
        Gossip the (FINISH, epoch_p) transaction issued by id

upon S ∈ simulations_p sends a message m:                         ▷ simulation instance S sends m
    M ← ⟨SIMULATION, epoch_p, m⟩                                  ▷ tag m with the current epoch
    ▷ Send the simulation message to the PoS-id associated with the permissioned-id m.receiver
    Send M to id_map_p[m.receiver]

upon receiving a simulation message M with M.epoch = epoch_p and simulations_p ≠ ∅:
    Forward M.message to simulation_map_p[M.message.receiver]

upon feed(Set(Transaction) txs):
    for each S ∈ simulations_p:
        Forward transactions txs to S                             ▷ simulation instance S receives transactions txs
```

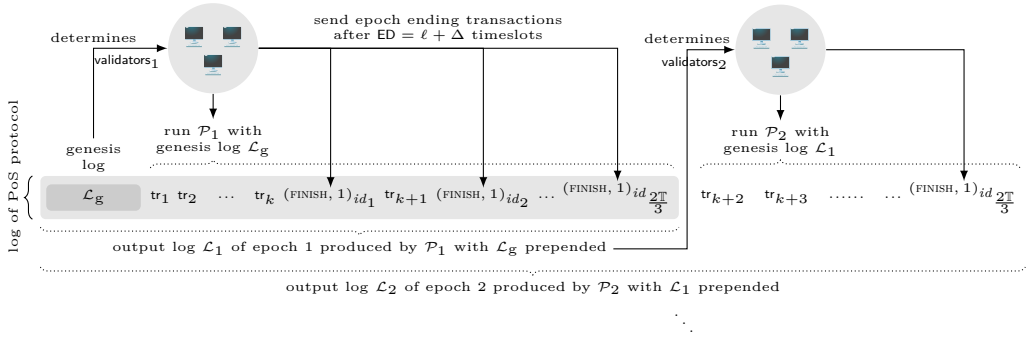


Figure 2 The quasi-permissionless PoS protocol $\mathcal{T}(\mathcal{P})$ proceeds in epochs (see Fig. 1). At each epoch e, a set of identifiers $\mathsf{validators}_e$ run a new instance $\mathcal{P}_e$ of the permissioned protocol $\mathcal{P}$; the genesis log for $\mathcal{P}_e$ is the log output in epoch $e-1$. $\mathsf{ED} = \ell + \Delta$ timeslots into the execution of $\mathcal{P}_e$, the validators issue a special epoch-ending transaction (FINISH, e). Once there are two-thirds-stake worth of finalized epoch-ending transactions, each validator stops executing the protocol $\mathcal{P}_e$. The log at that point determines the validators for the epoch $e+1$, and is input as the genesis log to $\mathcal{P}_{e+1}$.

Outputting logs. Within every epoch e, processes output a log produced by $\mathcal{P}$ as the $\mathcal{T}(\mathcal{P})$ log if the log is signed by any quorum of the set $\mathsf{validators}_e$ (ln. 20 and Def. 25 in [54]).

Epoch change. When *epoch_timer*$_i$ expires, process p gossips a special *epoch-ending* transaction (FINISH, e) on behalf of each of its validating PoS-ids (ln. 81). The purpose of these FINISH transactions is ensuring that epoch e eventually concludes by producing a complete log (cf. Def. 23 in [54]), i.e., an epoch-ending block.

Process p stops simulating $\mathcal{P}$ upon observing epoch-ending transactions from a quorum of identifiers with sufficient stake, i.e., some quorum $I \subseteq \{id | id \in \mathsf{validators}_e\}$ such that $\sum_{id \in I} \mathsf{S}(\mathcal{L}, id) \geq (1-\rho)\mathbb{T}$ (ln. 23 and Def. 23 in [54]). At that point, p stops each of its currently simulated state machines (ln. 72), cancels *epoch_timer*$_p$ (ln. 74), and resets the simulation-specific variables (ln. 75-ln. 78).

Determining the next set of validators. The set $\mathsf{validators}_{e+1}$ of validators is selected based on the stake distribution determined by the log produced in epoch e (ln. 62 and Def. 21 in [54]). More specifically, any identifier that has positive stake according to the log $\mathcal{L}$ produced in epoch e is eligible to participate in the permissioned protocol instance $\mathcal{P}$ associated with the next epoch $e+1$.

On leaving & joining validators. How do we ensure that all correct validators of epoch e eventually complete that epoch? And how do we ensure that all correct validators of epoch $e+1$ learn the up-to-date state of the system – that is, all blocks finalized up to epoch e? Both questions are resolved through the underlying communication model (cf. Sec. 2). Specifically, before any correct validator of epoch e departs, it gossips its current log of finalized blocks, which includes all blocks from epoch e (ln. 17). This log is fully certified, allowing any recipient to locally verify its validity and commitment status (see the following paragraph for details). This mechanism guarantees that every correct validator eventually receives the fully-certified log, finalizes the blocks of epoch e, and is thus permitted to exit. Similarly, joining validators are ensured to eventually receive the finalized log up to epoch e, enabling them to reconstruct the current system state and safely begin executing epoch $e+1$. Importantly, this mechanism allows us to avoid imposing any restrictions on validator departures: the sets of validators in epochs e and $e+1$ may be entirely disjoint.

Methods defined on logs. The pseudocode of our transformation $\mathcal{T}$ utilizes many methods defined on logs. Their formal definition can be found in App. C.1 in [54], while an informal description is given below:

- $\mathcal{L}$.current_ids: the set of identifiers with positive stake according to $\mathcal{L}$.
- $\mathcal{L}$.epoch: the epoch with which $\mathcal{L}$ is associated.
- $\mathcal{L}$.validators: the set of identifiers validating the epoch of $\mathcal{L}$, i.e., the set of identifiers that "produced" $\mathcal{L}$.
- $\mathcal{L}$.completed: *true* if and only if $\mathcal{L}$ contains sufficiently many (i.e. $(1-\rho)\mathbb{T}$ worth of stake) epoch-ending transactions.
- $\mathcal{L}$.ep_prefix$\big(e \in [0, \mathcal{L}.\mathsf{epoch})\big)$: the completed log $\mathcal{L}'$ of epoch e such that $\mathcal{L}$ extends $\mathcal{L}'$.

Moreover, we say that a log $\mathcal{L}$ is *certified* at a process p if and only if p receives signatures on the log $\mathcal{L}$ from a quorum of members (i.e. $(1-\rho)\mathbb{T}$ worth of stake) of $\mathcal{L}$.validators. Finally, a certified log $\mathcal{L}$ is *fully-certified* at a process p if and only if, for every $e \in [0, \mathcal{L}.\mathsf{epoch})$, $\mathcal{L}.\mathsf{ep_prefix}(e)$ is certified at process p. Note that two fully-certified logs cannot be conflicting. A log attains full certification only if it is approved by a quorum of validators for each epoch it covers. Since any two quorums intersect in at least one correct validator, it is impossible for two conflicting logs to both become fully certified. Consequently, upon receiving a fully-certified log, any correct validator can safely adopt it as its local log (ln. 21).

4.3 Correctness

In this subsection, we provide an informal analysis of the correctness of our transformation.

Consistency. The following theorem proves that our transformation preserves the consistency property from the permissioned to the quasi-permissionless setting.

▶ **Theorem 9** (Consistency)**.** *Consider a permissioned protocol $\mathcal{P}$ that satisfies consistency against a ρ-bounded adversary. Then, the quasi-permissionless protocol $\mathcal{T}(\mathcal{P})$ satisfies consistency against a ρ-bounded adversary.*

Proof sketch. We show that the consistency property translates from $\mathcal{P}$ to $\mathcal{T}(\mathcal{P})$ by induction. Initially, all correct processes have $\mathcal{L}_{\mathrm{g}}$ as the genesis log. Given that the logs output by the first epoch are consistent (due to $\mathcal{P}$'s consistency), consistency is satisfied in the first epoch. Moreover, all correct processes agree on the genesis log and the set of validators for the second epoch. By inductively applying the same argument, we can show that the logs output by the correct processes remain consistent throughout the entire execution. A formal proof of the theorem is given in App. C.3 in [54]. ◀

Composable log-specific safety properties. The theorem below asserts that our transformation carries over any composable log-specific safety property from the permissioned setting to the quasi-permissionless setting.

▶ **Theorem 10** (Composable log-specific safety property)**.** *Consider a permissioned protocol $\mathcal{P}$ that satisfies consistency and some composable log-specific safety property S against a ρ-bounded adversary. Then, $\mathcal{T}(\mathcal{P})$ satisfies S against a ρ-bounded adversary.*

Proof sketch. To prove that any composable log-specific safety property S translates from $\mathcal{P}$ to $\mathcal{T}(\mathcal{P})$, we again rely on induction. As $\mathcal{P}$ satisfies S, the set of logs output by the first epoch adheres to S. Similarly, the set of logs output by the second epoch adheres to S. Hence, their union adheres to S as S is composable (cf. Def. 7):

$$S(\{\mathcal{L} \mid \mathcal{L}.\mathsf{epoch} \in \{1,2\} \wedge \mathcal{L} \text{ is output by a correct process}\}) = \mathit{true}.$$

Again, the set of logs output by the third epoch adheres to S, which then implies:

$$S(\{\mathcal{L} \mid \mathcal{L}.\mathsf{epoch} \in \{1,2,3\} \wedge \mathcal{L} \text{ is output by a correct process}\}) = \textit{true}.$$

By inductively applying this argument, we can show that $\mathcal{T}(\mathcal{P})$ satisfies S. A formal proof of the theorem is given in App. C.4 in [54]. ◀

Liveness & optimistic responsiveness. The following theorem demonstrates the preservation of liveness and optimistic responsiveness by our transformation.

▶ **Theorem 11** (Liveness & optimistic responsiveness). *Consider a permissioned protocol $\mathcal{P}$ that satisfies consistency and ℓ-liveness, for some $\ell < \infty$, against a ρ-bounded adversary. Then, $\mathcal{T}(\mathcal{P})$ satisfies $\ell^\star$-liveness, with $\ell^\star = 2\Delta + 2\ell$, against a ρ-bounded adversary. Furthermore, if $\mathcal{P}$ additionally satisfies ℓ_{or}-responsiveness, for some $\ell_{\text{or}} \in O(\delta)$, where δ denotes the actual bound on message delays after GST, against a ρ-bounded adversary, then $\mathcal{T}(\mathcal{P})$ satisfies $\ell^\star_{\text{or}}$-responsiveness, with $\ell^\star_{\text{or}} = 2\delta + 2\ell_{\text{or}}$, against a ρ-bounded adversary.*

Proof sketch. If the first correct process p for which there exists an identifier $id \in \mathsf{id}(p) \cap \mathsf{validators}_e$ enters an epoch e at some timeslot $\tau \geq \text{GST}$, then all other correct validators enter epoch e within Δ timeslots after observing the log seen by p. Moreover, no correct process sends an epoch-ending transaction until $\mathsf{ED} = \ell + \Delta$ timeslots into epoch e, which implies that epoch e cannot be completed before timeslot $\tau + \mathsf{ED} = \tau + \ell + \Delta$. Hence, all correct validators stay in epoch e together for at least ℓ timeslots. As $\mathcal{P}$ satisfies ℓ-liveness, this overlap is sufficient to finalize new transactions in epoch e. To prove optimistic responsiveness, we follow the previous argument while factoring in that the actual message delay bound is δ. As a result, all correct validators join epoch e within δ timeslots – rather than $\Delta > \delta$ – after seeing the log observed by p. A formal proof is relegated to App. C.5 in [54]. ◀

Accountability. Finally, the following theorem demonstrates that our transformation ensures that the resulting quasi-permissionless protocol satisfies accountability. Note that this holds even if the original permissioned protocol lacks accountability.

▶ **Theorem 12** (Accountability). *Consider a permissioned protocol $\mathcal{P}$ that satisfies consistency against a ρ-bounded adversary. Then, the quasi-permissionless protocol $\mathcal{T}(\mathcal{P})$ satisfies $(1-2\rho)$-accountability.*

Proof sketch. Suppose two correct processes p and q output inconsistent logs $\mathcal{L}_p$ and $\mathcal{L}_q$, respectively. Therefore, $\mathcal{L}_p$ (resp., $\mathcal{L}_q$) is fully-certified at p (resp., q). Hence, disseminating these two logs, along with their respective signatures, enables accountability: every correct process eventually receives these inconsistent logs, combines the received signatures, and obtains a set of identifiers C such that, for each $id \in C$, id signs two inconsistent logs associated with the same epoch, thus proving id's culpability. Finally, since each of the two logs is signed by a quorum of validators – i.e., those holding at least $(1-\rho)\mathbb{T}$ stake – the identifiers in C collectively represent at least $(1-\rho)\mathbb{T} + (1-\rho)\mathbb{T} - \mathbb{T} = (1-2\rho)\mathbb{T}$ stake. A formal proof of the theorem can be found in App. C.6 in [54]. ◀

In terms of message complexity, the transformation $\mathcal{T}(\mathcal{P})$ introduces an additional quadratic term to that of $\mathcal{P}$ in order to satisfy the accountability property in the quasi-permissionless setting, stemming from the signatures that processes must exchange before outputting a log. However, since any consensus protocol inherently requires a quadratic number of messages in the worst case [37], our transformation does not introduce any asymptotic worst-case overhead. A proof is deferred to App. C.7 in [54].

5 Extensions

We now present some promising directions that can further improve our transformation.

Beyond public key infrastructure. In this work, we assumed the use of digital signatures (cf. Sec. 2). However, many permissioned protocols rely on "heavier" cryptographic primitives (e.g., [84, 61, 29]) such as threshold signatures. Importantly, our transformation can easily be adapted for these protocols, provided that the cryptographic primitives necessary for the underlying permissioned protocol are established in each epoch. Concretely, any setup procedure necessary for the permissioned protocol being transformed should be treated as an integral *part* of the permissioned protocol itself. There exists a body of work on proactive secret sharing across dynamic committees (e.g., [64, 45, 11]), which can be useful for avoiding a fresh DKG (or trusted) setup and public parameters for each epoch. Finally, we note that, to defeat long range attacks [36], key-erasure techniques [27] can be employed on the PoS protocol, so that the processes that have become passive, even if corrupted in the future, cannot create a log for the past epochs in retrospect.

On transforming weighted permissioned protocols. For simplicity, we have presented our transformation assuming that the underlying permissioned protocol assigns unit weight per validator – that is, it does not support heterogeneous weights. Then, each PoS validator must run one "virtual" validator in the permissioned protocol *per unit of stake*. This "virtualization" can be inefficient, especially when large amounts of stake are concentrated among few stakeholders. Note, however, that our transformation applies equally well to weighted permissioned protocols, where each PoS validator simply runs a single permissioned validator with weight proportional to its stake. This avoids the overhead of "virtualization". Notably, many widely used permissioned protocols – such as Tendermint [18], PBFT [25], and HotStuff [84] – offer weighted variants, allowing our transformation to pass through the stake weights directly, and for the resulting PoS protocol to operate more efficiently.

On lotteries and incentive mechanisms. Our transformation is agnostic to the choice of the underlying permissioned protocol; as long as the protocol supports lotteries (e.g., probabilistic leader selection), the resulting PoS protocol naturally inherits this property. State updates that reward participating processes can be incorporated seamlessly, though such incentive functionality belongs to the "virtual machine" layer rather than the consensus mechanism itself. Importantly, Thm. 12 ensures that the PoS protocol produced by our compiler satisfies accountability. This, in turn, enables the implementation of slashing mechanisms – an essential aspect of cryptoeconomic incentive design.

On establishing the EAAC property. Our transformation can be extended to enable the EAAC ("expensive to attack in the absence of collapse") property introduced in [62]. Informally, the EAAC property captures a strong form of security in PoS protocols: it guarantees that launching a successful attack is prohibitively costly. More precisely, if an adversary attempts to violate the protocol's guarantees, the property ensures that the faulty participants responsible for the attack will be penalized by having their stake slashed. At the same time, the property protects correct participants by ensuring they remain unharmed and retain their stake, even during adversarial conditions. Given the technical complexity and detailed formalism of the EAAC property, we omit the full formal definition here and instead provide the aforementioned informal description. We encourage interested readers to consult [62] for the complete and rigorous treatment of the EAAC property.

Obtaining EAAC using our transformation. It is established in [62] that no non-trivial EAAC guarantees can be provided by any quasi-permissionless protocol within the standard partially synchronous model: if the adversary is strong enough to cause consistency violations, it can also evade punishment. However, the same work demonstrates that, under a stronger notion of partial synchrony – one that imposes a pessimistic upper bound on message delays Δ' (potentially orders of magnitude greater than Δ) even before GST – it becomes possible to design a quasi-permissionless protocol that satisfies the EAAC property, assuming the adversary controls less than two-thirds of the total stake in every execution. We follow this approach: our transformation, in addition to the properties already discussed, enables the resulting quasi-permissioned protocol to satisfy the EAAC property assuming that this pessimistic bound Δ' on message delays (even before GST) holds. Importantly, the resulting protocol satisfies EAAC, regardless of whether the original permissioned protocol does.

Let us now provide modifications to our transformation $\mathcal{T}(\mathcal{P})$ sufficient for satisfying the EAAC property (assuming the pessimistic bound Δ' on message delays):

- Each epoch e is executed for a duration (at least) proportional to the pessimistic bound on message delays Δ'. Concretely, each epoch is executed for more than $2\Delta'$ time.
- When a process p validating epoch e receives a fully-certified log $\mathcal{L}$, the process (1) signs a message $m = \langle \text{CONFIRM}, \mathcal{L} \rangle$, (2) sends m to all identifiers validating epoch e, and (3) gossips the received signatures (that fully-certified $\mathcal{L}$). We underline that modified $\mathcal{T}(\mathcal{P})$ includes two rounds of voting on a log: the first makes logs fully-certified (and is present in original, EAAC-less $\mathcal{T}(\mathcal{P})$), and the second revolves around the aforementioned CONFIRM messages (unique to $\mathcal{T}(\mathcal{P})$ modified for EAAC).
- When a process p receives two-thirds-stake worth of CONFIRM messages for some log $\mathcal{L}$, process p "packs together" the received CONFIRM signatures and gossips $\mathcal{L}$ along with the signatures.
- A process p outputs a log $\mathcal{L}$ (i.e., sets $\mathsf{log}(p)$ to $\mathcal{L}$) only upon receiving two-thirds-stake worth of signatures on $\langle \text{CONFIRM}, \mathcal{L} \rangle$.
- Finally, once a consistency violation occurs, the processes repeatedly initiate instances of the Dolev-Strong protocol [38]. This protocol is used to collectively agree on an updated genesis block, which incorporates slashing penalties for any processes identified as faulty during the violation. When this updated genesis block is established, the system can safely resume normal execution from this new agreed-upon state. This recovery procedure follows the approach detailed in [62, Algorithm 2].

Why do these modifications enable the EAAC property? If there is a consistency violation in an epoch e, there exists a correct process p (resp., q) validating epoch e that receives two-thirds-stake worth of signatures on some log $\mathcal{L}_p$ (resp., $\mathcal{L}_q$) *while being in epoch* e; these are the signatures making inconsistent logs $\mathcal{L}_p$ and $\mathcal{L}_q$ fully-certified. As (1) both p and q gossip the received signatures, (2) message delays are bounded by Δ' even before GST, and (3) epoch $e+1$ is executed for a period of time proportional to Δ', all validators of epoch $e+1$ receive the conflicting signatures while in epoch $e+1$ and ensure the slashing of the responsible identifiers (via the Dolev-Strong recovery procedure). Moreover, no correct process is ever slashed as no correct process (i.e., validator) ever signs conflicting logs, which means that no proof of guilt of a correct process could ever be produced.

6 Related Work

Due to space constraints, expansive comparison to related work, including on *group membership & view synchronous communication* [15, 22, 28, 15, 75, 6, 7, 67, 17] and on *deterministic reconfiguration in asynchrony* [43, 2, 3, 49, 55, 44, 80, 46, 24, 56], is relegated to App. D in [54].

For *PoS blockchain protocols*, Ethereum [42, 20, 21] is the largest PoS blockchain by market cap, but the idea of PoS traces back to the Bitcoin community [12, 13, 81, 83, 82, 65, 52, 68]. The first provably-secure PoS protocols include SnowWhite [33, 74] and Ouroboros Praos [35, 51]. Hybrid Consensus [73, 53] proceeds in epochs, each of which has an associated permissioned consensus instance, with epoch transition and output log construction similar to that of our transformation. Lewis–Pye and Roughgarden [62] and Budish et al. [19] transform HotStuff [84] and Tendermint [18], respectively, from the permissioned to the PoS setting. PaLa [26] is a partially synchronous protocol with built-in reconfiguration. Sui Lutris [16] features a unique reconfiguration mechanism for its combined partially-ordering/totally-ordering consensus mechanism [77, 47, 10]. There is a substantial line of work on *reconfiguration of replicated state-machines* [57, 59, 60, 58, 1]. Systems implementing reconfiguration include BFT-SMaRt [14] and Raft [72]. Duan and Zhang [40] propose Dyno, a family of total-order broadcast protocols with reconfiguration carefully woven in in a bespoke manner. To the best of our knowledge, no earlier work describes a closed-box transformation from permissioned to PoS consensus that preserves and provides the range of desirable properties studied in this paper.

References

1 Ittai Abraham and Dahlia Malkhi. BVP: Byzantine vertical Paxos. In *Distributed Cryptocurrencies and Consensus Ledgers (DCCL)*, 2016.

2 Marcos Kawazoe Aguilera, Idit Keidar, Dahlia Malkhi, and Alexander Shraer. Dynamic atomic storage without consensus. *J. ACM*, 58(2):7:1–7:32, 2011. doi:10.1145/1944345.1944348.

3 Eduardo Alchieri, Alysson Bessani, Fabíola Greve, and Joni da Silva Fraga. Efficient and modular consensus-free reconfiguration for fault-tolerant storage. In *OPODIS*, volume 95 of *LIPIcs*, pages 26:1–26:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.OPODIS.2017.26.

4 Bowen Alpern and Fred B. Schneider. Defining liveness. *Inf. Process. Lett.*, 21(4):181–185, 1985. doi:10.1016/0020-0190(85)90056-0.

5 Bowen Alpern and Fred B. Schneider. Recognizing safety and liveness. *Distributed Comput.*, 2(3):117–126, 1987. doi:10.1007/BF01782772.

6 Yair Amir, Cristina Nita-Rotaru, Jonathan Robert Stanton, and Gene Tsudik. Secure Spread: An integrated architecture for secure group communication. *IEEE Trans. Dependable Secur. Comput.*, 2(3):248–261, 2005. doi:10.1109/TDSC.2005.39.

7 Yair Amir and Jonathan Stanton. The Spread wide area group communication system. Technical Report CNDS-98-4, The Center for Networking and Distributed Systems, The Johns Hopkins University, 1998.

8 Balaji Arun, Zekun Li, Florian Suri-Payer, Sourav Das, and Alexander Spiegelman. Shoal++: High throughput DAG BFT can be fast and robust! In *NSDI*, pages 813–826. USENIX Association, 2025. URL: https://www.usenix.org/conference/nsdi25/presentation/arun.

9 Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the limits of latency with uncertified dags. arXiv:2310.14821v4 [cs.DC], 2023. arXiv:2310.14821v4.

10 Mathieu Baudet, George Danezis, and Alberto Sonnino. Fastpay: High-performance byzantine fault tolerant settlement. In *AFT*, pages 163–177. ACM, 2020. doi:10.1145/3419614.3423249.

11 Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In *TCC (1)*, volume 12550 of *Lecture Notes in Computer Science*, pages 260–290. Springer, 2020. doi:10.1007/978-3-030-64375-1_10.

12 Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. arXiv:1406.5694v9 [cs.CR], 2014. arXiv:1406.5694v9.

13 Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending Bitcoin's proof of work via proof of stake [extended abstract]. *SIGMETRICS Perform. Evaluation Rev.*, 42(3):34–37, 2014. doi:10.1145/2695533.2695545.

14 Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. State machine replication for the masses with BFT-SMART. In *DSN*, pages 355–362. IEEE Computer Society, 2014. doi:10.1109/DSN.2014.43.

15 Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1):47–76, 1987. doi:10.1145/7351.7478.

16 Sam Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, Ashok Menon, Todd Nowacki, Alberto Sonnino, Brandon Williams, and Lu Zhang. Sui lutris: A blockchain combining broadcast and consensus. In *CCS*, pages 2606–2620. ACM, 2024. doi:10.1145/3658644.3670286.

17 Gabriel Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *PODC*, pages 154–162. ACM, 1984. doi:10.1145/800222.806743.

18 Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. arXiv:1807.04938v3 [cs.DC], 2018. arXiv:1807.04938v3.

19 Eric Budish, Andrew Lewis-Pye, and Tim Roughgarden. The economic limits of permissionless consensus. In *EC*, pages 704–731. ACM, 2024. doi:10.1145/3670865.3673548.

20 Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. arXiv:1710.09437v4 [cs.CR], 2017. arXiv:1710.09437v4.

21 Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining ghost and casper. arXiv:2003.03052v3 [cs.CR], 2020. arXiv:2003.03052v3.

22 Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011. doi:10.1007/978-3-642-15260-3.

23 Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer, 2001. doi:10.1007/3-540-44647-8_31.

24 Martina Camaioni, Rachid Guerraoui, Jovan Komatovic, Matteo Monti, Pierre-Louis Roman, Manuel Vidigueira, and Gauthier Voron. Carbon: Scaling trusted payments with untrusted machines. *IEEE Trans. Dependable Secur. Comput.*, 22(2):1168–1180, 2025. doi:10.1109/TDSC.2024.3428617.

25 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002. doi:10.1145/571637.571640.

26 T-H. Hubert Chan, Rafael Pass, and Elaine Shi. PaLa: A simple partially synchronous blockchain. Cryptology ePrint Archive, Paper 2018/981, 2018. URL: https://eprint.iacr.org/2018/981.

27 Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. doi:10.1016/J.TCS.2019.02.001.

28 Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001. doi:10.1145/503112.503113.

29 Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. Byzantine consensus is $\Theta(n^2)$: the Dolev-Reischuk bound is tight even in partial synchrony! *Distributed Comput.*, 37(2):89–119, 2024. doi:10.1007/S00446-023-00458-W.

30 Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable byzantine agreement. In *ICDCS*, pages 403–413. IEEE, 2021. doi:10.1109/ICDCS51616.2021.00046.

31 Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, and Jovan Komatovic. As easy as ABC: optimal (a)ccountable (b)yzantine (c)onsensus is easy! *J. Parallel Distributed Comput.*, 181:104743, 2023. doi:10.1016/J.JPDC.2023.104743.

32 Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, Zarko Milosevic, and Adi Seredinschi. Crime and punishment in distributed byzantine decision tasks. In *ICDCS*, pages 34–44. IEEE, 2022. doi:10.1109/ICDCS54860.2022.00013.

33 Phil Daian, Rafael Pass, and Elaine Shi. Snow White: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography*, volume 11598 of *Lecture Notes in Computer Science*, pages 23–41. Springer, 2019. doi:10.1007/978-3-030-32101-7_2.

34 George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *EuroSys*, pages 34–50. ACM, 2022. doi:10.1145/3492321.3519594.

35 Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98. Springer, 2018. doi:10.1007/978-3-319-78375-8_3.

36 Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725, 2019. doi:10.1109/ACCESS.2019.2901858.

37 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, 1985. doi:10.1145/2455.214112.

38 Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983. doi:10.1137/0212045.

39 Sisi Duan, Hein Meling, Sean Peisert, and Haibin Zhang. Bchain: Byzantine replication with high throughput and embedded reconfiguration. In *OPODIS*, volume 8878 of *Lecture Notes in Computer Science*, pages 91–106. Springer, 2014. doi:10.1007/978-3-319-14472-6_7.

40 Sisi Duan and Haibin Zhang. Foundations of dynamic BFT. In *SP*, pages 1317–1334. IEEE, 2022. doi:10.1109/SP46214.2022.9833787.

41 Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

42 Ethereum Foundation. Ethereum consensus specifications. https://github.com/ethereum/consensus-specs, 2023. Accessed: 2023-12-14.

43 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

44 Eli Gafni and Dahlia Malkhi. Elastic configuration maintenance via a parsimonious speculating snapshot solution. In *DISC*, volume 9363 of *Lecture Notes in Computer Science*, pages 140–153. Springer, 2015. doi:10.1007/978-3-662-48653-5_10.

45 Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In *Public Key Cryptography (1)*, volume 13177 of *Lecture Notes in Computer Science*, pages 252–282. Springer, 2022. doi:10.1007/978-3-030-97121-2_10.

46 Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, and Andrei Tonkikh. Dynamic byzantine reliable broadcast. In *OPODIS*, volume 184 of *LIPIcs*, pages 23:1–23:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.OPODIS.2020.23.

47 Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. The consensus number of a cryptocurrency. *Distributed Comput.*, 35(1):1–15, 2022. doi:10.1007/S00446-021-00399-2.

48 Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: practical accountability for distributed systems. In *SOSP*, pages 175–188. ACM, 2007. doi:10.1145/1294261.1294279.

49 Leander Jehl, Roman Vitenberg, and Hein Meling. Smartmerge: A new approach to reconfiguration for atomic storage. In *DISC*, volume 9363 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2015. doi:10.1007/978-3-662-48653-5_11.

50 Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In *PODC*, pages 165–175. ACM, 2021. doi:10.1145/3465084.3467905.

51 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388. Springer, 2017. doi:10.1007/978-3-319-63688-7_12.

52 Sunny King and Scott Nadal. PPCoin: Peer-to-peer crypto-currency with proof-of-stake. https://peercoin.net/assets/paper/peercoin-paper.pdf, 2012.

53 Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *USENIX Security Symposium*, pages 279–296. USENIX Association, 2016. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kogias.

54 Jovan Komatovic, Andrew Lewis-Pye, Joachim Neu, Tim Roughgarden, and Ertem Nusret Tas. From permissioned to proof-of-stake consensus. Cryptology ePrint Archive, Paper 2025/1139, 2025. URL: https://eprint.iacr.org/2025/1139.

55 Petr Kuznetsov, Thibault Rieutord, and Sara Tucci Piergiovanni. Reconfigurable lattice agreement and applications. In *OPODIS*, volume 153 of *LIPIcs*, pages 31:1–31:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.OPODIS.2019.31.

56 Petr Kuznetsov and Andrei Tonkikh. Asynchronous reconfiguration with byzantine failures. *Distributed Comput.*, 35(6):477–502, 2022. doi:10.1007/S00446-022-00421-1.

57 Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998. doi:10.1145/279227.279229.

58 Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Stoppable Paxos. Unpublished manuscript, https://lamport.azurewebsites.net/pubs/stoppable.pdf, 2009.

59 Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical Paxos and primary-backup replication. In *PODC*, pages 312–313. ACM, 2009. doi:10.1145/1582716.1582783.

60 Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Reconfiguring a state machine. *SIGACT News*, 41(1):63–73, 2010. doi:10.1145/1753171.1753191.

61 Andrew Lewis-Pye. Quadratic worst-case message complexity for state machine replication in the partial synchrony model. arXiv:2201.01107v1 [cs.DC], 2022. arXiv:2201.01107v1.

62 Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus. arXiv:2304.14701v5 [cs.DC], 2023. arXiv:2304.14701v5.

63 Andrew Lewis-Pye and Tim Roughgarden. Beyond optimal fault tolerance. arXiv:2501.06044v7 [cs.DC], 2025. arXiv:2501.06044v7.

64 Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. CHURP: dynamic-committee proactive secret sharing. In *CCS*, pages 2369–2386. ACM, 2019. doi:10.1145/3319535.3363203.

65 Gregory Maxwell and Andrew Poelstra. Distributed consensus from proof of stake is impossible. https://download.wpsoftware.net/bitcoin/pos.pdf, 2014.

66 Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *CCS*, pages 31–42. ACM, 2016. doi:10.1145/2976749.2978399.

67 Louise E. Moser, Yair Amir, P. M. Melliar-Smith, and Deborah A. Agarwal. Extended virtual synchrony. In *ICDCS*, pages 56–65. IEEE Computer Society, 1994. doi:10.1109/ICDCS.1994.302392.

68 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

69 Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *SP*, pages 446–465. IEEE, 2021. doi:10.1109/SP40001.2021.00045.

70 Joachim Neu, Ertem Nusret Tas, and David Tse. The availability-accountability dilemma and its resolution via accountability gadgets. In *Financial Cryptography*, volume 13411 of *Lecture Notes in Computer Science*, pages 541–559. Springer, 2022. doi:10.1007/978-3-031-18283-9_27.

71 Joachim Neu, Ertem Nusret Tas, and David Tse. Short paper: Accountable safety implies finality. In *FC (1)*, volume 14744 of *Lecture Notes in Computer Science*, pages 41–50. Springer, 2024. doi:10.1007/978-3-031-78676-1_3.

72 Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In *USENIX ATC*, pages 305–319. USENIX Association, 2014. URL: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro.

73 Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, volume 91 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.DISC.2017.39.

74 Rafael Pass and Elaine Shi. The sleepy model of consensus. In *ASIACRYPT (2)*, volume 10625 of *Lecture Notes in Computer Science*, pages 380–409. Springer, 2017. doi:10.1007/978-3-319-70697-9_14.

75 André Schiper and Alain Sandoz. Uniform reliable multicast in a virtually synchronous environment. In *ICDCS*, pages 561–568. IEEE Computer Society, 1993. doi:10.1109/ICDCS.1993.287667.

76 Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. BFT protocol forensics. In *CCS*, pages 1722–1743. ACM, 2021. doi:10.1145/3460120.3484566.

77 Jakub Sliwinski and Roger Wattenhofer. Abc: Proof-of-stake without consensus. arXiv:1909.10926v3 [cs.CR], 2019. arXiv:1909.10926v3.

78 Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. Shoal: Improving DAG-BFT latency and robustness. In *FC (1)*, volume 14744 of *Lecture Notes in Computer Science*, pages 92–109. Springer, 2024. doi:10.1007/978-3-031-78676-1_6.

79 Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In *CCS*, pages 2705–2718. ACM, 2022. doi:10.1145/3548606.3559361.

80 Alexander Spiegelman, Idit Keidar, and Dahlia Malkhi. Dynamic reconfiguration: Abstraction and optimal asynchronous solution. In *DISC*, volume 91 of *LIPIcs*, pages 40:1–40:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.DISC.2017.40.

81 User cunicula and M. Rosenfeld. Proof of stake brainstorming. https://bitcointalk.org/index.php?topic=37194.0, 2011.

82 User QuantumMechanic. Proof of stake instead of proof of work. https://bitcointalk.org/index.php?topic=27787.0, 2011.

83 User tacotime. Netcoin proof-of-work and proof-of-stake hybrid design. https://web.archive.org/web/20131213085759/http://www.netcoin.io/wiki/Netcoin_Proof-of-Work_and_Proof-of-Stake_Hybrid_Design, 2013.

84 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356. ACM, 2019. doi:10.1145/3293611.3331591.

Appendix

See full version for appendix: https://eprint.iacr.org/2025/1139 [54]

Two-Tier Black-Box Blockchains and Application to Instant Layer-1 Payments

Michele Ciampi
University of Edinburgh, UK

Yun Lu
University of Victoria, Canada

Rafail Ostrovsky
University of California, Los Angeles (UCLA), CA, USA

Vassilis Zikas
Georgia Institute of Technology, Atlanta, GA, USA

Abstract

Common blockchain protocols are *monolithic*, i.e., their security relies on a single assumption, e.g., honest majority of hashing power (Bitcoin) or stake (Cardano, Algorand, Ethereum). In contrast, so-called optimistic approaches (Thunderella, Meshcash) rely on a combination of assumptions to achieve faster transaction liveness.

We revisit, redesign, and augment the optimistic paradigm to a tiered approach. Our design assumes a primary (Tier 1) and a secondary (Tier 2, also referred to as *fallback*) blockchain, and achieves full security also in a tiered fashion: If the assumption underpinning the primary chain holds, then we guarantee safety, liveness and censorship resistance, irrespectively of the status of the fallback chain. And even if the primary assumption fails, all security properties are still satisfied (albeit with a temporary slow down) provided the fallback assumption holds. To our knowledge, no existing optimistic or tiered approach preserves both safety and liveness when any one of its underlying blockchain (assumptions) fails. The above is achieved by a new detection-and-recovery mechanism that links the two blockchains, so that any violation of safety, liveness, or censorship resistance on the (faster) primary blockchain is temporary – it is swiftly detected and recovered on the secondary chain – and thus cannot result in a persistent fork or halt of the blockchain ledger.

We instantiate the above paradigm using a primary chain based on *proof of reputation* (PoR) and a fallback chain based on *proof of stake* (PoS). Our construction uses the PoR and PoS blockchains in a mostly black-box manner – where rather than assuming a concrete construction we distil abstract properties on the two blockchains that are sufficient for applying our tiered methodology. In fact, choosing reputation as the resource of the primary chain opens the door to an incentive mechanism – which we devise and analyze – that tokenizes reputation in order to deter cheating and boost participation (on both the primary/PoR and the fallback/PoS blockchain). As we demonstrate, such tokenization in combination with interpreting reputation as a built-in system-wide credit score, allows for embedding in our two-tiered methodology a novel mechanism which provides collateral-free, multi-use payment-channel-like functionality where payments can be instantly confirmed.

2012 ACM Subject Classification Security and privacy → Cryptography; Security and privacy

Keywords and phrases Fault tolerant blockchain, instantly confirmed payments

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.19

Related Version *Full Version*: `https://eprint.iacr.org/2025/1405`

Funding Authors supported in part by Sunday Group, Inc. as part of its Möbby project.
Yun Lu: Supported by NSERC (RGPIN-03642-2023).
Rafail Ostrovsky: Supported in part by NSF grants CNS-2246355, CCF-2220450, US-Israel BSF grant 2022370
Vassilis Zikas: Supported in part by NSF Award #2448339. Work done in part while the author was at Purdue University.

© Michele Ciampi, Yun Lu, Rafail Ostrovsky, and Vassilis Zikas;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

The security of distributed cryptographic protocols typically relies on (unverifiable) assumptions about the number of (or the resources held by) corrupted vs. uncorrupted (aka honest) parties. As an example, it is known that an honest majority is necessary and sufficient for secure decentralization in the synchronous model; this includes (blockchain-based) decentralized ledgers, where honest majority (of either parties or a resource like stake, e.g., [4] or hashing-power [38]) is required. In a blockchain-based cryptocurrency, which might hold billions of dollars, violation of such assumptions can have devastating effect for its users, e.g., [1, 5].

Two types of defenses are typically employed to fortify blockchains in face of such silent threats: The first one is a game-theoretic argument often used to argue the so-called *economic robustness* of primitives, i.e., their resistance to rational attackers that might violate the underlying cryptographic assumption, but are only willing to perform attacks that maximally increase their utility. Such arguments have been used in the cryptography research literature [19, 11, 12, 9, 10, 35]. but have also become common in mainstream cryptocurrencies, like (PoS-based) Ethereum [4]. The second is to use *fallback security* where the security assumption is fortified by an additional assumption, so that only if both fail the protocol might lose (all) its security properties. This fallback paradigm has been employed in a wide range of cryptographic constructions [25, 14, 31, 6, 17].

An advantage of fallback security is that one is able to use one of the assumptions in an optimistic manner. A classical example here is that of fairness in multiparty computation (MPC), e.g., [7, 8]: It is well known that fair MPC requires that at most $t < n/2$ of the parties might be corrupted; however, without fairness, an arbitrary number, $t < n$ of corrupted parties can be tolerated (assuming a public-key infrastructure, in short PKI). Thus one can employ an external fall-back mechanism, e.g., a judge [7] or a blockchain compensation mechanism [25, 14], where we can optimistically run the (unfair) MPC protocol for $t < n$ and if a fairness violation is detected, then use the fallback to resolve the fairness violation. Such an optimistic model of execution has been also used in the blockchain realm to allow for better scalability and or network tolerance properties, with Thunderella [34] and Meshcash [13] being the most representative examples: In a nutshell, these protocols rely on smart refinements of the proof-of-work and/or proof-of-space-time paradigms, using novel optimistic methods to accelerate the blockchain and improve scalability and finality under more restrictive security assumption.

The starting point of our work is a novel, tiered fallback-mode for blockchains. The idea is to use two assumptions, each of which is sufficient for a blockchain, in a tiered manner: If *Assumption* 1 (which is often called the *primary* assumption) is true (we refer to this case as the system's *normal mode*) then the system maintains all its security and performance guarantees – including safety and liveness – irrespective of whether or not *Assumption* 2 is true. Additionally, if *Assumption* 1 fails, then the core properties of the system – safety and liveness – are not violated, but the system might be temporarily slowed down (we say that liveness gets slower). We refer to this latter case as *recovery mode*. We note that being able to guarantee both safety and liveness when either of the two assumptions fails, set our construction apart from the above optimistic approaches to blockchain [34, 13]. In addition to achieving the above tiered security, as we show below, by instantiating our proposed construction with appropriate assumptions can combines the optimistic mode with rational arguments in order to achieve better liveness properties while embedding DeFi operations on layer one.[1]

[1] As we discuss below, such DeFi-style properties are also put on hold during *recovery mode*.

We instantiate the above tiered paradigm as follows: For our primary chain we use reputation as a resource, relying on the Proof-of-Reputation (PoR) paradigm. We follow [26] for defining the relevant assumption (Assumption 1 in the above exposition); intuitively, reputation is utilized similar to how stake is used in a proof-of-stake (PoS) system, i.e., it is extracted from the contents of (a sufficiently deep block) in the blockchain[2]. In a nutshell, the reputation of a party corresponds to its probability of behaving honestly in the system. Our protocol uses these probabilities to select parties that are tasked to maintain the blockchain – in our iterated Byzantine fault-tolerant instantiation. These parties are block proposers and endorses for each slot (see below for more details). Then the primary assumption, hereafter referred to as the *PoR-assumption*, corresponds to requiring that the extracted reputations are close to the actual "ground-truth" probabilities of honest behavior (cf. Section 4.1.)

For our fallback chain, we use the *PoS assumption* that underpins the security of the PoS chain, e.g., that the majority of the stake in the system being held by honest parties[3]. We remark that, as discussed below, our results use mostly black-box access on the secondary (fall-back) chain, which can therefore be instantiated with different distributed ledgers – its fallback security will then rely on the security of that instantiation. However, for clarity, we chose to anchor our description to a PoS fallback blockchain.

The above combination of PoR with PoS fallback gives full security, i.e., both safety and liveness if the PoR assumption is true (independently of the stake distribution), and, if the PoR assumption fails but the PoS assumption is true, still both safety and liveness are guaranteed, albeit with a temporarily slower liveness parameter. More exciting, as discussed below, we introduce a mechanism which allows receivers to *instantly* confirm payments when the sender pledges a one-time collateral to them in case of a fork/double spending. With the use of a reputation system in PoR, we can even construct a mechanism that does this *without* parties locking their own coins. This latter improvement can be achieved by interpreting reputation as the blockchain-analogue of credit-score, a paradigm which we believe can we use to embed also additional (classical) finance operation onto the above layer one thus opening the path to better DeFi applications, that leverage the above tiered fallback approach. In the following we give additional details on our contributions and techniques.

2 Our Contribution and Techniques

At a high level, our PoR-with-PoS-fallback design follows the PoR/PoS-hybrid blockchain blueprint of [26] (which shares several ideas with [34] and [13].) However, our approach includes multiple improvements over [26] and extensions – among others, a recovery mechanism when PoR fails, censorship resistance, concrete instantiation of the lottery for selecting minters, and explicit mechanisms (and economic robustness arguments) for participation and embedded DeFi operations. Furthermore, rather than anchoring our protocol to that of [26], we follow a more abstract, nearly black-box approach: We extract properties of the primary (PoR) and fallback (PoS) blockchains that are sufficient for our arguments; then any instantiation of the above chains which satisfies these properties would yield an instantiation of our paradigm.

[2] The choice of the concrete algorithm that extracts the reputation is orthogonal to our results and is therefore treated here as a black box.

[3] As is common in the blockchain literature, we use the term "honest" to refer to parties that faithfully follow the protocol; unlike misbehavior, e.g., signing conflicting messages, honesty is not a property we can verify and is used only in the analysis of the protocol.

In more detail, both the PoR and the PoS chains rely on a lottery for selecting the nodes that are in-charge of maintaining them. For the sake of exposition, let us focus on the following structure for both the primary (PoR) and fallback (PoS) ledger protocols: (1) The protocols advance in fully synchronized slots, where each slot might take several (synchronous) rounds and is associated with the creation of a specific block (the round and slot number is reflected in the block's contents). (2) The lottery (which is seeded by randomness extracted from the PoR chain) selects *block proposers* and *block endorsers*: block proposers report their valid transactions they have heard and which are not already included in the previous slot to the block endorsers via a byzantine fault-tolerant broadcast protocol; the latter collect all valid transactions they receive (as broadcasted) from the proposers, add their signatures to it and propagate it (via standard gossip style diffusion) to the whole system. We stress that the above structure is for keeping the exposition as simple as possible, (and in particular will allow us to argue security assuming both sub-chains has finality); however, the proposed combination paradigm can be instantiated with different types of blockchains, including Nakamoto-style blockchains (although the concrete liveness and safety parameter will need to be tuned to account for eventual finality).

We stress that our fallback design assumes that a node runs both the PoR and the PoS chain (in parallel; however, in normal mode, the fallback chain has minimal transactions); in other words, the (Layer 1 of) the blockchain consists of a combination of the two ledgers. This is in contrast to approaches that use only PoS as Layer 1 and overlay a reputation-based protocol as a Layer 2 protocol to improve upon the properties of Layer 1 [15, 30]. We note in passing that such a Layer 2 solutions cannot achieve the fallback guarantees that we are aiming for, as failure of the assumption supporting Layer 1 would induce both safety and liveness failure, irrespective of how well the reputation reflects the ground truth.

We next sketch the main properties of our tiered blockchain design and the techniques for achieving them.

Detection and recovery from safety violation on the PoR chain

As already discussed in [26], reputation is a subjective and not necessarily future-proof resource. In particular, some malicious (covertly operating) nodes might behave in a way that makes the reputation-extraction algorithm assign to them high-reputation, and then, if they manage to form a majority of slot endorsers for some slot, attack the system, e.g., make it fork or stall. As discussed below, we will employ incentives, by means of reputation penalty, to deter such behavior, but this is not enough; we need a mechanism to ensure that such safety and liveness violations do not occur in the first place. For this we use the fallback PoS chain: the nodes that are selected in the PoS lottery to propose the next PoS block report their view of the most headers (these are hash pointers) of the PoR chain, starting from the hash-pointer reported in the previous PoS block. If any user observes that the reported hash-pointers are different than what he observes in his view of the PoR chain, then he complains in the next PoS block, also presenting evidence that his complain is valid – i.e., presenting the inputs to the contested hash chain. If such a dispute occurs, and both involved hash chains have *proper justification*, i.e., they are both valid continuations of the PoR chain[4] then the system enters a recovery mode, where the PoR chain takes control and resolves any double spending that might have occurred as a result of the above fork. As proven in [26], this can only happen when the assumption of the primary chain fails, i.e., the

[4] Intuitively, "proper justification" includes evidence that the hash pointers were (part of blocks) that were signed by the majority of valid endorsers – i.e., endorsers that according to the PoR lottery were chosen for the corresponding slot.

reputation considerably deviates from the ground truth meaning that the PoR assumption has failed. We also note that in the case a fork is detected on the fallback chain, there is sufficient "evidence" to allow us to remove (i.e., zero out the reputation of) parties who have misbehaved, i.e., achieve a certain level of *accountable safety* (e.g., [22, 32, 29, 33]), by identifying at least one corrupt party in case of a safety violation.

Although the idea of a detection mechanism was already proposed in [26], there was no recovery mechanism, and the treatment considered the stake on the PoS system independent of the contents of the PoR. This, for example, could work if one uses an already deployed PoS blockchain as a fallback chain, where the fallback assumption is on the stake of that blockchain. In this work, we are using PoR and PoS to build a self-contained, Layer-1 blockchain; this means that both the stake and the reputation need to be extracted from the state of the system, which raises several technical challenges as discussed below. The first challenge is:

> *Where do we extract the reputation (resp. stake distribution) and seeds for the PoR (resp. PoS) lottery from?*

The answer to the above question is not straightforward, because, due to the tiered security, the PoS chain is needed when PoR is found to be unreliable, e.g., it has forked. Hence one needs to be careful where to extract the reputation, stake distribution, and PoR and PoS lottery-seeds from – in particular, because the reputation is needed to check that a dispute on the PoS chain is justified. This is resolved by designating certain PoR-blockchain blocks (in predefined slot numbers) as *checkpointing* blocks and using them to extract the above runtime parameters. Such blocks need to be deep enough in the PoR chain, so that they have been already certified in the PoS chain without any dispute, and sufficient time has passed to ensure that any dispute should have been raised. By appropriate bounds on the network latency and the liveness parameters of the PoR vs PoS chains, we can provide such a depth.

The second challenge is with respect to recovery: Once the joint state of the two chains reflect the fact that the primary chain has forked, i.e., there is a dispute on the secondary chain with proper justification, the challenge becomes to recover from this situation without any honest user losing any funds. Since transactions require a signature from the owner of the funds, the only events that make such loss possible is double spending. To ensure that this event does not occur we require that a party only considers a transaction, say from PoR block B_i, as settled once the PoS block that endorses B_i is deep enough in the PoS chain and remains undisputed. Once again, this interval depends on the relative liveness of the PoR vs the PoS chain. (As we shall see below, our DeFi-style mechanisms also allows also for transactions can be confirmed at the speed of just one PoR slot).

Thus the core task of the recovery mechanism is to re-bootstrap the PoR chain. In a nutshell, this is done (in recovery mode) by temporarily using the PoS chain as primary, to create a new PoR genesis block, which includes (1) a pointer to the most recent PoR block B^* that has been certified in a deep-enough PoS block and is undisputed, (2) any transactions that were included (in any PoR-chain fork) following B^* – the PoS chain receives such transactions for a fixed period, and (3) any new transactions received (on the PoS chain) during recovery mode. Any parties who have been detected to sign conflicting messages have their reputation zeroed-out, which allows to sanitize the reputation as part of the recovery process.[5] At the end of the recovery period, the PoS chain puts all the collected

[5] This elimination of cheaters already allows for sanitizing; however, one can also use the recovery period to collect, on the PoS chain, new/additional/modified reputation data from the users and apply the reputation extraction AI on these new data.

transaction in a PoR block that points (by a hash pointer) to B^*; this new block along with the state extracted from the PoR chain up to B^* serve as a genesis block for restarting (in fact, resuming from the state ending to B^*) the PoR chain as the primary chain. We stress that the above is a very high level description, and a careful treatment is necessary to ensure security of the recovery procedure. The details can be found in Section 6.

Detection and recovery from liveness violation (on the PoR chain) and transaction censorship

The next new feature of our two-tier architecture is the detection and recovery of liveness violation and transaction censorship on the PoR chain. Let us start with liveness violation: The aforementioned safety-violation detection kicks in (on the PoS chain) when two parties report a fork, each with proper justification, as above.

However, a liveness violation (i.e., a party *not* sending something at the correct time) is much harder to detect and prove. In particular, it is shown that *accountable liveness* (analogous to the accountable safety discussed above) is impossible in our setting (of dishonest majority) [28]. Nevertheless, we can achieve a compromise that is still, in a sense, the "best of both worlds": even if one of the PoR or PoS chains can fail, we either have liveness, or all honest parties will go into recovery mode, where reputations can be reset (e.g., through an external process). Intuitively, we do so by having PoS block creators vote on whether they believe a liveness violation is happening (this is abstracted through the external signal defined in Prop. 5.1). If PoR liveness is violated (and thus PoS assumptions hold), it means that (most) PoS blocks will have honest majority block creators – who will either (1) help propagate PoR blocks (if a PoR block is simply delayed), or (2) vote to confirm a liveness violation. In the case of (1), we still achieve liveness (albeit slightly slower). In case PoS block creators agree (have majority vote) on a liveness violation, then those disagreeing with the claim of liveness violation can post the contested PoR block (achieving liveness, again), or if no one disagrees, then this is evidence for liveness violation (i.e., no party claims to have the PoR block), and all parties can go into recovery mode to reset reputations. On the other hand, If PoR liveness is *not* violated, all honest parties would receive blocks without delay, and would never go into recovery regardless of the results of votes on the PoS.

Another complicated case is transaction censorship, aka transaction-liveness violation: A user u attempts to put its transaction to the PoR chain for a long time, but (due to violation of the PoR assumption) it is ignored by several consecutive slot committees. In this case, u posts its transaction (along with an annotation that this is a censored transaction) on the PoS blockchain – again, u by observing censorship, u knows that the PoR assumption failed, which means that the PoS assumption holds. Every party that observe such a transaction on the PoS chain, attempts to put it in the PoR chain in one of the next several blocks. If parties observe that an (allegedly) censored transaction fails to make it in the PoR chain by a sufficiently long timeout (cf. Section 6) they confirm the censorship (and therefore the issue on PoR) and can therefore trigger the recovery procedure as above. In this case, members of the most recent committee who did not include the transaction, have their reputation zeroed out. This does not penalize honest nodes, who would have seen the transaction on the PoS chain and would have therefore included it.

Reputation as DeFi-enabler: Instantly confirmed *safe transactions*

Last but not least, we demonstrate how having (monetized, as above) reputation baked into the protocol's Layer 1, allows for improving both the speed and the functionality of the blockchain system. In brief, the idea is the following: a party p_i can pledge (deposit/lock as

collateral) a certain amount of coins v to a specific recipient p_j through a special PLEDGE transaction. Having created such a pledge, whenever p_i sends a transaction tx to p_j with amounts up to v, this transaction tx is considered *safe* and p_j can accept it as soon as they see it on a PoR block. Note that since p_j did not wait for tx to settle on the secondary chain, it can no longer ensure that the coins in tx cannot be double spent in the time it takes to the PoS to confirm. However, if double spending occurs (which means that the PoR blockchain has forked), p_j can claim (as part of the recovery mechanism) the original amount v through the collateral.

At first sight the above might appear as a special, Layer 1, version of a payment channel [3]. However, it is not hard to verify that unlike payment channels, the above allows p_i to make instant payments to p_j several times with a single pledge, as long as the interval between any two such payments is long enough for the PoS chain to have the time to detect forks.

Notwithstanding, the novelty in the above mechanism becomes more evident once the above pledge is replaced by a mechanism that allocates system funds (called a *treasury*) as collateral, with the amount allocated to each party dependent on their reputation. Intuitively, we use the parties' reputation analogous to how financial services use the concept of credit score. Specifically, the system can pledge for each party an amount (depending on the reputations) that acts as the above pledged coins. The party can then pledge (without needing to deposit their own funds as collateral) to a recipient, or several recipients if they specify the associated fractions of the pledged amount for each. By properly tuning the pledged amount for each party using the reputations, we can limit the total risk of the system (i.e., the amount that could be lost if all parties misbehaved); furthermore, we can prove that the strategy of any (malicious) node to double-spend is dominated by the node playing honestly. We give the relevant details and proofs in Section 7. We believe that the above demonstrates how our tiered PoR-with-PoS-fallback paradigm can enable DeFi mechanisms, beyond what is currently achieved by monolithic (single assumptions) or Layer-2 solutions. Devising additional such mechanisms, as well as additional applications of monetized reputation is an interesting research direction. We refer to Tab. 1 for a schematic summary of our contribution.

3 Related Work

The most relevant work is [26], from which we extend the design of our protocol. We achieve several improvements: First, in [26], when a safety failure in PoR is detected, all execution falls back to the PoS; in contrast, we propose a recovery mechanism to restart the PoR while removing misbehaving part(ies). This is akin to the idea of *accountable safety* (e.g., [22, 32, 29, 33]), where a fraction of corrupt parties can be identified and removed [29, 22] when a safety violation happens (note, due to the nature of the PoR, it is only possible to cause a fork in the event of dishonest majority with at least one identifiable, double-signing party – thus we can support removing this party). Our two-chain setting offers us additional benefits, however. In addition to identifying and removing corrupt parties, our protocol can mitigate all safety violations – even in the event of a safety violation in the PoR chain, payments using our "safe transactions" will never be reverted after recovery.

We remark that accountable (transaction) liveness is known to be harder to achieve than accountable safety described above [28]. In fact, accountable liveness (in the single-blockchain setting) is shown to be impossible in the case of dishonest majority (which we must consider when the PoR chain assumption is broken). Nevertheless, more than detecting safety failures, our work considers also the case when specific transactions may be censored on PoR, by

taking inspiration from the two-tiered approach of Thunderella [34] (more comparisons below). In contrast to [26], our protocol uses in a (nearly) black-box manner the underlying PoR protocol. Lastly, we extend the security analysis of [26] by proposing an incentive structure, such as block rewards, and prove that corrupt parties have no incentive to misbehave.

We compare below with other relevant works. Recently, Trustboost [39] presented a protocol in the validator/client setting that reaches consensus by using in a black-box (only read/write) manner some m blockchains, of which f number of them can be faulty Notably, they proved that under assumption of partial synchrony and their black-box model, it is impossible to reach consensus if $m \leq 3f$. Intuitively, this follows from the analogous impossibility in consensus literature. In this work, we circumvent this impossibility, by achieving a weaker *payment finality* property with $m = 2, f = 1$ – this property informally says that once a "safe" transaction appears in a block held by an honest party, the transaction's recipient will eventually receive the same amount of coins described by the transaction's output (even if safety may be violated). We do so by modifying the following: our protocol, while also using the underlying PoR and PoS blockchains in a somewhat black-box way: we do not change how their block creators are chosen or how they reach consensus about a block, but we parametrize them by defining e.g., a transaction validity predicate, as well as dictating how stake and reputation are computed. Moreover, we consider synchronous network. Interestingly however, our protocol also achieves a *stronger* property than previous work: if the safety of our protocol's underlying PoR blockchain fails, we not only detect such failure, but can also restart the PoR chain and remove at least one dishonest party. This means that we could recover from any inaccuracies in our reputation system (the assumption underlying the PoR blockchain's safety).

Other previous works also construct blockchains combining (several) underlying blockchain protocols. Thunderella [34] achieves instantaneous confirmation in the optimistic case of 3/4 honest majority of an agreed-upon committee and safety when its underlying (PoW) blockchain has honest majority. Intuitively, Thunderella combines chains – one slow and one fast. The fast path requires signatures from 3/4 parties from an agreed-upon committee. In parallel, a slow (e.g., PoW) chain is run, on which parties post hashes of fast blocks or transactions which they would like to see on the fast chain. These allow parties to detect failures in the fast chain and fallback to the slow one. Our liveness property uses a similar method as Thunderella, but we guarantee safety in case either of the fast (PoR) or slow (PoS) chains may fail. This is in contrast to Thunderella, where (even) a temporary fork on any of its two chains can result in a safety violation. In addition, we also outline in Sec. 7 how to incentivize parties (i.e., reward them) to follow our protocol. Meshcash [13] devised a race-free mechanism, termed tortoise and hare consensus to take advantage of a DAG-based, fast asynchronous but possible buggy protocol to improve liveness and scalability, while relying on a slower synchronous consensus mechanism to ensure safety. The work of Tas et al. [40] considers the same problem as ours, but is somewhat incomparable. They treat safety and liveness as properties that can be separately broken, whereas we consider the case where the underlying assumption of the PoR (or PoS) can be broken (in which case, *both* safety and liveness may not hold). In particular, [40] achieves safety if at least one underlying blockchain is safe, and liveness if all blockchains are live.

Works that achieve better safety by checkpointing blocks (e.g., on another chain) [23, 36, 37, 41] are similar to how we detect safety failures in our PoR chain; however, in our work, we allow the checkpointing mechanism (i.e., our PoS chain) to also fail to satisfy safety. Moreover, even in the case where our fast chain (PoR) fails, we still achieve (instant) finality for *transaction payments*, though with worse liveness. Compared to payment channels [3],

which allow for off-chain payments by first "locking" some amount of coins on chain, this work achieves payment finality by enabling the sender to pledge coins to the recipient. In contrast to the locked coins in payment channels, this amount of pledged coins (which we call *collateral*) is not used up until a transaction sender double-spends – which does not happen when the sender is honest (hence, pledge coins can be re-used for multiple payments whose total amount is more than the pledge coins). Finally, we note that a mechanism for guaranteeing quick payments between a merchant and a customer is also proposed in [27]. In this, a third party, upon receiving a transaction (that is not yet settled on the chain) from the customer directed to a merchant, makes the quick payment to the merchant over the payment channel, effectively taking the risk that the customer transaction may not settle on-chain (i.e., the third party works as a credit card company in some sense) [27]. In our approach, there is no third party, and there is no risk of any party losing money for transactions issued using our *safe transactions* mechanism.

4 Preliminaries and Model

We use "=" to denote equality of two different elements (i.e. $a = b$ then...) and "$\leftarrow$" as the assignment operator (e.g. to assign to a the value of b we write $a \leftarrow b$). A randomized assignment is denoted with $a \xleftarrow{\$} A$, where A is a randomized algorithm and the randomness used by A is not explicit. We model the hash function as a random oracle. Following [20, 18], we prove our result in the q-bounded model, in which all our theorems are proven assuming that the adversary does not exceed a fixed (polynomial) number of queries to the random-oracle.

Round, Slot, Epoch. Synchrony Assumption

In the blockchain protocol, we consider time as divided into discrete units called ***rounds***. For simplicity we assume the parties are synchronized – that is, honest parties agree on which round it is. We consider blockchain protocols which run in a series of ***slots***, which are contiguous lengths of rounds (may be of fixed or dynamic length[6]) in which a lottery is run to determine the parties who would be responsible for creating the block for that slot. An ***epoch*** instead is a fixed length of contiguous slots. For example, if the epoch length of a protocol is R, then epoch 1 consists of slots $\{1, \cdots, R\}$, epoch 2 consists of slots $\{R+1, \cdots, 2R\}$, and so on. In the protocols we consider, parties communicate over a network of authenticated multicast channels.

4.1 Abstract Blockchain Protocol

At a high level, a blockchain system is a cryptographic protocol executed by multiple parties, where the purpose of the parties is to agree on a set of inputs, called *transactions* and include this transaction in a *block*. This block is added to previously added blocks, starting from a special genesis block, that we denote with $\mathcal{G}$.

The main properties that a blockchain protocol guarantees are that, as long an underlying assumption holds (i.e., the majority of the computational power is controlled by the honest parties), then transactions generated by honest parties eventually become part of the chain (*liveness*), and that all honest parties always eventually agree (i.e. have the same view)

[6] For example, Algorand [21] is currently implemented with slots that vary in length of time, dependent on network conditions.

on the content of the chain (*safety*). In this work, we will use a simple abstraction of a blockchain protocol. This is described in Fig. 4.1. At a high level, the blockchain protocol is parametrized by a few procedures that we discuss here.

- GetChain, which allows each party to get access to their local view of the chain.
- TxSubmit is a multi-party protocol used by users to submit a transaction tx (e.g., this is the procedure that diffuses the transaction over the network)
- Pstatus is a procedure used by each party to obtain the most recent system status. For example, in a PoS chain, the status is represented by the stake distribution of the parties, or in the PoR chain, the status represents the reputation vector. Pstatus requires each party to input the view of the chain in every epoch and the current epoch number t. How Pstatus works depends on the type of blockchain we are using.
- BlockCreate. This is an interactive multi-party protocol that allows the creation of a new block that will be added to the chain. BlockCreate is run by each party at the beginning of every new slot using as input a seed (this is the seed used for the lottery to decide who are the block leaders are, more on this later). BlockCreate takes as input the output obtained by Pstatus, the local chain of the caller, the slot index, and a set of transactions, and it returns the new block, a set denoting the block leaders, and a certificate. The certificate contains the signatures of the block hash computed by the block leaders.
- certify is used to check if two hash headers are valid headers of two consecutive blocks in the chain.
- checklead is a procedure that allows verifying if a party was elected as a leader (often referred to as block creator) to generate a specific block.
- validity takes a blockchain view chain, a transaction tx, a buffer `buffer`, and some auxiliary information aux, and determines whether a transaction is valid or not concerning the transactions in chain $\cup$ `buffer`. The exact specification of this validity predicate depends on the application for which the blockchain is used.
- isvalidchain is the predicate that checks whether a chain is a valid hash chain, i.e., the chain validation predicate in [20], and that every transaction tx in chain is valid (the validity of the chain can be tested using the validity predicate above setting `buffer` $= \emptyset$.)
- GOracle is an oracle that when called, returns the (same) genesis block (i.e., the first block of the chain) to every party invoking it. GOracle is parametrized by an initial set of transactions and by an initial seed. In this work, we follow [16] and consider a relaxed version of the genesis block oracle. We indeed allow the adversary to bias the seed, and get an advantage over the honest parties, by seeing the seed before the honest parties do. How much the seed can be biased, and how much time in advance the adversary can see the seed compared to the honest parties, is defined by the parameters m and `delay` respectively. For example, when `delay` $= 0$ and $m = 1$ correspond to the case where there is only one candidate genesis block and all the parties can see it at the same round. We refer to the full version for the formal description of the oracle.
- Pseed is an oracle that returns the seed that will be used to run the lottery (e.g., the hash of previously generated blocks).

Figure 4.1: Abstract Blockchain Protocol

Parameters and setup

- Slot length $\mathsf{SlotLen}$ denoted in terms of number of rounds.
- Each party p_i running the protocol, has associated an address addr_i (a verification key of a signature scheme).
- At the beginning of the protocol the honest parties invoke $\mathsf{GOracle}$ with the command $\mathsf{get_genesis}$ thus obtaining a genesis block $\mathcal{G}$.

Each party p_i keeps track of the following:

- t: Current slot (all parties agree on the slot number due to synchronicity)
- chain: Current chain obtained via the invocation of $\mathsf{GetChain}$.
- b: List of block creators/leaders, indexed by slot. Initialized to $b[s] = \emptyset$ for all s. We only store block creators/leaders in position s, if in the slot s a block was successfully created.
- Chains contains the view of the chain of the party in every slot.
- An initially empty set $\mathtt{buffer}$.

In each slot t, each party executing the protocol, on input a set of transaction Txs does the following:

1. Run $\mathsf{Pstatus}(\mathsf{Chains}, t)$ thus obtaining status, and invoke Pseed thus obtaining seed.
2. For every $\mathsf{tx} \in \mathsf{Txs}$, if $\mathsf{validity}(\mathsf{tx}, \mathsf{chain}, \mathtt{buffer}, \mathsf{aux}) = 1$ then add tx to $\mathtt{buffer}$.
3. Run $\mathsf{BlockCreate}$ on input $(\mathsf{chain}, \mathsf{status}, \mathsf{seed}, t, \mathtt{buffer})$ thus obtaining (B, C, cert). Let B_n be the most recent block added to chain.
4. If for every $\mathsf{addr} \in C$ $\mathsf{checklead}(\mathsf{status}, \mathsf{seed}, \mathsf{addr}, \mathsf{cert}, |\mathsf{chain}| + 1) = 1$ and $\mathsf{isvalidchain}(\mathsf{chain}||B) = 1$ and $\mathsf{certify}(\mathsf{status}, H(B), H(B_n), \mathsf{seed}, |\mathsf{chain}| + 1, \mathsf{cert}) = 1$ then continue as follows, else reject the block and stop.
5. $\mathsf{chain} \leftarrow \mathsf{chain}||B$.

On request $\mathsf{GetChain}$**:** Return chain

Having defined the syntax of a blockchain protocol, we now state the security properties that we may want from such protocol. Before doing that, we define a predicate ASP. This predicate takes as input a slot number and returns 1 or 0. We require the standard properties that one expects from a blockchain protocol to hold, conditioned on the assumption ASP being equal to 1. In a nutshell ASP tells us whether the blockchain protocol we are running can be trusted due to the underlying assumption being valid. By underlying assumption here, we mean, for example, that there is an honest majority of the nodes running the protocol, or that the majority of the stake is controlled by honest parties. The exact specification of ASP depends on the protocol. The predicate ASP is of course never used by the parties running the protocol, but it is used to state our theorems and definitions as it allows us to specify what properties can be guaranteed depending on whether, in a given time frame, ASP holds or not. This will be particularly relevant when defining the security of our hybrid protocol, where the liveness and safety parameters will depend on which of the two blockchain protocols is safe (i.e., which of the two assumptions is valid). We revisit the notion of safety, block liveness, and transaction liveness to capture this aspect. These definitions are formally described in our full version.

4.1.1 Reputation and Stake

For our approach, we will rely on a PoS and a PoR chain, and we will formally describe the properties of our system depending on which of the chains is *safe*. In particular, we will denote the assumption predicate of the PoS chain with $\mathsf{ASP}^{\mathsf{PoS}}$ and the assumption for the PoR chain with $\mathsf{ASP}^{\mathsf{PoR}}$. To define the meaning of this predicate, we recall the definition of reputation system from [26]: Informally, a reputation system Rep describes the probability distribution on the honesty/dishonesty of reputation parties, described by a reputation vector $(h_1, \cdots, h_m)$. For example, if according to Rep, we have $\Pr[(h_1, \cdots, h_m) = (0, \cdots, 0, 1)] = 0.6$, then with probability 0.6, everyone except P_m is dishonest. Following [26], we call a reputation system Rep *feasible* if there exists a (PPT) sampling algorithm $\mathtt{A}$ which, with overwhelming probability, chooses a subset of parties which has honest majority.

We make a distinction between real-world reputation $\mathsf{Rep}^{\mathsf{real}}$ (which is the actual, ground-truth probabilities that parties are honest/dishonest) and system-extracted reputation $\mathsf{Rep}^{\mathsf{sys}}$. We say that $\mathsf{Rep}^{\mathsf{sys}}$ is *accurate* if the probability distribution actually describes (within negligible error) the real probability in which parties are honest/dishonest (i.e., $\mathsf{Rep}^{\mathsf{real}}$). Conversely, we call a reputation system $\mathsf{Rep}^{\mathsf{sys}}$ that is not accurate, *inaccurate*. For brevity in our theorems, we introduce the following predicates, $\mathsf{ASP}^{\mathsf{PoR}}$ and $\mathsf{ASP}^{\mathsf{PoS}}$: We say that for a PoR chain, in a slot t $\mathsf{ASP}^{\mathsf{PoR}}(t) = 1$ if and only if the reputation vector Rep is ϵ-feasible. For a PoS stake instead, we say that in a slot t $\mathsf{ASP}^{\mathsf{PoS}}(t) = 1$ if and only if the fraction of honest stake is a constant fraction above 2/3. We refer the reader to the full version for more details on reputation systems.

5 Security properties, and main tools

In this section, we describe our main tools: a PoR and a PoS blockchain system that follow the abstraction proposed in Section 4.1. We argue that our abstractions are realized by the PoR protocol proposed in [26] and by almost any PoS protocol. Before describing the exact properties we want from the PoR and the PoS chain, we highlight how our protocol will work and what are the main security properties that we guarantee.

At a very high level, in our protocol each party will run the two protocols blockchains in parallel, using the PoR protocol as the primary chain (where the transactions will be posted), and using the PoS protocol *only* to checkpoint the hash of the block generated via the PoR protocol. The PoR chain will be the only chain actually managing the coins and the treasury (more on the treasury later). Hence, any time that a party p_i wants to send money to p_j, p_i signs and issues on the PoR chain a transaction of the following type $(\textsc{standard}, \mathsf{addr}_i, \mathsf{addr}_j, v)$, where addr_i and addr_j denote the addresses of p_i and p_j respectively, and v is the number of coins that p_i wants to send to p_j. The main property we guarantee in our system is that if the assumption underlying the PoS chain (i.e., $\mathsf{ASP}^{\mathsf{PoS}}$ holds) is valid then the following happens.

- If $\mathsf{ASP}^{\mathsf{PoR}}$ (i.e., the assumption that underlines the PoR chain) holds in a sufficiently long time interval $[t_1, \ldots, t_2]$, then the transaction will be confirmed at the speed of the PoR chain in that interval (i.e., we get the transaction liveness of the PoR chain).
- If $\mathsf{ASP}^{\mathsf{PoR}}$ does not hold, then in the worst case transactions will be confirmed at the speed of the PoS chain (plus an additional constant parameter).
- Regardless of whether $\mathsf{ASP}^{\mathsf{PoR}}$ holds or not, safety is always guaranteed (the parties will eventually agree on a non-trivial common prefix).

Table 1 With *Yes* and *No* we denote if a chain is secure or it is not secure respectively. The column indicates slot intervals. For example, the first row captures the case where the PoR chain is compromised, but then it regains its security. In this case, the performances (in terms of safety and liveness) of our protocol, after some fixed recovery time Δ, come back to the PoR performances. We we write *Yes or No*, it means that the performances of the protocol are independent of whether that blockchain is secure or not.

	$0,\ldots,t_1$	$t_1,\ldots,t_2$	$t_2+\Delta,\ldots,t_3$	**Performance/security in the interval** $t_2+\Delta,\ldots,t_3$
PoR	Yes or No	No	Yes	PoR chain
PoS	Yes	Yes	Yes	and *safe transactions*
PoR	Yes	Yes	Yes	PoR chain
PoS	Yes or No	Yes or No	Yes or No	and *safe transactions*
PoR	Yes or No	No	No	PoS chain
PoS	Yes	Yes	Yes	

Moreover, if $\mathsf{ASP}^{\mathsf{PoR}}$ holds, we get the safety and liveness properties of the PoR chain, regardless of whether $\mathsf{ASP}^{\mathsf{PoS}}$ holds. Please see full version for the formal statements regarding the security of our protocol.

In a nutshell, we design a blockchain protocol that guarantees security as long as only one of the blockchain protocols is guaranteed to work as prescribed. We stress that the security of the protocol is guaranteed assuming that throughout the lifetime of the system, if an assumption fails, then the other assumption is always guaranteed to hold (i.e., we do not consider cases where the two assumptions alternate their validity).

5.1 Safe transactions

Another interesting property that we prove in our system, is that when the PoR assumption does not hold (hence forks and double spending could be performed), honest parties that accept the special type of transactions, that we call *safe transaction*, are guaranteed to get refunded. These transactions are particularly interesting, as they can be confirmed at the speed of the PoR chain, even if there is an undetected fork in the PoR chain, and still guarantee that the receiver that confirmed the transaction will eventually get refunded. The refund will be issued via a treasury. Managing this mechanism requires some care as in the event the PoR chain forks potentially unlimited double-spending attacks could be performed. We will discuss how this mechanism works in the next section (and refer to the introductory section for a glimpse of this aspect of our protocol). We provide a summary of the properties of our protocol in Tab. 1.

5.2 Main tools

We now provide a detailed description of the main tools used to design our protocol. For the PoR (PoS) abstraction, denoted with Π_{PoR} (Π_{PoS}) we name any predicates, procedures, and parameter x of the abstract protocol, with x^{PoR} (x^{PoS}). From our abstract PoR protocol Π_{PoR}, we require safety with safety parameter $\mathsf{co}^{\mathsf{PoR}}$, block liveness (with liveness parameter $\mathsf{bliv}^{\mathsf{PoR}}$), and transaction liveness (with liveness parameter $\mathsf{liv}^{\mathsf{PoR}}$). Similarly, for our abstract PoS protocol Π_{PoS}, we denote the safety parameter with $\mathsf{co}^{\mathsf{PoS}} = 1$, the block liveness parameter with $\mathsf{bliv}^{\mathsf{PoS}}$), and the transaction liveness parameter by $\mathsf{liv}^{\mathsf{PoS}}$. We now just need to define the type of transactions accepted by each blockchain protocol, how the validation predicates work, how the genesis block oracles are initialized, and what the relationship is between the PoS

and the PoR parameters. Let the slot size of the PoR and of the PoS protocol be respectively $\mathsf{SlotLen}^{\mathsf{PoR}}$ and $\mathsf{SlotLen}^{\mathsf{PoS}}$. We consider the setting where $\mathsf{SlotLen}^{\mathsf{PoR}} << \mathsf{SlotLen}^{\mathsf{PoS}}$ as this captures the setting we are interested in, where the primary chain (the PoR chain) is faster than the secondary chain (PoS). To simplify the description of our protocol we assume that $\mathsf{SlotLen}^{\mathsf{PoS}} = \ell\mathsf{SlotLen}^{\mathsf{PoR}}$ for some parameter $\ell = \mathsf{poly}(\lambda)$ (but this simplification can be easily avoided).

We require both chains to enjoy the following additional property that we call Δ-*External Signaling*. At a high level, this property requires that if an event external to the blockchain occurs, and all honest parties are aware of this event, then the event will be reported on the blockchain. Most existing blockchains satisfy this property, as the nodes running the chain can vote to claim whether a certain event occurred or not, and if the majority of the nodes support the event (majority in terms of stake or reputation), then the event is deemed valid.

▶ **Property 5.1** (Δ-External Signaling)**.** Let $\mathcal{O}$ be a deterministic oracle that, when queried by a party p, with a string q (denoting a certain event), returns 1 (denoting that the event has actually happened) or 0 (denoting that the event has not happened). The property of *external signaling* for a ledger $\mathcal{L}$ requires the following:

1. If $\mathsf{ASP}(t) = 1$ for every slot $t \in [0, \ldots, m]$ and $\mathcal{O}$ returns 1 to all the honest maintainers of $\mathcal{L}$ querying the oracle with some question query in a given time slot $\tau \in [0, \ldots, m - \Delta]$, then the transaction $(\text{ORACLE}, \mathsf{query})$ is included a part of the state of $\mathcal{L}$ within Δ time slots (i.e., accepted as a valid transaction).
2. If $\mathsf{ASP}(t) = 1$ for every slot $t \in [0, \ldots, m]$ and a transaction $(\text{ORACLE}, \mathsf{query})$ appears in the view of all the honest maintainers of $\mathcal{L}$, then it must be that at least one honest party has invoked $\mathcal{O}$ with input query and obtained 1 as answer in a slot belonging to the interval $t \in [0, \ldots, m]$.

We now describe how each individual protocol is parametrized, and then provide a high-level and formal description of our protocol.

5.3 The properties of Π_{PoR}

The genesis functionality $\mathsf{GOracle}^{\mathsf{PoR}}$ is parametrized by $\mathtt{delay} = 2\mathsf{cq}^{\mathsf{PoS}}\ell$. $\mathsf{GOracle}^{\mathsf{PoR}}$ will contain an initial seed, the initial reputation vector, and a set of transactions that contains the initial balances (stake) of the parties in the system.

Other than the properties we have just mentioned, we require the PoR protocol to enjoy this additional property. If $\mathsf{certify}^{\mathsf{PoR}}$ certifies that a hash (h_j) in the chain is subsequent to another hash h_{j-1}, then this implies that in the view of the honest parties running the PoR chain, it is indeed the case that B_{j-1} comes right before B_j with $h_j = H(B_j)$ and $h_{j-1} = H(B_{j-1})$. This property is required to hold only if the assumption underlying the PoR chain holds. More formally:

▶ **Property 5.2** (Certificate unforgeability)**.** If $\mathsf{ASP}^{\mathsf{PoR}}(t) = 1$ for every slot $t \in [0, \ldots, m]$, then for every j with $0 \leq j \leq m$, no *ppt* adversary can create B^0, cert^0, B^1, cert^1 with $B^0 \neq B^1$, such that $\mathsf{certify}(\mathsf{status}, h', H(B^0), j, \mathsf{cert}^0) = 1$ and $\mathsf{certify}(\mathsf{status}, h', H(B^1), j, \mathsf{cert}^1) = 1$ where status is obtained from an honest party executing $\mathsf{Pstatus}$.

We describe the type of transactions that can be parsed by the validity predicate $\mathsf{validity}^{\mathsf{PoR}}$, and provide a high-level description of how $\mathsf{validity}^{\mathsf{PoR}}$ works[7]. Formal description of $\mathsf{validity}^{\mathsf{PoR}}$ is found in the full version.

[7] All the transaction are signed, and the validity predicates accepts only transaction that are properly signed. We assume that the check of the signatures is implicit, and avoid restating it any time that the validation of a transaction is needed. Similarly, we assume that each transaction has a unique identifier, and that transactions with the same identifier, coming from the same address, are rejected.

- $\mathsf{tx} = (\text{STANDARD}, \mathsf{v}, \mathsf{addr}_i, \mathsf{addrs}, \mathsf{id}, \sigma)$. v is a vector of values, addr_i denotes the address of the sender and addrs is a list of receiver. The transaction specifies that $\mathsf{addrs}[k]$ should receive $\mathsf{v}[k]$ coins. σ just represents the signature of the transaction with respect to the public-key addr_i.[8] To validate this transaction, $\mathsf{validity}^{\mathsf{PoR}}$ simply checks if the balance of addr_i is sufficient to pay the receivers by looking at all the accepted transactions so far, and computing the balance of addr_i accordingly.
- $\mathsf{tx} = (\text{PLEDGE}, \mathsf{addr}_i, \mathsf{addr}_j, \mathsf{id}_i, T, \sigma_i)$. This transaction does not specify any amount, but just a sender/receiver address, and an expiration time T. Such a transaction is valid, only if addr_i has not issued any other *pledge* transaction that has still a valid timer. A timer T is valid if $T \geq t + \Delta$, where t denotes the current slot, and Δ is a parameter that we will specify later. This transaction represents a promise that the sender will pay the receiver, even in the event of a fork. In particular, if a receiver has been the victim of a double spending attack due to a fork on the PoR chain, this party can later complain on the PoS chain, and ask for a refund. This refund will be taken from the system treasury, and the refundable amount depends on the reputation of the party with address addr_i (the higher the reputation is, the higher the amount of money that a party can pledge, and hence, be used as a refund). This transaction is what will help us to create what we have called a *safe transaction*. We will provide more details on these mechanisms later.
- $(\text{REWARD}, C, m, \mathsf{cert})$, this transaction contains a set C of block leaders who generated the m-th block on the secondary chain (the PoS chain). This transaction is validated only if the m-th block of the secondary chain contains a valid checkpoint and the leaders (PoS block creators) are all part of the set C. Looking ahead, this transaction is used to reward the parties that generate a block in the secondary chain. I.e., the existence of this transaction on the PoR chain, means that all the addresses in C are entitled to receive a reward. For simplicity, we assume that this reward is fixed and defined by the parameter $\mathtt{rew}$. We will provide more detail on this aspect on Section 6

5.4 The properties of Π_{PoS}

The genesis functionality $\mathsf{GOracle}^{\mathsf{PoS}}$ is parametrized by $\mathtt{delay} = 1$. $\mathsf{GOracle}^{\mathsf{PoS}}$ contains an initial seed and only one transaction $(\text{CHECK-POINT}, H(\mathcal{G}^{\mathsf{PoR}}), 0, \perp)$, where $h_0 \leftarrow H(\mathcal{G}^{\mathsf{PoR}})$ and $\mathcal{G}^{\mathsf{PoR}}$ is the genesis block of the PoR chain. We will elaborate more on each transaction considered in our protocol but in a nutshell, the genesis block of the PoS chain contains a checkpoint of the PoR chain. Initially, this checkpoint simply represents the hash of the genesis block of the PoR chain. We do not specify how $\mathsf{BlockCreate}^{\mathsf{PoR}}$ works, except the following. Let (B, C, cert) be the output of $\mathsf{BlockCreate}^{\mathsf{PoR}}$, we require that $\forall \mathsf{addr} \in C \exists \sigma \in \mathsf{cert}$ such that $\mathtt{Ver}(\mathsf{addr}, h(B), \sigma) = 1$ (i.e., we require that the block leaders provide a valid signature of the hash of the created block).

We also require the following property to hold, which guarantees that honest block leaders are selected relatively often.

▶ **Definition 1** (Chain Quality). *A blockchain protocol that follows the abstraction of Fig. 4.1 satisfies* Chain Quality *with parameter* cq *if for every* $t \in [0, \ldots, m]$ *with* $\mathsf{ASP}^{\mathsf{PoS}}(t) = 1$ *and* $\mathsf{GOracle}^{\mathsf{PoS}}.\mathsf{corrupted} = 0$ *the following occurs: at least every* cq *slot in the interval* $[0, \ldots, m]$ *the output of* $\mathsf{BlockCreate}^{\mathsf{PoS}}$*, denoted with* (B, C, cert)*, is such that* C *contains the address of at least one honest party.*

[8] For simplicity we just assume that an address corresponds exactly to a public-key of a signature scheme.

We define formally the validity predicate $\mathsf{validity}^{\mathsf{PoS}}$ in the full version, and describe below the type of transactions that can be parsed by such a validity predicate.

- $\mathsf{tx} = (\text{CHECK-POINT}, h, j, \mathsf{cert})$ This transaction is used to checkpoint block-hashes of the PoR (the primary) chain on the PoS (the secondary) chain. In particular, the transaction contains the hash h of a block generated in the PoR chain (recall that this transaction is for the PoS chain), j is the block index, and cert represents the certificate that the block is actually the j-th block of the PoR chain. This transaction will be accepted, only if $\mathsf{certify}^{\mathsf{PoR}}$ returns one (recall that $\mathsf{certify}^{\mathsf{PoR}}$ is guaranteed to return one if a hash does indeed correspond to a valid PoR block).
- $\mathsf{tx} = (\text{COMPLAINT}, h^\star, \mathsf{cert}^\star, j)$ this transaction is used when a party sees a check-point with hash value $h \neq h^\star$, with respect to the same index block (i.e., a party noticed that his local view of the secondary chain is different from the view of the parties who did the checkpointing). If this transaction is accepted, then all the parties running the PoS chain will understand that a fork happened in the PoR chain, and will go in a so-called *recovery mode*. We stress that the only way that a complaint can be accepted is if indeed there is a fork on the primary chain (hence $\mathsf{ASP}^{\mathsf{PoR}}$ does not hold anymore). Indeed, when $\mathsf{ASP}^{\mathsf{PoR}}$ holds, then no valid complaint can be generated due to the property of *certificate unforgeability* (Property 5.2). Also in this case, we will elaborate more on this in the next section.
- $\mathsf{tx} = (\text{PAYBACK}, \mathsf{addr}_j, \mathsf{tx}, \sigma)$ this transaction is issued by an honest party p_j only in the following situation
  - p_j is in *recovery mode*.
  - p_j accepted a transaction $\mathsf{tx}' = (\text{STANDARD}, \mathsf{addr}_i, \cdot)$ that was discovered to be part of a fork.
  - let t be the slot at which p_j accepted the transaction tx, then the PoR chain contained a transaction $\mathsf{tx} = (\text{PLEDGE}, \mathsf{addr}_i, \mathsf{addr}_j, \mathsf{id}_i, T, \sigma_i)$ with $T \geq t + \Delta$.

  In summary transaction can be accepted only when in recovery mode. If the transaction is accepted, it will allow p_j to obtain whatever amount was specified in tx', by either taking the coins from addr_i account or from the treasury system if this money is not available (mode detail will follow). This is another transaction that we will use to support *safe transactions*.
- $\mathsf{tx} = (\text{LIVENESS-VIOLATION})$. This transaction appears on the ledger only when the majority of the parties have noticed a liveness violation on the primary (PoR) chain. In practice, this is realized by making parties vote about whether they think the PoR chain has a liveness issue. As mentioned, we abstract this voting aspect by relying on Property 5.1
- $\mathsf{tx} = (\text{BLOCK-REQUEST}, i)$. This transaction is accepted only if the ledger has already accepted a transaction $\mathsf{tx}' = (\text{LIVENESS-VIOLATION})$, otherwise it is ignored. The transaction is issued by a party that did not see the i-th PoR block (this can happen only when the security of the primary chain has been violated), and requests this block to appear on the secondary chain (PoS). An honest party that sees such a transaction, which instead has seen the i-th PoR block B, can react by issuing the transaction $\text{BLOCK-REQUEST-RESPOSE}, i, B)$.

5.4.1 Instantiations of Π_{PoR} and Π_{PoS} below

We instantiate Π_{PoR} with the PoR protocol of [26] (without its fallback mechanism to a PoS chain). Intuitively, Thm. 5.2 holds since when the assumption underlying the PoR security holds (i.e., accuracy of the reputation system), honest PoR parties agree on the latest block

without ambiguity. This is guaranteed by the fact that a public lottery is run to select n committee members, and a block is accepted if the majority of the committee members sign the block. An honest member of the committee will never sign two conflicting blocks for the same slot. Hence, a certificate in this case can be thought has been the signature of all the committee members.

Any PoS protocol where chain quality, chain growth, and consistency hold (under assumption on the honest (super-)majority of stake and synchrony), will suffice as our fallback chain Π_{PoS}. This includes, for example, the Algorand [21] or Cardano/Ouroboros [24] PoS protocols. Note that the choice of PoS can have different advantages and disadvantages – for example, Algorand satisfies *instant finality* (all transactions that appear on a valid block are final) [2], but is designed assuming a higher fraction of honest parties.

6 Our Hybrid PoR/PoS protocol

To describe our protocol, which we denote with Π, we use the PoR and PoS abstract protocols that we have introduced in Section 5.3 and Section 5.4, which we have denoted with Π_{PoR} and Π_{PoS} respectively. The formal description of our protocol is provided in the full version; here we provide a more informal description of how the protocol works.

6.1 High-level overview

Our protocol has two main modes of operation: *normal* and *recovery* mode. At a high level, in *normal mode* each party will run the two protocols Π_{PoR} and Π_{PoS} using Π_{PoR} as the primary chain (where the transactions will be posted), and using Π_{PoS} *only* to checkpoint the hash of the block generated via Π_{PoR}. In more detail, any time that a block B is generated in the PoR chain (let us say that B is the k-th block), the maintainers issue a transaction $(\text{CHECK-POINT}, H(B), j, \mathsf{cert})$ (recall that cert is used in the secondary chain to certify that $H(B)$ does correspond to a hash of an actual block of the PoR chain.

6.1.1 Fork detection

If any party p_j executing the protocol notices that the hash of, for example, the k-th block in his PoR chain differs from the hash checkpointed on the PoS chain, then this party creates the transaction $\mathsf{tx} = (\text{COMPLAINT}, h^\star, \mathsf{cert}^\star, j)$, where $h^\star$ is the hash of the block in the local chain of p_j, and $\mathsf{cert}^\star$ is the certificate p_j obtained when the k-th block of the PoR chain was generated by running $\mathsf{BlockCreate}^{\mathsf{PoR}}$. Then p_j submits tx via $\mathsf{TxSubmit}^{\mathsf{PoS}}(\mathsf{tx})$.

Once tx is accepted by the PoS chain, then all the parties running our protocol will notice that the PoR chain had a fork, hence, they will enter in what we call *Recovery Mode*. In this mode, the parties will discard all the PoR blocks that have been checkpointed after the slot $\Delta = 2\mathsf{liv}^{\mathsf{PoS}}$. This guarantees that the chopped PoR chain is a chain agreed upon by all the honest parties, and this is for the following reasons. Let us say that at time t an honest party noticed that his local PoS chain was inconsistent with the checkpoint (i.e., it noticed a fork). This means that there could have been something wrong with the PoR chain already $\mathsf{liv}^{\mathsf{PoS}}$ slots earlier (this is because in the worst case, an honest party checkpoint is included in every $\mathsf{liv}^{\mathsf{PoS}}$ slots). Moreover, once a fork is detected, the complaint can take up to $\mathsf{liv}^{\mathsf{PoS}}$ slots. Hence, chopping the PoR chain of all the blocks generated in the latest $\Delta = 2\mathsf{liv}^{\mathsf{PoS}}$ slots guarantees that all the parties in recovery mode agree with the current view of the PoR chain. We note that we can use cert and $\mathsf{cert}^\star$ to eventually identify (malicious) block

leaders that accepted different block headers for the same slot. This information can be used to put to 0 the reputation of these parties. Formal details on fork detection is found in the full version.

6.1.1.1 Recovery mode

During Recovery Mode, the main goal of the protocol is to refund parties that were victims of double spending (due to the forks happening) while still allowing parties to post new transactions, but at the speed of the secondary chain (the PoS chain). To explain how we refund parties that are victims of double spending in the recovery mode, we need to introduce the type of transaction accepted in our system. In our protocol, we distinguish between two types of transactions: *standard* and *safe* transactions. A party p_i with address addr_i, to issue a standard transaction and pay a party p_j some amount of coins v, creates the transaction $\mathsf{tx} = (\text{STANDARD}, v, \mathsf{addr}_i, \{\mathsf{addr}_j\}, \mathsf{id}, \sigma)$, where id is a unique identifier, and σ is the signature of the transaction for the secret key addr_i. A sender can also specify a vector of receivers and a vector of coins to pay multiple parties via a single transaction. Then p_i invokes $\mathsf{TxSubmit}^{\mathsf{PoR}}(\mathsf{tx})$ (recall that $\mathsf{TxSubmit}^{\mathsf{PoR}}$ is the protocol that a party can use to submit a transaction to the PoR chain). Once the receiver (with address addr_j) sees tx appearing in his local chain, then it declares the transaction as valid (recall that PoR has instant finality). This is the most basic type of transaction that a blockchain protocol could offer. However, if a fork happens, then a malicious p_i can spend more than he has actually available by creating a fork (this attack is often referred to as *double spending*). We have argued how our protocol can detect a fork (using the PoS chain), and let parties agree on a common view of the PoR (pre-fork), and continue running the protocol on the PoS chain.

In Sec. 7, we describe two sources of funds to refund the victims of double spending. The first is simple: we can require that if a party wishes their transaction to be instantly confirmed, they must have pledged a sufficient amount of collateral towards the transaction receiver (the coins pledged will then be locked). The second method is to use system funds (called the treasury) to pay back victims of double spending, which we detail the implications of in Sec. 7. Intuitively, when p_i wants to pay p_j, they issue a standard transaction $\mathsf{tx} = (\text{STANDARD}, v, \mathsf{addr}_i, \{\mathsf{addr}_j\}, \mathsf{id}, \sigma)$. Then, p_j accepts the payment if it appears on a PoR block, and if there is a sufficient pledged collateral to pay for tx if a conflicting transaction is confirmed instead. Such a pledge is captured by transaction $\mathsf{tx}^\star = (\text{PLEDGE}, \mathsf{addr}_i, \mathsf{addr}_j, T)$, where T is an expiry timer, which must be checkpointed on the PoS chain. If a conflicting transaction indeed is confirmed, a fork (and double spending) happens, and the system goes into Recovery Mode, then p_j creates a *payback* transaction $\mathsf{tx}' = (\text{PAYBACK}, \mathsf{addr}_j, \mathsf{tx}, \sigma)$ to submit it on the PoS chain by invoking $\mathsf{TxSubmit}^{\mathsf{PoS}}(\mathsf{tx}')$ (recall that $\mathsf{TxSubmit}^{\mathsf{PoS}}$ is the protocol that a party can use to submit a transaction to the PoS chain). The PoS chain will accept this transaction now, and it will use the locked coins to pay p_j. We note that if no fork happens, p_i can keep issuing safe transactions to p_j as long as the timer in $\mathsf{tx}^\star$ has not expired. Moreover, p_j, upon seeing tx part of his local chain, it can immediately assume (at the speed of the PoR chain) that he will receive the v coins (recall that tx was a payment of v coins). p_j just needs to be careful to only accept unconfirmed transactions whose total amount does not exceed the pledged amount. For more detail on how these transactions are managed, we refer to the part in the full version named *"Normal mode for users: accepting safe transactions"*.

6.1.1.2 Finalizing the recovery

he Recovery Mode lasts a fixed number of rounds that we denote by $\mathsf{rectime}$. After that, the maintainers will try to restart the PoR chain. This is done by computing the balances of all the parties using the transactions included in the pre-fork PoR chain (that is not extended in Recovery Mode) and using the standard transactions that have been included in the PoS chain, and the transactions $(\textsc{payback}, \cdot)$ (included in the PoS chain as well). For every balance $\mathsf{balance}_i$ associated to a party p_i, each maintainer creates a special transaction $\mathsf{tx}_i = (0, \mathsf{addr}_i, \mathsf{balance}_i)$. Then it will detect the parties that tried to double spend. This can be easily done as the balances of these parties will be less than 0. This is because every receiver will submit a transaction of the type $(\textsc{payback}, \cdot, \mathsf{tx})$, where tx is an attempt of double spending from some corrupted sender p_i. When computing the balances, we consider all the transaction tx wrapped in *payback* transactions, hence, the balance of a party that tried to double spend will necessarily become less than 0, unless he paid his dept via a standard transaction during the recovery mode. For all the parties that have been detected double spending, the maintainers set their reputation to 0 thus obtaining a new reputation vector $\mathsf{status}^{\mathsf{PoR}}$. Then each maintainer creates a transaction $\mathsf{tx}^\star = (\textsc{reputation}, \mathsf{status}^{\mathsf{PoR}})$, creates a new genesis block $\mathcal{G}^{\mathsf{PoR}}$ that includes the transactions $(\mathsf{tx}^\star, \{\mathsf{tx}_i\}_i)$, and the seed computed by hashing the latest block generated in the PoS chain. Then checkpoint $h \leftarrow H(\mathcal{G}^{\mathsf{PoR}})$ on the PoS chain (by submitting a transaction $(\textsc{checkpoint}, h, 0)$ via $\mathsf{TxSubmit}^{\mathsf{PoS}}$). Each party waits $\mathsf{liv}^{\mathsf{PoS}}$ slots, to make sure that all the parties have the new genesis block checkpointed, and go in normal mode, using the new genesis block to run the PoR protocol from scratch. Please see full version for a formal description.

Block and Transaction Liveness Violation. We refer to the introductory part of the paper for an informal description of how we guarantee block and transaction liveness, and our full version for a formal description of this, as well as how rewards for the PoS chain are generated and exported to the primary chain.

7 Collaterals for Fast Payments

Recall that to achieve *safe* transactions that can be accepted by a receiver p_j instantly, the sender must *pledge* an amount of coins to the receiver, which will be used to pay the receiver in case of a double-spend. We describe two sources for the pledged coins (which can be used in conjunction with each other): (1) The sender p_i can deposit collaterals using their own coins. (2) We design a system that manages a pot of funds called the *treasury* used for pledging, where the amount allocated to each pledge is proportional to (a function of) the pledger's reputation. As is with the rest of the paper, $\Delta \leftarrow 2\mathsf{liv}^{\mathsf{PoS}}$, where $\mathsf{liv}^{\mathsf{PoS}}$ is the PoS's liveness and Δ is the number of slots to finalize a PoR block in the hybrid protocol. We refer to the full version for formal details, discussions and proofs.

7.1 Fast Payments without Depositing Collateral

We leverage reputation in PoR as a "credit-score" to allocate system funds (which we call the *treasury*) to parties as the collateral. We define the following operations on the treasury.

Initialization: We hard-code a starting amount T_0 in the treasury (e.g., $T_0 = 0$).

- On $(\mathsf{CheckTreasury}, \mathsf{chain})$: Returns the amount of coins in the treasury according to PoR chain chain at latest slot t. The amount returned is T_0 plus the total inflow to the treasury, minus the total outflow from the treasury.

The total *outflow* is the sum of all amounts in payback transactions (i.e., those labeled PAYBACK). Similarly the total *inflow* is the total amount of coins entering the treasury (e.g., through transaction fees, as a percentage of block rewards, etc. – for generality here we don't specify a particular mechanism for inflow).

- On $(\mathsf{CheckAllocation}, \mathsf{chain}, p, \Delta)$: Returns the amount allocated to party p from the treasury, according to PoR chain chain. This returns the output of $\mathsf{TreasuryAllocation}_\Delta(p, \mathsf{chain})$. It returns an amount proportional to the reputation of p, based on confirmed blocks in the PoR; for formal description please see full version. Honest parties will call this operation with $\Delta = 2 \cdot \mathsf{liv}^{\mathsf{PoS}}$.

Accepting payments using collateral from the treasury. Suppose a party p_j receives a transaction from p_i to p_j at slot t. Then p_j can accept this as a safe transaction if it appears on a PoR block and (1) a PLEDGE is checkpointed in the hybrid protocol (Δ slots have passed without a PoR fork detected), and (2) the total amount in accepted, not-yet-checkpointed transactions from p_i to p_j would not exceed the allocated collateral from the treasury. Formally, for (2) p_j verifies that, if p_j accepts this transaction, then the total amount in accepted transactions that are from p_i to p_j, between slots $t - \Delta$ and t, is at most $(\mathsf{CheckAllocation}, \mathsf{chain}, p_i, \Delta)$. Here chain is p_i's PoR chain at slot t.

Recovery phase and payback using collateral from the treasury. Suppose a party sees a PAYBACK transaction (which is processed on the PoS chain during recovery phase) claiming p_i owes v coins to p_j. This transaction is valid if we observe on the PoS chain: (1) two conflicting transactions that prove p_j is the victim of double-spending by p_i for amount v, (2) p_i pledged to p_j, and (3) $(\mathsf{CheckAllocation}, \mathsf{chain}, p, \Delta) \geq v$, where chain is the most recent PoR chain in the honest party's view before the recovery phase.

7.1.1 Defining and Proving Desiderata for Treasury Allocation

We list and prove the following natural desiderata for a system which allocates system funds (a treasury) to parties, under both static and dynamic reputation systems.

- **Allocation agreement.** For any party p and PoR chains chain and chain' held by any two honest parties at some slot t, $(\mathsf{CheckAllocation}, \mathsf{chain}, p, \Delta) = (\mathsf{CheckAllocation}, \mathsf{chain}', p, \Delta)$
- **Allocation persistence.** Let chain_1 be the PoR chain held by some honest party at slot t_1 and and chain_2 be the PoR chain held by some honest party at slot $t_2 > t_1$. Then unless all honest parties end recovery phase at some $t \in [t_1, t_2]$, $(\mathsf{CheckAllocation}, \mathsf{chain}_1, p, \Delta) \geq (\mathsf{CheckAllocation}, \mathsf{chain}_2, p, \Delta)$
- **Treasury always sufficient.** The value returned by $(\mathsf{CheckTreasury}, \mathsf{chain})$ for a PoR chain held by an honest party's view will be non-negative.
- **Fair allocation.** If current slot is t and parties p_i and p_j both have the same reputation according to chain at $t - \Delta$, then $(\mathsf{CheckAllocation}, \mathsf{chain}, p_i, \Delta) = (\mathsf{CheckAllocation}, \mathsf{chain}, p_j, \Delta)$.
- **No incentives to misbehave for a period of Δt.** This property holds if, for any pair p_h and p_a with the same stake and reputation, where p_h is an honest party and p_a a dishonest party who had their reputation zeroed out (which would be observed by all honest parties during recovery) at some time t_a, the utility $u_{t_a,\Delta t}(p_a) < u_{t_a,\Delta t}(p_h)$. The utility u is defined below:

$$u_{t_a,\Delta t}(p) = \sum_{t \in [t_a, t_a + \Delta t]} \mathtt{rew}_{\mathsf{PoR}} \cdot \Pr(I^{\mathsf{PoR}}_{t,r_t}) + \sum_{t \in [t_a, t_a + \Delta t]} \mathtt{rew}_{\mathsf{PoS}} \cdot \Pr(I^{\mathsf{PoS}}_{t,s_t}) + \mathtt{fprofit} \cdot \Pr(D)$$

Here, I^{PoR}_{t,r_t} is the event where p has reputation r_t at slot t and is chosen as a PoR block creator for slot t by an honest party in the hybrid protocol (and $\mathtt{rew}_{\mathsf{PoR}}$ is the block reward in PoR); $I^{\mathsf{PoS}}_{t,s}$ is the analogous event for the PoS where s_t is the stake of party p at slot t (and $\mathtt{rew}_{\mathsf{PoS}}$ is the block reward in PoS). $\Pr(D)$ is the probability that p has double-spent (make another party p' accept a transaction which is later found to be double-spending), and `fprofit` is the amount of coins gained by p by double-spending. **Informally**, this says that over a period of Δt, the honest party's total expected utility from PoR and PoS block rewards (honest parties don't double-spend), is greater than the expected utility of the adversarial party.

In particular, the first three properties imply that collateral from the treasury can be treated in the same way as collateral pledged/locked using a party's own funds.

▶ **Lemma 2.** *Using a treasury scheme that is always sufficient, and has allocation persistence and agreement, the hybrid protocol achieves payment finality.*

Case 1: Static Reputation System. We first prove the above desiderata in a static reputation system, for the operations we defined on our treasury. This is the case when reputation and number of parties do not change, except the reputation of misbehaving parties are zero'd out during recovery phase, after all double-spent transactions are paid back.

Intuitively, our allocation algorithm $\mathsf{TreasuryAllocation}$ is fair – it allocates the treasury proportionally to a function of the reputation. Moreover, it allocates based only on the confirmed checkpointed blocks in the PoR chain (so honest parties agree on the allocation, whose amount persists until the end of recovery phase where all double spending transactions are already resolved). For "no incentives to misbehave", we require an extra assumption that the amount in the treasury (and thus the amount allocated to each party) is not too large, to prevent incentives to double-spend.

▶ **Theorem 3.** *For the treasury specified in Sec. 7.1, under a static reputation system:* allocation agreement*,* allocation persistence*,* treasury always sufficient*, and* fair allocation *are satisfied with overwhelming probability. Moreover, if for any* Δt*,* $(\mathsf{CheckTreasury}, \mathsf{chain}) < m \cdot \Delta t \cdot \mathtt{rew}_{PoR}$ *for any* chain *held by an honest party (and* m *is the number of parties chosen by the PoR lottery), there are* no incentives to misbehave for Δt*.*

We defer the discussion on dynamic reputation to the full version.

References

1 Ethereum classic hit by third 51 percent attack in a month. https://www.coindesk.com/markets/2020/08/29/ethereum-classic-hit-by-third-51-attack-in-a-month/.

2 Instant finality - what makes algorand stand among blockchains. https://developer.algorand.org/solutions/avm-evm-instant-finality/.

3 Payment channels - bitcoin wiki. https://en.bitcoin.it/wiki/Payment_channels.

4 Proof-of-stake (pos). https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.

5 Verge's blockchain attacks are worth a sober second look. https://www.coindesk.com/markets/2018/06/05/verges-blockchain-attacks-are-worth-a-sober-second-look/.

6 Joël Alwen, Jonathan Katz, Ueli Maurer, and Vassilis Zikas. Collusion-preserving computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 124–143. Springer, Berlin, Heidelberg, August 2012. doi:10.1007/978-3-642-32009-5_9.

7 N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In Richard Graveman, Philippe A. Janson, Clifford Neuman, and Li Gong, editors, *ACM CCS 97*, pages 7–17. ACM Press, April 1997. doi:10.1145/266420.266426.

8 N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *1998 IEEE Symposium on Security and Privacy*, pages 86–99. IEEE Computer Society Press, 1998. doi:10.1109/SECPRI.1998.674826.

9 Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, and Subhra Mazumdar. Securing lightning channels against rational miners. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 393–407. ACM, 2024. doi:10.1145/3658644.3670373.

10 Zeta Avarikioti, Stefan Schmid, and Samarth Tiwari. Musketeer: Incentive-compatible rebalancing for payment channel networks. In Rainer Böhme and Lucianna Kiffer, editors, *6th Conference on Advances in Financial Technologies, AFT 2024, September 23-25, 2024, Vienna, Austria*, volume 316 of *LIPIcs*, pages 13:1–13:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.AFT.2024.13.

11 Christian Badertscher, Juan A. Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But why does it work? A rational protocol design treatment of bitcoin. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 34–65. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78375-8_2.

12 Christian Badertscher, Yun Lu, and Vassilis Zikas. A rational protocol treatment of 51% attacks. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 3–32, Virtual Event, August 2021. Springer, Cham. doi:10.1007/978-3-030-84252-9_1.

13 Iddo Bentov, Pavel Hubáček, Tal Moran, and Asaf Nadler. Tortoise and hares consensus: The meshcash framework for incentive-compatible, scalable cryptocurrencies. In Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander Schwarzmann, editors, *Cyber Security Cryptography and Machine Learning*, pages 114–127, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-78086-9_9.

14 Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 421–439. Springer, Berlin, Heidelberg, August 2014. doi:10.1007/978-3-662-44381-1_24.

15 Alex Biryukov, Daniel Feher, and Dmitry Khovratovich. Guru: Universal reputation module for distributed consensus protocols. Cryptology ePrint Archive, Report 2017/671, 2017. URL: https://eprint.iacr.org/2017/671.

16 Michele Ciampi, Nikos Karayannidis, Aggelos Kiayias, and Dionysis Zindros. Updatable blockchains. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 590–609. Springer, Cham, September 2020. doi:10.1007/978-3-030-59013-0_29.

17 Michele Ciampi, Yun Lu, and Vassilis Zikas. Collusion-preserving computation without a mediator. In *CSF 2022 Computer Security Foundations Symposium*, pages 211–226. IEEE Computer Society Press, August 2022. doi:10.1109/CSF54842.2022.9919678.

18 Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78375-8_3.

19 Juan A. Garay, Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *54th FOCS*, pages 648–657. IEEE Computer Society Press, October 2013. doi:10.1109/FOCS.2013.75.

20 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Berlin, Heidelberg, April 2015. doi:10.1007/978-3-662-46803-6_10.

21 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017. doi:10.1145/3132747.3132757.

22 Tiantian Gong, Gustavo Franco Camilo, Kartik Nayak, Andrew Lewis-Pye, and Aniket Kate. Recover from excessive faults in partially-synchronous bft smr. *Cryptology ePrint Archive*, 2025.

23 Dimitris Karakostas and Aggelos Kiayias. Securing proof-of-work ledgers via checkpointing. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–5, 2021. doi:10.1109/ICBC51069.2021.9461066.

24 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Cham, August 2017. doi:10.1007/978-3-319-63688-7_12.

25 Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Berlin, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_25.

26 Leonard Kleinrock, Rafail Ostrovsky, and Vassilis Zikas. Proof-of-reputation blockchain with nakamoto fallback. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 16–38. Springer, Cham, December 2020. doi:10.1007/978-3-030-65277-7_2.

27 Mario Larangeira and Maxim Jourenko. Maravedí: A secure and practical protocol to trade risk for instantaneous finality. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 285–313. Springer, Cham, July 2023. doi:10.1007/978-3-031-37679-5_13.

28 Andrew Lewis-Pye, Joachim Neu, Tim Roughgarden, and Luca Zanolini. Accountable liveness. *arXiv preprint*, 2025. arXiv:2504.12218.

29 Andrew Lewis-Pye and Tim Roughgarden. Beyond optimal fault tolerance. *arXiv preprint*, 2025. arXiv:2501.06044.

30 Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Afgjort: A partially synchronous finality layer for blockchains. Cryptology ePrint Archive, Report 2019/504, 2019. URL: https://eprint.iacr.org/2019/504.

31 Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 508–526. Springer, Cham, February 2019. doi:10.1007/978-3-030-32101-7_30.

32 Joachim Neu, Ertem Nusret Tas, and David Tse. The availability-accountability dilemma and its resolution via accountability gadgets. In *International Conference on Financial Cryptography and Data Security*, pages 541–559. Springer, 2022. doi:10.1007/978-3-031-18283-9_27.

33 Joachim Neu, Ertem Nusret Tas, and David Tse. Short paper: Accountable safety implies finality. In *International Conference on Financial Cryptography and Data Security*, pages 41–50. Springer, 2024. doi:10.1007/978-3-031-78676-1_3.

34 Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 3–33. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78375-8_1.

35 Sophie Rain, Georgia Avarikioti, Laura Kovács, and Matteo Maffei. Towards a game-theoretic security analysis of off-chain protocols. In *36th IEEE Computer Security Foundations Symposium, CSF 2023, Dubrovnik, Croatia, July 10-14, 2023*, pages 107–122. IEEE, 2023. doi:10.1109/CSF57540.2023.00003.

36 Ranvir Rana, Dimitris Karakostas, Sreeram Kannan, Aggelos Kiayias, and Pramod Viswanath. Optimal bootstrapping of pow blockchains. In *Proceedings of the Twenty-Third International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, pages 231–240, 2022. doi:10.1145/3492866.3549731.

37 Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. Blockchain CAP theorem allows user-dependent adaptivity and finality. In Nikita Borisov and Claudia Díaz, editors, *FC 2021, Part II*, volume 12675 of *LNCS*, pages 84–103. Springer, Berlin, Heidelberg, March 2021. doi:10.1007/978-3-662-64331-0_5.

38 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

39 Peiyao Sheng, Xuechao Wang, Sreeram Kannan, Kartik Nayak, and Pramod Viswanath. TrustBoost: Boosting trust among interoperable blockchains. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 1571–1584. ACM Press, November 2023. doi:10.1145/3576915.3623080.

40 Ertem Nusret Tas, Runchao Han, David Tse, and Mingchao Yu. Interchain timestamping for mesh security. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 1585–1599. ACM Press, November 2023. doi:10.1145/3576915.3616612.

41 Ertem Nusret Tas, David Tse, Fangyu Gai, Sreeram Kannan, Mohammad Ali Maddah-Ali, and Fisher Yu. Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities. In *2023 IEEE Symposium on Security and Privacy*, pages 126–145. IEEE Computer Society Press, May 2023. doi:10.1109/SP46215.2023.10179426.

Selfish Mining Under General Stochastic Rewards

Maryam Bahrani
Ritual, Vancouver, BC, Canada

Michael Neuder
Ethereum Foundation, New York, NY, USA

S. Matthew Weinberg[1]
Princeton University, NJ, USA

Abstract

Selfish miners selectively withhold blocks to earn disproportionately high revenue. The vast majority of the selfish mining literature focuses exclusively on block rewards. [7] is a notable exception, observing that similar strategic behavior is profitable in a zero-block-reward regime (the endgame for Bitcoin's quadrennial halving schedule) if miners are compensated with transaction fees alone. Neither model fully captures miner incentives today. The block reward remains 3.125 BTC, yet some blocks yield significantly higher revenue. For example, congestion during the launch of the Babylon protocol in August 2024 caused transaction fees to spike from 0.14 BTC to 9.52 BTC, a 68× increase in fees within two blocks.

Our results are both practical and theoretical. Of practical interest, we study selfish mining profitability under a combined reward function that more accurately models miner incentives. This analysis enables us to make quantitative claims about protocol risk (e.g., the mining power at which a selfish strategy becomes profitable is reduced by 22% when optimizing over the combined reward function versus block rewards alone) and qualitative observations (e.g., a miner considering both block rewards and transaction fees will mine more or less aggressively respectively than if they cared about either alone). These practical results follow from our novel model and methodology, which constitute our theoretical contributions. We model general, time-accruing stochastic rewards in the Nakamoto Consensus Game, which requires explicit treatment of difficult adjustment and randomness; we characterize reward function structure through a set of properties (e.g., that rewards accrue only as a function of time since the parent block). We present a new methodology to analytically calculate expected selfish miner rewards under a broad class of stochastic reward functions and validate our method numerically by comparing it with the existing literature and simulating the combined reward sources directly.

2012 ACM Subject Classification Applied computing → Electronic commerce; Security and privacy → Distributed systems security

Keywords and phrases Proof-of-Work, Selfish Mining, MEV

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.20

Related Version *Full Version*: https://arxiv.org/abs/2502.20360 [3]

Funding *S. Matthew Weinberg*: Supported by NSF CAREER Award CCF-1942497.

1 Introduction

Blockchain consensus mechanisms rely on incentives to coordinate behavior. To remain safe and live, crypto-economic systems require a majority (as in Proof-of-Work) or a super-majority (as in Proof-of-Stake) of participants to adopt the protocol-specified (sometimes referred to as "honest") actions. Selfish mining [10] first demonstrated that this honest behavior

[1] During Professor Weinberg's development of this paper, he participated as an expert witness on behalf of the State of Texas in ongoing litigation against Google (the "Google Litigation").

© Maryam Bahrani, Michael Neuder, and S. Matthew Weinberg;
licensed under Creative Commons License CC-BY 4.0

might not be incentive compatible for the rational miner who could earn a disproportionately large fraction of block rewards by selectively delaying the publication of their blocks. In the ensuing decade, a rich literature around strategic behavior in consensus protocols developed (e.g., in Ethereum Proof-of-Stake [27, 31, 24]). The vast majority of this literature focuses on strategies that optimize for the portion of the protocol-assigned rewards earned by the agent. These rewards, sometimes referred to as "protocol issuance" or "consensus rewards," have historically accounted for nearly all of the value in consensus participation; this is no longer true.

As modern blockchains gain usage and facilitate more significant economic activity, their decentralized applications generate revenue. Consensus participants can collect some of this revenue through the block producer's ability to arbitrarily re-order, insert, and delete transactions when they are elected leader; [8] introduces this concept as Miner/Maximal Extractable Value (abbr. MEV). MEV has been studied theoretically and measured empirically, leading to significant changes in blockchain design. Ethereum best exemplifies this, as over 90% of its blocks are built using a public, open-outcry block-building auction. The motivation for this auction is grounded in the notion of "fairness" of validator rewards. By creating a transparent market for buying and selling transaction orderings, each consensus participant should earn about the same amount of MEV – a principle originally encoded into consensus rewards, which are proportional to investment (measured in either work or stake).

A separate line of literature studies strategic behavior in decentralized finance (abbr. DeFi), which represents another source of rewards generated at the application layer. For example, loss-versus-rebalancing [21] (abbr. LVR) measures the amount of loss incurred by liquidity providers in decentralized exchanges as arbitrageurs balance the price of the decentralized exchange against an infinitely deep centralized exchange. These losses are precisely the profit available to those performing the arbitrage. This model completely abstracts the block creation and consensus processes, only considering the profits available to traders. In reality, the block producer has the final say over the transactions in their block, resulting in a large portion of this value flowing back to the consensus participants themselves.

The perspectives of the selfish mining, MEV, and DeFi literatures are incomplete in isolation. The co-mingling of revenue across the consensus and application layers necessitates a more precise model of rewards and their impact on strategic behavior, as demonstrated in the following real-world examples.

▶ **Example 1** (The launch of Bablyon)**.** On August 22, 2024, the Babylon [33] protocol launched on Bitcoin. The launch allowed `BTC` tokens to be "locked" through a transaction processed on the chain. With a cap of 1000 `BTC`, demand for transaction inclusion spiked as people rushed to be among the first to lock their tokens. This congestion led to a 68× increase in transaction fee revenue from 0.138 to 9.515 `BTC` between parent and child blocks `857909`, `857910`; over the four block range of `857908` to `857911`, the fee revenue increased by 500× from 0.031 to 15.551 `BTC` [19]. This immense growth in transaction fees persisted for only seven blocks, with an average per-block fee revenue of 9.64 `BTC`, after which the protocol reached its cap and fees returned to baseline levels. For those seven blocks, the block reward of 3.125 `BTC`, which normally represents nearly the entire source of miner revenue, was only 25% of the rewards claimed. Despite the limited scope of Bitcoin applications, Babylon exemplifies how non-protocol-specified rewards can dramatically distort miner incentives.

▶ **Example 2** (The "Low-Carb Crusader")**.** Proof-of-Stake differs from Proof-of-Work in that it requires stakers to explicitly lock up capital to participate in the system. While Proof-of-Work is limited only to incentivizing miners with positive rewards, Proof-of-Stake

enforces a subset of the protocol rules through the credible threat of destroying the capital owned by a misbehaving staker. Historically, this stick has served as an effective deterrent, but on April 2, 2023, an attacker referred to as the "Low-Carb Crusader" exploited a piece of infrastructure in the Ethereum protocol motivated by application layer-generated rewards. By tricking a server facilitating the block building auction referenced above, the attacker accessed private transaction data, which they exploited to 20 million `USD` of MEV [9]. In the Ethereum specification, this behavior violated the rules and thus was subject to a slashing penalty of 1 `ETH` (2600 USD at current prices) levied against the attacker's stake. Clearly, the consensus reward and penalty mechanism could not account for this magnitude of profit arising from the application layer. This example demonstrates the risk facing consensus mechanisms, where non-honest behaviors are incentivized with multi-million dollar exogenous rewards originating from the application layer.

These examples shows how the economic value generated in the application layer bleeds into the consensus layer rewards; see Appendix A.1 in the extended version of the paper [3] for a discussion on "timing games," which is another source of revenue for consensus participants (particularly in Proof-of-Stake).[2] To fully understand consensus incentives, a more general model for rewards is needed. In particular, a more accurate view of rewards would capture the aggregate incentives for following a specific strategy under many distinct revenue streams. The present work was motivated by that reality and takes the first step toward modeling general stochastic rewards in longest-chain protocols.

1.1 Related work

Combining the proportion of block rewards and the linear-in-time transaction fee models of [10] and [7] was the initial motivation for this work. We build upon their Markov Chains to analyze expected attacker rewards and study the β-cutoff strategies for selfish mining. As previously noted, neither work captures the state of the world in 2025; the fundamental question of "how vulnerable is Bitcoin to Selfish Mining now?" remains unanswered and of interest to the research community; this work seeks to address this gap in the literature. [37] demonstrates how large "whale transaction" fees in conjunction with the standard block rewards may result in attacker profitability at lower hashrates. They use reinforcement learning to approximate the optimal policy and profit for attackers. We also model these rewards as granting bonus value to blocks depending on the outcome of a Bernoulli trial. Our framework (Section 4) accommodates much more general rewards, and our instantiation (Section 5) includes a third source – linear-in-time transaction fees. Further, we analytically solve for the profitability of strategies rather than approximating them.

The selfish mining literature has grown extensively in the past dozen years; see [13] for a recent survey. [23, 30, 18] generalized the basic selfish mining strategy to broader strategy spaces. [34] studied the effect of the relative sizes of block rewards and transaction fees and the impact on miners' decisions on when to mine; [14] extended that analysis and show that if all miners are rationale, the equilibrium hash rates will be far below the maximal capacity. [5] demonstrated that longest chain Proof-of-Stake protocols would also be vulnerable to selfish mining – a result instantiated through numerous selfish strategies in various staking protocols: [27, 31, 24] in Ethereum, [12, 11] in Algorand's cryptographic self-selection, [25, 26] in Tezos. We extend our model of the Nakamoto Consensus Game from [4], which studies the detectability of selfish mining in Proof-of-Work.

[2] The appendix is excluded from the proceedings version of this paper due to the page limit – see the extended version [3] for the full appendices. Hereafter, we simply refer to the appendices directly without referencing the full version to reduce redundancy.

MEV is one of the most relevant topics existing blockchains are reckoning with; we focus how MEV impacts consensus mechanisms. [8] coined the term and introduced many of the key properties of MEV in permissionless systems. [36] systematized MEV strategies and proposed mitigations. [2, 6, 16] focused on the centralizing nature of MEV and how Ethereum's block building market is implemented through "Proposer-Builder Separation." [28, 32] studied timing games and their impact on consensus. [35, 29] empirically analyzed Ethereum block builders and how the market structure has evolved. We also draw on the DeFi literature when considering application-generated revenue for consensus participants. We focus on arbitrage profits as captured in LVR [21]. [20] extends the original model to capture trading fees.

1.2 Summary of results

We partition our results into two sets: practical and theoretical.

Despite the analysis of selfish mining under block rewards and transaction fees alone being several years old ([10, 7]), there remains a glaring hole in the literature to study selfish behavior under the combined rewards. Quoting from [13], a recent selfish mining SoK, "we find that only 3 works include transaction fees in their modeling; 2 consider both block rewards and transaction fees." As described in Section 1.1, [37] model transaction fees as "whale transactions" instead of linear-in-time; [15] approximate transaction fees using the average amount of time in each block; thus, they simply increase the size of the fixed block rewards. **Our first contribution is an analysis of selfish mining under the combined model of block and transaction fee rewards; this practical contribution helps paint a more accurate picture of the selfish mining in Bitcoin under a realistic aggregate reward function.**

Section 5.3 contains numerical results and discussion (see Figure 2) for the basic combination of fees and block rewards, along with other aggregate reward functions. Critically, as demonstrated in Figure 3, the protocol risk depends greatly on the reward model. For example, we show that the threshold at which an attack becomes profitable decreases by 22% when considering the two rewards together instead of only block rewards. Additionally, our plots allow us to make qualitative observations about miner behavior under different reward schemes. For example, a miner considering both block rewards and transaction fees will mine more or less aggressively, respectively, than if they cared about either alone. We confirm these analytical results through simulations (Figure 7 in the full version [3]) and by directly comparing them to existing literature (Appendices G and H in [3]).

To derive the aforementioned practical results, we develop a set of theoretical results that may be of independent interest. **We present (i) a model of the Nakamoto Consensus Game with general stochastic reward sources, (ii) a novel methodology to analytically solve for a selfish miner's rewards, and (iii) a natural set of reward function properties.** Section 2 describes the new structure we impose on the NCG and how general, time-accruing reward sources interact with difficulty adjustment, which we must explicitly account for. Further, unlike previous work,[3] our reward functions can be stochastic. Namely, we study a much more general class of *static* rewards (Definition 6), which we define as functions that accrue randomly and independently only as a function of time since the parent block. Calculating attacker profits under these general reward sources

[3] With the sole exception of [37], which studies a narrow set of random rewards – see Section 1.1 for discussion

requires the novel, path-counting technique presented in Section 4. The practical results (Section 5) described in the previous paragraph follow as a corollary since the combination of block and transaction fee rewards is static.

Lastly, we characterize a natural set of reward function properties motivated by existing blockchains (Section 3 and Appendix B in the full version [3]). We illustrate these properties through two extensive case studies. Section 3.1 examines transaction fees and describes how different assumptions about block size, transaction patience, and arrival rate manifest in very different reward functions captured by our properties. Appendix B.2 in the full version [3] focuses on arbitrager profits under various assumptions about price trajectories and leader-election mechanisms.

2 Preliminaries and model

We start by defining a stylized model of Proof-of-Work mining with general stochastic rewards. This necessitates some crucial differences between our model and previous selfish mining literature. For example, general rewards can be sensitive to specific inter-block times, requiring explicit modeling of difficulty adjustment. Section 2.2 discusses these differences in detail.

2.1 Nakamoto Consensus Game with general rewards

Let M denote the set of n miners, where miner $m \in M$ has hashrate α_m.

Views

At any time t, there is a public *view* V_t, consisting of the "state" of the blockchain known to all miners at time t. This view includes all blocks that have already been broadcast, their creation times, and the identity[4] of their creators in M. It also includes the content of each block, which contains enough information to compute the values of all variables and account balances in every block across forks. For each block B in a view, we have $\texttt{Timestamp}(B)$, the time[5] that the block was produced.

At any time t, there is also a private view V_t^m for each miner m that includes V_t and potentially some additional blocks m knows about that are unknown to all other miners (e.g., a private fork). We assume that miners don't selectively exclude a subset of miners when they broadcast, and all broadcasting happens instantaneously (e.g., no eclipse attacks [17]). As a result, V_t^m will only include V_t and any blocks mined by m that have not yet been broadcast (along with their contents).

General Rewards

Miners are rewarded for creating blocks on the eventual longest chain in the form of block rewards (a fixed value issued once per block), fees from included transactions, and potentially additional revenue stemming from their monopolistic control over the content of the block (MEV). The size of this reward can be different across blocks and might be stochastic. We abstractly model these rewards as a function R.

[4] Real-world blockchains are often pseudonymous, and the "identities" of miners refer to their public keys.
[5] Timestamp here refers to the actual creation time of the block, rather than a reported time stated by the miner.

Fix a time t, a view V, a block B in V, and a miner m. We use r to capture any exogenous randomness that could impact the value of blocks that a miner creates (e.g., the launch of a protocol that could create large amounts of congestion and resultingly higher transaction fees as in Example 1). We denote by $\mathcal{B}^m(t, V, B, r)$ the set of *valid* blocks that m can create. Because not all views are achievable under a specific realization of the randomness r, when we invoke a view V together with r, we implicitly restrict r such that V is realizable.

▶ **Definition 3** (Reward Function)**.** *A* reward function R^m *for miner* m *takes as input a time* t*, a view* V*, a block* B *in* V*, randomness* r*, as well as a block* $B' \in \mathcal{B}^m(t, V, B, r)$*, and outputs a real number,*

$$R^m(t, V, B, r, B') \to \mathbb{R}.$$

The output of R^m can be interpreted as the amount of reward collected by m for creating a block B' that extends B in V at time t given randomness r, assuming B' ends up on the eventual longest chain.

We allow different miners to have different reward functions to keep the model general. This per-miner reward can capture miner heterogeneity (e.g., from private order flow or better trading strategies). For the properties we define in Section 3 and the selfish mining analysis in Sections 4 and 5, however, we restrict our study to *miner-independent* (see Definition 4 below) reward functions.

Miner Strategies

Each miner m has a strategy that takes as input a time t, a view V_t^m, and the reward $R^m(t, V_t^m, B, r, B')$ for extending each block $B \in V_t^m$ by a valid block $B' \in \mathcal{B}^m(t, V_t^m, B, r)$, and outputs

- a block $B \in V_t^m$ to mine on,
- contents of the next block $B' \in \mathcal{B}^m(t, V_t^m, B, r)$, and
- a (potentially empty) subset of blocks in $V_t^m \setminus V_t$ to broadcast.

For each miner m, we denote by $\texttt{Next}(m, t, V_t^m, r)$ the first time after (or equal to) t that m broadcasts a block assuming their private view remains V_t^m, and by

$$\texttt{Next_Broadcaster}(t, r) := \arg\min_{m \in M} \{\texttt{Next}(m, t, V_t^m, r)\},$$

the identity of the next miner to broadcast after (or at t), breaking ties arbitrarily. We use these functions to determine the ordering of broadcasters as the game progresses (see details in [3], Appendix – Algorithm 1).

Note that miner strategies cannot directly observe the randomness r but might indirectly depend on it through the realizations of R^m and $\mathcal{B}^m(t, V_t^m, B, r)$, all of which take as input the same randomness r. While we focus on deterministic miner strategies in this paper, our model can easily be extended to account for randomized behavior.

Nakamoto Consensus Game (NCG)

The Nakamoto Consensus Game describes how views evolve given a fixed set of miner strategies. We model the game after difficulty has already been adjusted according to these strategies, resulting in a stable orphan rate λ,[6] and we normalize time so that the average block time is 1. We let time 0 refer to a point after which the difficulty of mining puzzles remains constant. We further assume that miners only extend blocks created after time 0.

[6] See Section 2.2 for extended discussion and [22] for a more comprehensive overview of difficulty adjustment is used in the Bitcoin protocol.

Prior to the game, we draw the following random variables independently:[7]

- *Miner selection* – A sequence of miners $\vec{m} \in M^{\mathbb{N}}$, where m_i is the creator of the i^{th} block. For each i, m_i is selected independently such that it equals $m \in M$ with probability $\alpha_m / \sum_{j=1}^{n} \alpha_j$.
- *Block times* – A sequence of block creation times $\vec{t} \in \mathbb{R}^{\mathbb{N}}$, where $t_0 := 0$, and the duration $t_j - t_{j-1}$ for $j \geq 1$ is drawn i.i.d. from an exponential distribution with rate $1/(1-\lambda)$.
- *Remaining randomness* – The randomness r.

Initially, there is some public view V_0 but no hidden blocks, so $V_0^m = V_0$ for all $m \in M$, where $V_0 := \{B_0\}$ is the view containing a single genesis block B_0 such that $\texttt{Timestamp}(B_0) = 0$. Starting with $j = 1$ (the variable used to index the miners $\vec{m}$ and block times $\vec{t}$) and $t = 0$, we check if there are new blocks to broadcast before updating the block that each miner is building on based on the contents of the pre-determined strategy. See Appendix C.1 in [3] for the procedure to carry out the NCG and for a note on ensuring uniqueness of the longest chain.

2.2 Notes on model

We briefly summarize how we model difficulty adjustment below. See Appendix C.2 in [3] for an extended comparison of our model to previous work and the role of independence in the randomness of rewards.

Difficulty adjustment

In practice, mining involves solving computational puzzles with adjustable difficulty. Since miners can enter (or exit) permissionlessly, the total hashrate of all miners can vary over time, resulting in varying block production rates. The protocol varies the difficulty of these puzzles based on timestamps of recent blocks, targeting a fixed average inter-block time. In Bitcoin, the difficulty updates once every difficulty *epoch* (2016 blocks/roughly every two weeks assuming ten-minute block times) by the *difficulty adjustment algorithm (DAA)*. The difficulty of extending any blocks is the same within an epoch, except for forks across the epoch boundary. Note also that forks are rarely longer than a few blocks, so this represents an insignificant fraction of the blocks in an epoch.

Fixing a set of miner strategies, one can compute the expected fraction of blocks per epoch that do not end up on the longest chain. We assume the difficulty adjusts based on this expected value (rather than directly modeling per-epoch updates described above) and calculate the profitability of various strategies under this new difficulty. Specifically, we calculate the expected orphan rate λ (Lemma 16), which implies the difficulty-adjusted rate of block production is $1/(1-\lambda)$. This corresponds to blocks on the longest chain growing at an average rate of 1.

3 Reward functions: properties and examples

Recall that miner strategies take as input the amount of reward available for extending each existing block at time t, as specified by the reward function R, and make decisions about where to mine, what to include, and what to broadcast accordingly. This section defines a set of natural properties that reward functions might have. In Section 3.1, we

[7] See Appendix C.2 in [3] for a discussion of why we can assume independence.

motivate these properties with an extensive case study on transaction fees, one of the primary revenue sources observed empirically to date. See Appendices A.2, B, B.1, and B.2 in [3] for additional properties and a second case study on LVR, a prominent source of revenue on chains with significant DEX volume.

While we define these properties in the context of the NCG in this paper, we believe their applicability extends far beyond Proof-of-Work and selfish mining. Our framework can be used to characterize rewards and their implications for the incentives of consensus participants across blockchain protocols.

Recall that in the NCG, given a set of miner strategies, three independent random variables $\vec{t}, \vec{m}, r$ are drawn and used to compute a set of views V_t^m for all miners m and all times t. Let $\mathcal{V}_t^m$ be the support V_t^m, meaning the set of views achievable at time t for *some* realization of $\vec{t}, \vec{m}, r$. Initially, $\mathcal{V}_0^m = \{V_0\}$ for all m, where $V_0 := \{B_0\}$ is the view containing a single genesis block B_0 such that $\texttt{Timestamp}(B_0) = 0$. Miner strategies in the NCG take the realization of a reward function as input. That is, at time t, miner m sees the reward $R^m(t, V_t^m, B, r, B')$ for extending each block $B \in V_t^m$ by a valid block $B' \in \mathcal{B}^m(t, V_t^m, B, r)$.

A miner-independent reward function yields the same value for the block regardless of who created it. This corresponds to a setting where all miners have access to the same set of rewards (e.g., the common value setting), and thus, we drop the superscript m. In practice, some reward sources may be heterogeneous between block producers (e.g., from private order flow or from differing abilities to extract MEV [1]). All reward functions considered in this paper will be miner-independent, but the properties can be readily generalized by tracking the subset of miners with access to each reward source. See Section 6 for a discussion of extending this work.

▶ **Definition 4** (Miner-Independent Rewards). *A reward function R is* miner-independent *if for all times t, all miners have the same set of valid views, the same set of valid blocks extending each block in those views, and equal rewards from any such valid block.*[8] *Formally, R is miner-independent if for all t, and all $m, m' \in M$,*

- $\mathcal{V}_t^m = \mathcal{V}_t^{m'}$,
- *for all $V \in \mathcal{V}_t^m$, all blocks B in V, and all r, we have $\mathcal{B}^m(t, V, B, r) = \mathcal{B}^{m'}(t, V, B, r)$,*
- *for all $V \in \mathcal{V}_t^m$, all r, all parent blocks B in V, and all valid blocks $B' \in \mathcal{B}^m(t, V, B, r)$, we have $R^m(t, V, B, r, B') = R^{m'}(t, V, B, r, B')$.*

We can also characterize reward functions that grow according to the same distribution without depending on the chain's history. The following property limits the dependence of R on the view. Intuitively, it says that the only relevant information in the view that affects the amount of reward in a block is the *timestamp of its parent.*

▶ **Definition 5** (View-Independent Rewards). *A reward function R is* view-independent *if for all times $t' < t$, any two views $V_1, V_2 \in \mathcal{V}_{t'}$ such that $\texttt{Timestamp}(B_1) = \texttt{Timestamp}(B_2) = t'$ for some blocks $B_1 \in V_1, B_2 \in V_2$, we have:*

- *for all r, the set of valid blocks extending B_1 at t in V_1 is the same as the set of valid blocks extending B_2 at t in V_2, $\mathcal{B}(t, V_1, B_1, r) = \mathcal{B}(t, V_2, B_2, r)$,*[9] *and*

[8] Technically, since blocks include information about their creator, it would be more accurate to say that there is a bijection between the set of valid views/blocks for any pair of miners. We overlook this formality to simplify notation.

[9] Recall that when we invoke a view and randomness together as inputs to a function, we implicitly assume that the randomness could give rise to the view.

- *for every valid block* $B' \in \mathcal{B}(t, V_1, B_1, r)$*, we have*

$$\Pr_{r,\vec{t},\vec{m}|V_1}[R(t, V_1, B_1, r, B') = x] = \Pr_{r,\vec{t},\vec{m}|V_2}[R(t, V_2, B_2, r, B') = x]$$

for all x*.*

Note that fixing a view V_1 (resp. V_2) can update the distribution of the $r, \vec{t}, \vec{m}$. We use the subscript $r, \vec{t}, \vec{m}|V_i$ to refer to the posterior distribution of these random variables conditioned on V_1, V_2. For example, block rewards are view-independent (within the same four year halving window) because each block earns the same fixed reward from the protocol. Example 7 below is a non-example that demonstrates how transaction fees that are not fully claimed by a parent block (e.g., arising from finite block sizes) are not view-independent because the reward of the resulting child block depends on the amount of unclaimed transaction fees.

View-independence already limits the dependence of R on the view to the timestamp of the parent block. We next define a subset of view-independent rewards where the dependence on view is limited to the length of *elapsed time since the parent block* (and is the same regardless of the exact parent block timestamp).

▶ **Definition 6** (Static Rewards). *A reward function* R *is* static *if for all* $\Delta > 0$*, all times* t_1, t_2 *and views* $V_1 \in \mathcal{V}_{t_1}$ *and* $V_2 \in \mathcal{V}_{t_2}$ *such that* $\texttt{Timestamp}(B_1) = t_1 - \Delta$ *and* $\texttt{Timestamp}(B_2) = t_2 - \Delta$*, we have:*

- *for all* r*, the set of valid blocks extending* B_1 *at* t_1 *in* V_1 *is the same as the set of valid blocks extending* B_2 *at* t_2 *in* V_2*,* $\mathcal{B}(t_1, V_1, B_1, r) = \mathcal{B}(t_2, V_2, B_2, r)$*, and*
- *for all valid blocks* $B' \in \mathcal{B}(t_1, V_1, B_1, r)$*, we have*

$$\Pr_{r,\vec{t},\vec{m}|V_1}[R(t_1, V_1, B_1, r, B') = x] = \Pr_{r,\vec{t},\vec{m}|V_2}[R(t_2, V_2, B_2, r, B') = x]$$

for all x*.*

Example 9 below highlights that transaction fees are static using the [7] model with constant arrival rate and infinite block sizes. See appendix Examples 37 and 38 in [3], which demonstrate the conditions under which LVR is or is not static.

3.1 Properties of transaction fees

To illustrate the value of the aforementioned properties of reward functions, we perform an extensive case study on transaction fees (see Appendix B.2 in [3] for a similar study but on LVR). We consider the relevant properties that arise from different assumptions about block sizes, user patience levels, and accrual rate of transactions. These examples aim to justify the properties we focus on in Section 3 and motivate Sections 4 and 5, which measure attacker revenue under multiple static reward sources.

Transaction fees

Users pay transaction fees to interact with blockchains. A mempool collects transactions as they arrive, and its state at all times is captured in our model through the realization of the randomness r. Consider transactions as infinitely divisible,[10] belonging to the same

[10] We could instead consider transactions as heterogeneous in size (e.g., as in Ethereum where transactions consume different amounts of gas) or exclusive to miners (e.g., from private order flow), but the additional complexity doesn't add anything to the qualitative observations and is thus elided.

mempool,[10] and specifying a fee. A valid block B' mined at time t and extending a parent block B can include any transactions in the mempool at t that are not already included in $\texttt{Chain}(B)$. The corresponding reward function for a valid candidate block is the sum of the fees paid by the transactions it includes.

We call users *patient* if their transactions remain valid until they are eventually included in a later block. We shorthand transactions originating from patient users as *patient transactions*.

As demonstrated in the following example, we cannot claim any further structure on the patient-user transaction fee reward function without restricting the set of valid blocks.

▶ **Example 7** (Patient transaction fees may be view-dependent)**.** Consider two blocks B_1, B_2 with the same timestamp t' and with the same parent mined at t. B_1 claims all transaction fees arriving in $[t, t']$, while B_2 claims none. The rewards of maximizing candidate blocks B_1', B_2' built on B_1, B_2 respectively, are different, as B_2' can claim more transaction fees than B_1'.

The key observation is that miners may not claim the complete set of available transactions, thus impacting the claimable rewards of descendant blocks in that view (for more formalism, see Lemma 30 in the appendix of [3]). Alternatively, consider the case where each block can include all transactions (e.g., infinite block size as in [7]). If we additionally restrict the set of views for each miner $\mathcal{V}_{t'}^m$, we *can* make the following stronger claim.

▶ **Example 8** (Patient transaction fees are view-independent if blocks are infinite capacity and fully-claiming)**.** Assume blocks have infinite capacity and restrict views to only include blocks that contain all available transaction fees at the time of mining. Then, the distribution of rewards for B' built at time t on parent block B_1 or B_2, which have the same timestamp t', is the same. Namely, the reward is the sum of patient transaction fees arriving in the interval $[t', t]$.

Here, view-independence arises from the mempool fully emptying after each block is created. Thus, the reward function only depends on newly arriving transaction fees after the parent block is mined. Importantly, this reward function *may not be static* (which is a stronger condition than view independence) because the transaction fee arrival rate may not be homogeneous over time. For example, some hours of the day (such as trading hours in Asia time zones) might result in higher transaction fee arrivals. Assuming a constant transaction arrival rate, we can further establish staticness.

▶ **Example 9** ([7]'s model of transaction fees is static)**.** Assume 1 unit of patient transaction fees arrive per unit of time, blocks have infinite capacity, and all blocks in the view claim all available transaction fees (as in [7]). A block B' extending B at time $\texttt{Timestamp}(B) + \Delta$ can claim any reward in $[0, \Delta]$. Therefore, this reward function is static.

While the previous example considers *deterministic* transaction fee arrivals (1 unit of fees per unit of time), the same claim holds if the arrival rate is a random function of r (but still identically distributed over time). Constant accrual, in addition to the mempool clearing, results in the reward function being independent of the timestamp of the parent block, making it static.

Until now, we have only considered patient users. In contrast, consider *impatient users*, who submit transactions that are only valid for the next block produced (e.g., by checking the height of the block they are included in before executing). We similarly shorthand these as *impatient transactions*.

▶ **Example 10** (Identically distributed, impatient transaction fees are static)**.** Assuming the impatient transactions arrive according to a fixed distribution over time since the parent block, this reward function is static because the mempool clears after each block.

Note that the mempool clearing after each block was necessary for both Examples 9 and 10 to be static. However, the clearing came about differently – infinite block sizes in the former and impatient users in the latter. The mempool clearing is a *sufficient* condition for staticness if the distribution of rewards doesn't depend on global clock time.

Varying the assumptions on block size and user patience allows us to describe reward functions under differing models of congestion; we now consider transaction fees that are high regardless of the block size. This *contentious transaction* model is motivated by the launch of Babylon (Example 1). Transaction fees may spike because there is immense demand not just for inclusion in a block but also for a specific ordering (e.g., needing to be one of the first 100 transactions of a particular type).

▶ **Example 11** (Bernoulli rewards are static)**.** Consider contentious transaction fees modeled as independent Bernoulli trials that occur once per block height, resulting in a constant random reward of size E with probability p. This is a static reward function.

In Section 5, we study a variant of selfish mining under a combined reward function that includes Bernoulli rewards, linear-in-time transaction fees as in Example 9, and block rewards. This combined reward function is static, which is crucial to the tractability of that analysis. See Section 1.1 for a discussion on the similarities between our model of Bernoulli rewards and that of [37]. See Appendix B.1in [3] for examples pertaining to the properties defined in Appendix B and Appendix B.2 for an extended case study on LVR.

These examples showcase the properties we ascribe to general reward functions in Section 3. While these case studies allow us to demonstrate View-Independence (Definition 5) and Staticness (Definition 6) in familiar settings, they do not cover all MEV types. As mentioned in Section 6, we see characterizing the complete set of properties and applying them to other forms of MEV (e.g., sandwiches and liquidations) as a key direction for future work. With these properties in place, we now focus on calculating expected attacker profits from performing β-cutoff selfish mining strategies under general static reward functions.

4 Selfish mining with static rewards

Sections 2 and 3 presented our model of general stochastic rewards and created a structure around these reward functions. The subsequent sections study a specific set of miner strategies to analyze their profitability and feasibility under general static rewards (Definition 6). We examine $\beta-$cutoff selfish mining strategies [7], in which the attacker determines whether or not to hide their blocks based on the amount of reward realized during the mining process.

4.1 Mining strategies in the NCG

In the NCG defined in Section 2, miners make three decisions at each time t:

1. which block to extend,
2. the contents of their next mined block, and
3. which blocks to broadcast.

Based on these decisions, we define the protocol-prescribed mining as honest.

▶ **Definition 12** (Honest mining). *The honest mining strategy is defined as,*

1. *mine on the longest chain,*
2. *claim all available rewards, and*
3. *publish every block immediately.*

In words, the honest miners always follow the longest chain and immediately share any block they find with the rest of the network. If the remainder of the network is honest, the rewards that an honest miner, i, controlling α_i fraction of the hash power is proportional to their mining power.

[10] and [7] demonstrate that selfish mining is profitable for miners (even under various tie-breaking schemes) when considering *only* block rewards or *only* transaction fees that are linear-in-time respectively. [7] also introduced β-cutoff selfish mining strategies, in which the attacker mines selfishly as long as the rewards they earn on their hidden block are sufficiently small. If their rewards are larger than a threshold β, they instead broadcast immediately to avoid losing the valuable block.

▶ **Definition 13** (β-cutoff selfish mining [7]). *If there is no private chain, the attacker follows the rules:*

1. *mine on the public longest chain,*
2. *claim all available rewards, and*
3. *withhold any block found where the time since parent is less than β (create a private chain).*

The third step above creates the private chain for the attacker; they transition into the following rules (same as original selfish mining):

1. *mine on the private chain,*
2. *claim all available rewards, and*
3. *withhold any block found unless an honest block is found and the difference in length between the public chain and the private chain is ≤ 1.*

This strategy differs from pure selfish mining only in Step 3 under no private chain, where the attacker decides whether or not to publish based on the rewards captured in the block. Note that the strategies we consider claim all available rewards; miners could instead choose to intentionally leave some rewards on the table to incentivize subsequent miners to build on their chain ("undercutting" [7]). See Section 6 for discussion on extending our framework to a broader class of miner strategies.

Given a static reward function, we want to determine the per-unit-time expected attacker rewards from following the β-cutoff strategy as in Definition 13. We develop a new technique based on a Markov Chain similar to Figure 13 in [7] and Figure 1 in [10].

▶ **Definition 14** (β-cutoff Markov Chain). *Consider the NCG where the $1-\alpha$ of the mining power follows the honest strategy and α follows the β-cutoff strategy. Then define* `State i` *for $i \geq 1$ where the attacker has a hidden chain i blocks longer than the public chain. Let* `State 0` *denote the attacker having no hidden blocks and* `State 0'` *denote the race state between the honest and attacker forks each of length 1. Let* `State 0"` *denote the state immediately after the attacker publishes their private chain.*

Figure 1 depicts this Markov Chain. We now derive the transition probabilities using a general, static reward function. When considering static reward sources, notice that R is only a function of the time since the parent block was mined; we hereafter denote this static reward source as $R(t)$, where t is the time since the parent block. This simplification allows

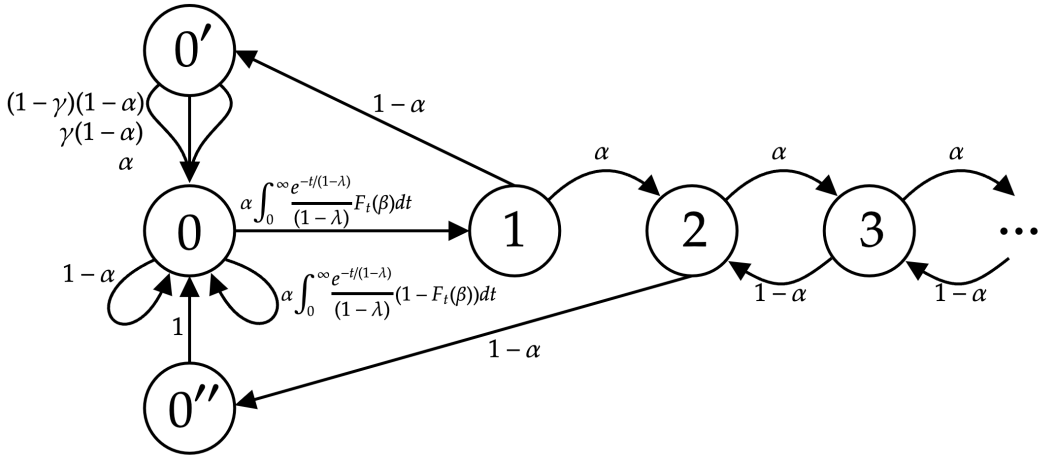


Figure 1 The Markov Chain capturing the $\beta-$cutoff strategy for miners deciding whether to publish blocks depending on the size of the static reward. $F_t(\beta)$ is the CDF of the rewards given time t since the parent block, $\Pr[R(t) \leq \beta]$. The rate of the chain is $1/(1-\lambda)$, which explicitly captures the difficulty adjustment that results from a specific β-cutoff strategy.

us to compute the probability of transitioning from `State 0` $\to$ `State 1` by comparing the expected amount of rewards earned in `State 0` conditioned on those rewards being less than β (the cutoff threshold for publishing the block in `State 0`).

▶ **Definition 15** (Static Reward CDF & PDF). *For a static reward source R and randomness r, let $F_t(x)$ denote the CDF of the reward function indexed by time t,*

$$F_t(x) = \Pr_r[R(t) \leq x].$$

Similarly, let $F'_t(x)$ denote the PDF of the reward function,

$$F'_t(x) = \Pr_r[R(t) = x].$$

To calculate the probability of withholding the block, we integrate the probability distribution of the time until the next block multiplied by the CDF of the rewards at each time.

$$\Pr[\texttt{State 0} \to \texttt{State 1}] = \alpha \int_0^\infty \underbrace{\frac{e^{-t/(1-\lambda)}}{(1-\lambda)}}_{\text{density of time}} \cdot \underbrace{F_t(\beta)}_{\substack{\text{rewards} < \beta \\ \text{by time } t}} \, dt \tag{1}$$

Intuitively, given a reward source R, this value tells us how likely it is that the rewards within an attacker block are less than β. Notice that the density function of the exponential depends on a rate parameter $1/(1-\lambda)$ (as discussed in Section 2.1), where λ the explicitly calculated orphan block rate calculated as a function of β to account for difficulty adjustment. See Lemma 16 for its derivation. Conversely, given an attacker block we can also calculate the probability that the attacker publishes the block immediately if the block rewards are be greater than β,

$$\Pr[\texttt{State 0} \to \texttt{State 0} \wedge \text{attacker block}] = \alpha \int_0^\infty \underbrace{\frac{e^{-t/(1-\lambda)}}{(1-\lambda)}}_{\text{density of time}} \cdot \underbrace{(1 - F_t(\beta))}_{\substack{\text{rewards} \geq \beta \\ \text{by time } t}} dt \tag{2}$$

With Equations (1) and (2), we construct the entire Markov chain in Figure 1. Note that it differs from Figure 1 in [10] and Figure 13 in [7], only in the transition probabilities from `State 0` calculated above for general static reward sources (Equations (1) and (2)). As in previous work, γ is the tie-breaking rate dictating the fraction of honest miners who mine on the attacker block after it is published, and there is a race of length-1 forks (in `State 1`). This parameter doesn't impact the β-cutoff itself and only affects the probability that the attacker fork wins the tie. See [3], Appendix E.1, for the calculation of the stationary distribution of this Markov Chain.

With the stationary distribution, we can explicitly solve for the proportion of orphan blocks, $\lambda \in [0, 1]$, which in turn gives us the difficulty-adjusted rate of the Poisson process of the transitions in the Markov Chain as $1/(1-\lambda)$. This rate is *faster* than the rate of canonical blocks (normalized to 1) because the orphaning process causes a reduction in difficulty.

▶ **Lemma 16** (Calculating λ)**.** *Let λ measure the probability that a block produced in the Markov Chain is orphaned. Then,*

$$\lambda = p_1(1-\alpha)\left(1 + \frac{\alpha}{1-2\alpha}\right).$$

Proof in Appendix D.1 of the full version [3]. With λ, the new block production rate is $1/(1-\lambda)$. This is the rate at which blocks are found by any miner (i.e., the rate of transitioning between states in the Markov Chain; Figure 1) assuming a constant hash rate and results in the canonical chain blocks being produced at a rate of 1.

4.2 Expected attacker rewards

The stationary distribution alone is incomplete. To determine the attacker profit for a given cutoff strategy, we calculate their expected profit from each state and multiply those values by the stationary distribution of the Markov Chain to determine the expected profit per unit of time.

▶ **Definition 17** (Per-state attacker rewards, f_i)**.** *Let f_i denote the expected reward of a canonicalized attacker block mined in `State i`.*

To calculate this value, we need to find the expected value of the reward function by integrating the time distribution over the possible paths that include an attacker block claiming rewards arriving during `State i`. We first enumerate all possible paths that result in a canonical attacker block from `State i`; we then integrate the reward function over each path. The following example demonstrates this technique, and we generalize it in [3], Lemma 40.

▶ **Example 18** (`State 3` paths)**.** Consider the rewards arriving after the attacker has a lead of length three. These rewards can be canonicalized in four different ways:

1. the attacker finds the next block, extending their lead to four,
2. the honest parties find the next block, then the attacker finds the subsequent,
3. the honest parties find the next two blocks, causing the attacker to publish their hidden chain, and then the attacker finds the first block after publishing,
4. the honest parties find the next two blocks, causing the attacker to publish their hidden chain, and then the honest parties find the first block after that.

We can succinctly represent these four outcomes using the strings, `A, HA, HHA, HHH`, where `H` & `A` denote honest and attacker blocks, respectively. This example prompts the definition of attacker paths.

▶ **Definition 19** (Attacker paths). *Given `State i` for all $i \geq 2$, there are i distinct paths resulting in the attacker capturing rewards accrued in that state. The paths are enumerated as the string `(H*)A`, where `H` & `A` denote honest and attacker blocks respectively and `H` is repeated $0, 1, \ldots i-1$ times.*

Continuing our `State 3` example, we now calculate the expected reward from each attacker path; adding these together is precisely the value of interest, f_3.

▶ **Example 20** (f_3 continued). Consider the three attacker paths of `State 3`: `A, HA, HHA`. These paths have lengths 1,2,3 and occur with probabilities $\alpha, (1-\alpha)\alpha, (1-\alpha)^2\alpha$, respectively. Thus, we calculate the expected reward as,

$$f_3 = \underbrace{\alpha \int_0^\infty \frac{e^{-t/(1-\lambda)}}{(1-\lambda)} \mathbb{E}_r[R(t)]\, dt}_{\texttt{A}} + \underbrace{(1-\alpha)\alpha \int_0^\infty \frac{te^{-t/(1-\lambda)}}{(1-\lambda)^2} \mathbb{E}_r[R(t)]\, dt}_{\texttt{HA}}$$
$$+ \underbrace{(1-\alpha)^2\alpha \int_0^\infty \frac{t^2 e^{-t/(1-\lambda)}}{2(1-\lambda)^3} \mathbb{E}_r[R(t)]\, dt}_{\texttt{HHA}}$$

Each of these expressions can be viewed as the product of three independent sources of randomness. The coefficients of the integrals are the probabilities of each path determined by the winning miner, which depends on $\vec{m}$. The first expression in the integrand is the PDF of the Erlang Distribution, which measures the sum of i.i.d. exponential random variables (all with rate $1/(1-\lambda)$) to determine the amount of time of the path, which depends on $\vec{t}$. The second expression in the integrand is the expected value over all remaining randomness, r, of the reward function at time t. See Appendix E.2 in [3] for the remaining f_i calculations. Combining the stationary distribution values, p_i, with the per-state expected rewards, f_i, we can calculate the full expected attacker reward.

▶ **Theorem 21.** *The attacker's expected reward is,*

$$\textit{ATTACKER REWARD} = f_0 p_0 + f_1 p_1 + \alpha \sum_{i=2}^{\infty} f_i p_{i-1}.$$

Proof. For `State 0` and `State 1`, we multiply the stationary distribution probability by the expected per-state attacker reward to calculate the contribution to the full attacker reward. For `State i`, $i \geq 2$, we need to avoid double counting the contributions from each state (e.g., you can transition to `State 3` from either `State 2` *or* `State 4`). To account for this we only consider the probability of arriving in each state from the $i-1$ state, which occurs with probability αp_{i-1}. Thus, for each state, we add the contribution to the total attacker reward as $\alpha f_i p_{i-1}$. The resulting value tells us the expected attacker reward per unit time of following a β-cutoff strategy under the static reward function and as a function of α, β, γ. ◀

5 Selfish mining with three reward sources

Selfish mining strategies were analyzed with *just* transaction fees and *just* block rewards in [10, 7], respectively. With the more general notion of miner rewards as defined in Section 2, a similarly general analysis is required to describe the profitability of selfish mining under

different reward schedules. The methodology of path counting and integrating the general reward function established in Section 4 works for any static reward functions. We now instantiate a specific aggregate reward function, which more accurately captures complete miner incentives as they exist in Bitcoin today. This combined reward function, which we denote $\hat{R}$, is composed of (1) a fixed block reward of size C, (2) a linear-in-time transaction fee reward, and (3) an "extra" reward of size E awarded to a block based on the outcome of a Bernoulli trial with probability p. Note that this new reward function considers the sum of each of these rewards, a more representative model of how miners are rewarded in reality rather than considering each of the rewards in isolation. For more straightforward examples of applying the path-counting technique to single-source reward functions, see Appendix H in [3] for only considering block rewards as in [10] and Appendix G for only considering transaction fees as in [7].

5.1 Rewards #1 & #2: block rewards and transaction fees

Each block that a miner produces earns a "fixed block reward" of magnitude C, which is paid directly to the miner as the first transaction in a block. We consider the block reward fixed.[11]

▶ Remark 22 (Block rewards are static). Block rewards are a constant function that doesn't depend on t,

$$R(t) = C. \tag{3}$$

As such, they are static because each block reward is identically distributed no matter the timestamp of the parent block.

The miners are also paid through the contents of the block they create. In particular, the transactions themselves specify a fee[12] to be paid to the miner for including the transaction in the block. As in [7], we start by assuming transaction fees arrive at a deterministic rate and are fully claimable by any subsequent block. See [34] for empirical measurements justifying the linear-in-time transaction fee rewards.

▶ Remark 23 (Deterministic transaction fees with fully claiming blocks are static). Using the [7] definition of fixed-rate transaction fee arrival, we have

$$R(t) = t. \tag{4}$$

This reward is static, as it is deterministic and the same for all blocks (only depending on timestamp of the parent block).

5.2 Rewards #3: non-deterministic extra rewards

We also introduce a third type of reward to our model, motivated by the reality that some blocks have much higher transaction fee revenue than others due to contention. [37] use a similar model to capture high-fee-paying transactions in addition to block rewards; see Section 1.1 for further discussion. Consider, for example, that a new type of transaction can

[11] The Bitcoin block reward is cut in half every four years, which impacts the relative size of the block reward compared to other reward sources. Our model considers the strategies available to miners within the same block reward period.

[12] In Bitcoin, the UTXO model defines a set of inputs and outputs for a transaction. Any balance that doesn't specify an output is claimable by the miner.

become available at a specific block height, and only a fixed amount of those transactions are valid (e.g., the first 10,000 transactions that purchase a specific NFT). To get their transaction included, participants submit bids specifying the fee they will pay to the block producer for higher-priority inclusion (assuming transactions are ordered by fee). This contention for block space leads to much higher revenue for the miner (who serves as the auctioneer) because even assuming infinite block sizes, the finite nature of the transaction type induces the competition (sometimes referred to as a "priority gas auction" [8]). We model this reward as a fixed size "extra reward" of magnitude E available to a miner of a block with probability p (a Bernoulli trial) and independent of time. We refer to this reward function as "Bernoulli rewards."

▶ Remark 24 (Bernoulli rewards are static). Bernoulli rewards are static because each block has the same distribution of rewards according to the outcome of the trial,

$$R(t) = \begin{cases} E & \text{if } X = 1 \\ 0 & \text{otherwise,} \end{cases} \quad \text{where } X \sim \text{Bernoulli}(p). \tag{5}$$

Note that this model doesn't allow for the "predictability" of these Bernoulli rewards. Since miners may know a priori what block height a new set of transactions will arrive at, miners' strategy space would be different than the standard selfish mining strategies we explore below. See Section 6 for more discussion.

▶ **Definition 25** (Reward function instantiation, $\hat{R}$)**.** *Combining the three reward sources (Equations* (3)*–*(5)*), we have the full reward function, which we denote as* $\hat{R}$*,*

$$\hat{R}(t) = C + t + E \cdot \mathbb{1}[X = 1],\ X \sim \textit{Bernoulli}(p). \tag{6}$$

Recall that the path-counting technique defined in Section 4 applies to any static reward function. Since $\hat{R}$ is the sum of three independent, static rewards sources, it is static itself, and thus, we can analyze it. Under $\hat{R}$, we seek to calculate the attacker reward (Theorem 21). Following the structure above, we define the Markov Chain as a function of $\hat{R}$, which induces a stationary distribution p_i before explicitly calculating the per-state attacker reward f_i. For the derivation of the stationary distribution, an instantiation of the general technique in Section 4.1, and the Markov chain under this combined reward function, see [3], Appendix E.3. For the derivation of the expected attacker rewards, an instantiation of the general technique in Section 4.2, see Appendix E.4 in [3].

5.3 Numerical results and discussion

Linear-in-time transaction fees and block rewards

Figure 2 analyzes the simple combination of the linear-in-time transaction fee rewards and block rewards. As expected, the attacker rewards under this combined function interpolates between the two extremes. `Selfish` (in red) shows the percentage of the block rewards collected when always hiding in `State 0` (which is exactly the reward in [10] – see Appendix H in [3] for the full derivation). $\beta-$`cutoff (linear)` (in blue) shows the percentage of the linear-in-time transaction fees collected on the attacker chain when choosing β to maximize this ratio (which is exactly the reward in [7] – see Appendix G in [3] for the full derivation). $\beta-$`cutoff (linear + block)` (in tan) shows the attacker's reward when considering both reward sources together. One interpretation of Figure 2 examines how different reward regimes can lead to dramatically different conclusions regarding the "risk of attack" a

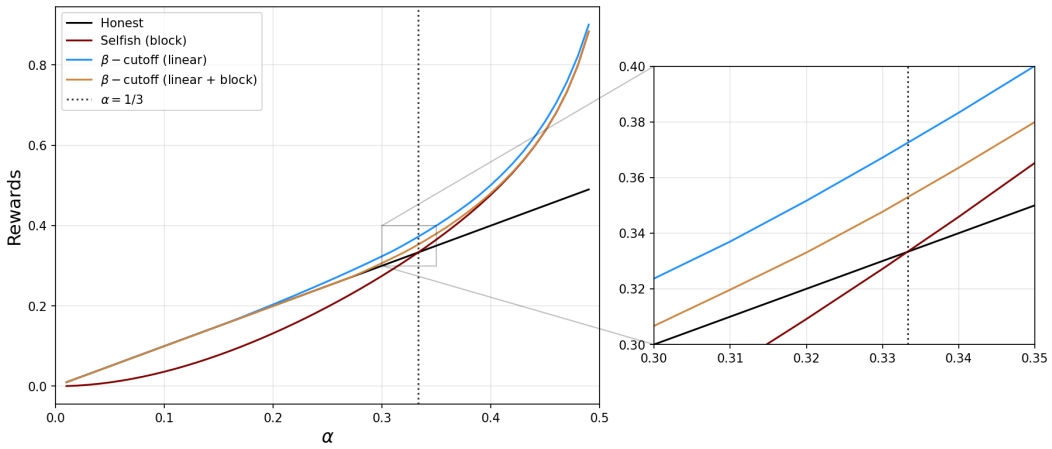


Figure 2 The attacker rewards as a function of α under different metrics of rewards. We consider miners who optimize for block rewards, linear-in-time rewards, or a combination of both. Considering both rewards together paints a more realistic picture of the protocol risk.

protocol faces. In this case, the selfish miner who only optimizes for the ratio of block rewards is not profitable until $\alpha = 1/3$. On the other hand, if we only consider the fraction of linear-in-time transaction fees capturable by a β-cutoff selfish miner, the story looks much worse. In particular, that miner becomes profitable around $\alpha = 0.15$. Considering both rewards results in a more measured conclusion, where the strategy becomes profitable around $\alpha = 0.25$. By varying the relative size of the block reward compared to the per-unit linear-in-time transaction fees, we can thus fully capture the dynamics of both reward models by interpolating between the two strategies, which consider the sub-rewards in isolation. Additionally, this figure can be interpreted qualitatively. We see that the attacker considering both rewards (tan) behaves *less aggressively* than the linear optimizing attacker (blue) for $\alpha \in [0.15, 0.25]$, as the optimal reward in that range is equivalent to honest. Conversely, for $\alpha \in [0.3, 0.33]$, a pure selfish mining strategy would not be profitable; thus, the attacker considering both rewards would be *more aggressive* than the block-reward maximizing miner (who would choose to mine honestly).

Profitability thresholds

Figure 3 shows the value of α at which various strategies become profitable under different reward sources as a function of γ. This extends Figure 3 of [10] to include more strategies. For each γ, we consider the optimal β cutoff for an attacker, maximizing block, linear, and total rewards, respectively. For each candidate α, we check if the optimal β results in a total reward that exceeds the benchmark of the honest performance under that reward function (i.e., the proportional block rewards from honest mining). We find the lowest candidate α such that the rewards exceed the benchmark and identify that as the profitability threshold. Intuitively, this is the fraction of the mining power needed to perform this strategy profitably.

For the pure selfish miner (in green), we see that the profitability thresholds of $1/3, 0.3, 0.25$ for $\gamma = 0, 0.25, 0.5$ are identical to [10]. When considering just linear and block rewards (in blue) and the total rewards (linear + block + bernoulli) (in pink), we see that for all values of γ, the profitability threshold decreases significantly. For example, at $\gamma = 0$, the profitability threshold is reduced from $1/3 \to 0.26 \to 0.18$ (reductions of 22% and 31% respectively) when considering the different reward sources. Similarly, at $\gamma = 0.5$, the profitability threshold is reduced from $0.25 \to 0.18 \to 0.09$ (reductions of 28% and 50% respectively).

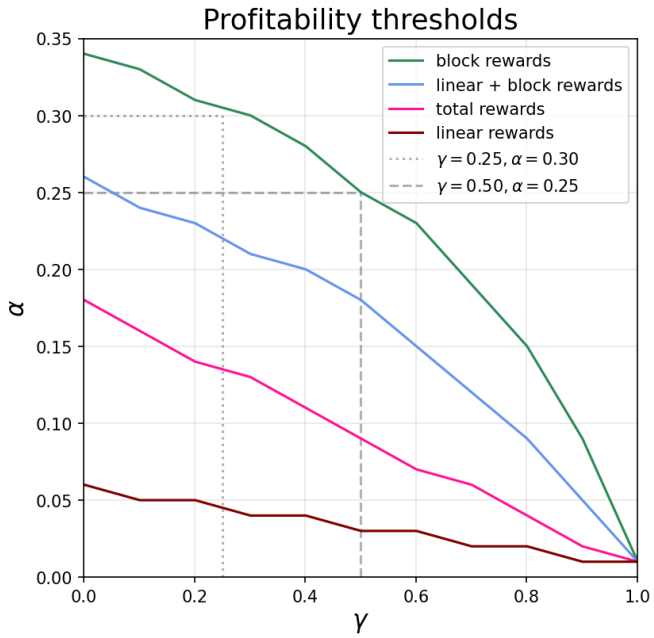


Figure 3 Demonstrating the α at which each strategy becomes profitable over honest as a function of γ. This extends Figure 3 from [10] to include more strategies. Each respective strategy considers profitability when only measuring a subset of the total rewards. For example, `linear + block rewards` (in blue) denotes a $\beta-$cutoff strategy for α profitable if, when selecting β to maximize the sum of linear and block rewards, the expected attacker reward exceeds 2α.

Interestingly, the attacker that only considers linear-in-time transaction fees (shown in red) is profitable (measured against just linear-in-time rewards) for nearly all values of α. While this may seem concerning, we believe that only linear-in-time rewards are not a large enough component of the current Bitcoin incentive structure to warrant concern. The aggregate view of the rewards (e.g., total shown in pink) more accurately represents rewards as they exist in Bitcoin today. These results only consider the case where all three components are approximately equal in magnitude. See Section 6 for a discussion on how choosing the relative sizes of these different reward sources based on empirical values would be a valuable application of our technique.

Measuring Bernoulli reward

Figure 4 examines the profitability of two other mining strategies: optimizing β for Bernoulli rewards (in green) versus optimizing β for the sum of linear, block, and Bernoulli rewards (in tan). Again, the combined rewards interpolate between the Bernoulli and the block-optimizing miners. For the miner maximizing over all three rewards, we normalize them each to have an expected value of 1 per block. The miner who only considers Bernoulli rewards (in green) is always profitable and significantly outperforms honest when $\alpha \geq 0.1$.

Bernoulli rewards aim to capture the volatility of MEV, which depends greatly on exogenous randomnesss (e.g., global capital market volatility). As the block reward continues on it's halving schedule and if transaction fees for Bitcoin transfers remain relatively low, MEV may come to dominate the miner incentives in the long-term. The fact that a miner becomes profitable under MEV rewards at such a low value of α may be a canary in the

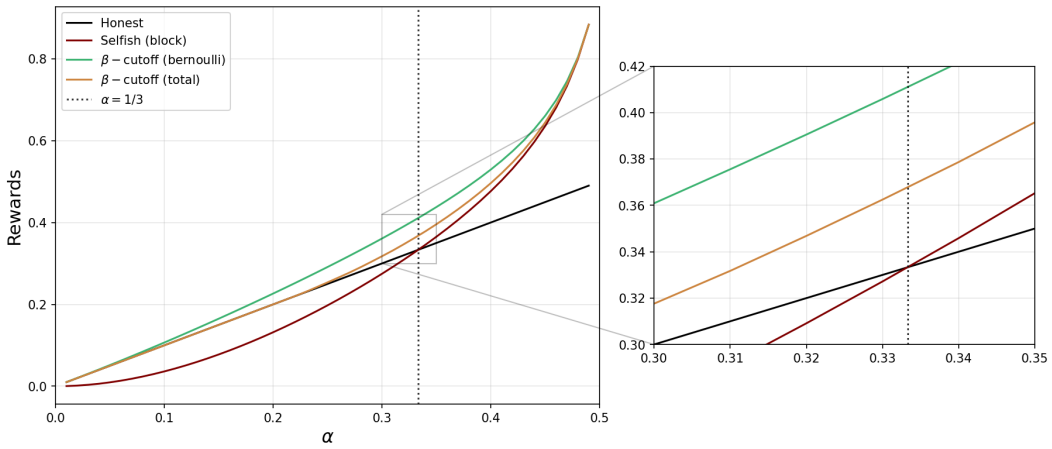


Figure 4 The attacker rewards as a function of α under different metrics of rewards. We consider miners who optimize for block rewards, Bernoulli rewards, and the full $\hat{R}$ containing block, Bernoulli, and linear rewards. Note that the Bernoulli reward-optimizing attacker is profitable for all values of α and meaningfully deviates from honest for $\alpha > 0.1$.

coal mine for the endgame analysis of selfish mining under MEV. We see understanding how Bitcoin MEV may play out under increased financial activity secured by the chain (e.g., through Bitcoin L2s that need to post data to the base chain itself) as one of the most important avenues for future work.

See Appendix F in [3] for more numerical examples and simulation results confirming the accuracy of the technique.

6 Conclusion and future work

We hope this work is a starting point for a more complete picture of participants' incentives in permissionless consensus mechanisms. **Our practical contributions of analyzing selfish mining under a more representative reward model in Section 5** fills a long-standing gap in the selfish mining literature. **Our modeling contribution of general rewards in the NCG and characterizing properties of observed reward sources in Sections 2 and 3** serves as the basis for a richer theoretical treatment of MEV and its implications for consensus. **Our methodological contribution of presenting a path-counting technique to calculate expected attacker profit under general static rewards in Section 4** provides a turn-key solution for future work to analyze selfish mining under newly identified reward sources. To conclude, we outline many potential future research directions.

Applying our methodology more broadly. Our reward instantiation in Section 5 is a reasonably realistic model of reward sources in the Bitcoin blockchain today. The methodology and instantiation represent a significant step in understanding the risk of selfish mining in the presence of multi-faceted rewards, especially since prior work generally considered one reward source at a time. However, empirical analysis may strengthen our results by forming a more nuanced understanding of these rewards in practice (e.g., measuring the relative size and probability of different MEV events). Note that our methodology still applies to any static reward sources that can be analytically calculated using the path-counting technique presented in Section 4. Beyond explicitly using our methodology, our technique has relatively straightforward extensions that can reach beyond static rewards and $\beta-$cutoff strategies.

Extending our methodology. There are several natural extensions to our methods. For example, considering the profitability of β-cutoff selfish mining under non-static reward functions is feasible. Such reward functions depend on additional information not captured in the states of the Markov Chain. However, suppose that the additional information is exogenous to the chain and independent of views. In that case, it is possible to augment the state space of the Markov Chain to include this information. Sme reward sources exhibit "periodic" behavior; in the case of Babylon (Example 1), elevated rewards persisted for a seven-block period. A simple modification of the Markov chain would allow the rewards to depend on whether the system was in a "high" versus "low" regime.

Another extension is to study MDP-based optimal strategies as in [30] rather than β-cutoff selfish mining. [37] demonstrate the impact of changing the reward function on optimal selfish mining profits when considering the combination of block rewards and occasional "whale" transactions, and they note that the resulting large state spaces were intractable with traditional MDP solving tooling and required machine learning. Considering how to more succinctly represent multi-reward state spaces or using the Deep RL approach with more combinatorial rewards are both promising directions. While we consider strategies that make decisions based on the realization of rewards in the *current* block, the broader MDP strategy space can be future-looking; for example, an attacker may want to start creating a hidden chain of several blocks in advance of an anticipated large reward (e.g., from an NFT drop occurring at a specific block height).

A complete picture of consensus incentives. As demonstrated in Examples 1 and 2, modern blockchains have faced and will continue to face distortion of consensus incentives from the application layer handling growing amounts of economic activity. Section 3 is a first step at modeling properties of general reward functions, but applying these properties to MEV beyond the transaction fee (Section 3.1) and LVR (Appendix B.2 in [3]) case studies remains open. It would be interesting to derive a set of necessary and sufficient properties to fully taxonomize MEV. Studying heterogeneity of reward sources was out of the scope of this work, but expanding the properties of reward functions when block producers have highly different rewards realizations is another key future direction.

References

1 Maryam Bahrani, Pranav Garimidi, and Tim Roughgarden. Centralization in block-building and proposer-builder separation. In *International Conference on Financial Cryptography and Data Security*, pages 331–349. Springer, 2024. doi:10.1007/978-3-031-78676-1_19.

2 Maryam Bahrani, Pranav Garimidi, and Tim Roughgarden. Transaction fee mechanism design with active block producers. In *International Conference on Financial Cryptography and Data Security*, pages 85–90. Springer, 2024. doi:10.1007/978-3-031-69231-4_6.

3 Maryam Bahrani, Michael Neuder, and S Matthew Weinberg. Selfish mining under general stochastic rewards. *arXiv preprint arXiv:2502.20360*, 2025. doi:10.48550/arXiv.2502.20360.

4 Maryam Bahrani and S Matthew Weinberg. Undetectable selfish mining. In *Proceedings of the 25th ACM Conference on Economics and Computation*, pages 1017–1044, 2024. doi:10.1145/3670865.3673485.

5 Jonah Brown-Cohen, Arvind Narayanan, Alexandros Psomas, and S Matthew Weinberg. Formal barriers to longest-chain proof-of-stake protocols. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, pages 459–473, 2019. doi:10.1145/3328526.3329567.

6 Agostino Capponi, Ruizhe Jia, and Sveinn Olafsson. Proposer-builder separation, payment for order flows, and centralization in blockchain. *Payment for Order Flows, and Centralization in Blockchain (February 12, 2024)*, 2024.

7 Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 154–167, 2016. doi:10.1145/2976749.2978408.

8 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927, 2020. doi:10.1109/SP40000.2020.00040.

9 Francesco D'Amato and Michael Neuder. Equivocation attacks in mev-boost and epbs. https://ethresear.ch/t/equivocation-attacks-in-mev-boost-and-epbs/15338, 2023. Accessed: 2025-02-05.

10 Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018. doi:10.1145/3212998.

11 Matheus VX Ferreira, Aadityan Ganesh, Jack Hourigan, Hannah Huh, S Matthew Weinberg, and Catherine Yu. Computing optimal manipulations in cryptographic self-selection proof-of-stake protocols. In *Proceedings of the 25th ACM Conference on Economics and Computation*, pages 676–702, 2024.

12 Matheus VX Ferreira, Ye Lin Sally Hahn, S Matthew Weinberg, and Catherine Yu. Optimal strategic mining against cryptographic self-selection in proof-of-stake. In *Proceedings of the 23rd ACM Conference on Economics and Computation*, pages 89–114, 2022.

13 Colin Finkbeiner, Mohamed E Najd, Julia Guskind, and Ghada Almashaqbeh. Sok: Time to be selfless?! demystifying the landscape of selfish mining strategies and models. *Cryptology ePrint Archive*, 2025.

14 Guy Goren and Alexander Spiegelman. Mind the mining. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, pages 475–487, 2019. doi:10.1145/3328526.3329566.

15 Cyril Grunspan and Ricardo Pérez-Marco. On profitability of selfish mining. *arXiv preprint arXiv:1805.08281*, 2018. arXiv:1805.08281.

16 Tivas Gupta, Mallesh M Pai, and Max Resnick. The centralizing effects of private order flow on proposer-builder separation. *arXiv preprint*, 2023. arXiv:2305.19150.

17 Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX security symposium (USENIX security 15)*, pages 129–144, 2015. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman.

18 Charlie Hou, Mingxun Zhou, Yan Ji, Phil Daian, Florian Tramer, Giulia Fanti, and Ari Juels. Squirrl: Automating attack analysis on blockchain incentive mechanisms with deep reinforcement learning. *arXiv preprint*, 2019. arXiv:1912.01798.

19 mempool.space. Bitcoin block 000000000000000000 25c7d9798e97c8f5d8502b03f4bd6b99c365991c5f03b, 2025. URL: https://mempool.space/block/000000000000000000025c7d9798e97c8f5d8502b03f4bd6b99c365991c5f03b.

20 Jason Milionis, Ciamac C Moallemi, and Tim Roughgarden. Automated market making and arbitrage profits in the presence of fees. *arXiv preprint arXiv:2305.14604*, 2023.

21 Jason Milionis, Ciamac C Moallemi, Tim Roughgarden, and Anthony Lee Zhang. Automated market making and loss-versus-rebalancing. *arXiv preprint arXiv:2208.06046*, 2022.

22 Arvind Narayanan. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

23 Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 305–320. IEEE, 2016. doi:10.1109/EUROSP.2016.32.

24 Joachim Neu, Ertem Nusret Tas, and David Tse. Two more attacks on proof-of-stake ghost/ethereum. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, pages 43–52, 2022. doi:10.1145/3560829.3563560.

25 Michael Neuder, Daniel J Moroz, Rithvik Rao, and David C Parkes. Selfish behavior in the tezos proof-of-stake protocol. *arXiv preprint arXiv:1912.02954*, 2019. arXiv:1912.02954.

26 Michael Neuder, Daniel J Moroz, Rithvik Rao, and David C Parkes. Defending against malicious reorgs in tezos proof-of-stake. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 46–58, 2020. doi:10.1145/3419614.3423265.

27 Michael Neuder, Daniel J Moroz, Rithvik Rao, and David C Parkes. Low-cost attacks on ethereum 2.0 by sub-1/3 stakeholders. *arXiv preprint arXiv:2102.02247*, 2021. arXiv:2102.02247.

28 Burak Öz, Benjamin Kraner, Nicolò Vallarano, Bingle Stegmann Kruger, Florian Matthes, and Claudio Juan Tessone. Time moves faster when there is nothing you anticipate: The role of time in mev rewards. In *Proceedings of the 2023 Workshop on Decentralized Finance and Security*, pages 1–8, 2023. doi:10.1145/3605768.3623563.

29 Burak Öz, Danning Sui, Thomas Thiery, and Florian Matthes. Who wins ethereum block building auctions and why? *arXiv preprint arXiv:2407.13931*, 2024. doi:10.48550/arXiv.2407.13931.

30 Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In *Financial Cryptography and Data Security: 20th International Conference, FC 2016, Christ Church, Barbados, February 22–26, 2016, Revised Selected Papers 20*, pages 515–532. Springer, 2017. doi:10.1007/978-3-662-54970-4_30.

31 Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In *International Conference on Financial Cryptography and Data Security*, pages 560–576. Springer, 2022. doi:10.1007/978-3-031-18283-9_28.

32 Caspar Schwarz-Schilling, Fahad Saleh, Thomas Thiery, Jennifer Pan, Nihar Shah, and Barnabé Monnot. Time is money: Strategic timing games in proof-of-stake protocols. *arXiv preprint arXiv:2305.09032*, 2023. doi:10.48550/arXiv.2305.09032.

33 Ertem Nusret Tas, David Tse, Fangyu Gai, Sreeram Kannan, Mohammad Ali Maddah-Ali, and Fisher Yu. Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 126–145. IEEE, 2023. doi:10.1109/SP46215.2023.10179426.

34 Itay Tsabary and Ittay Eyal. The gap game. In *Proceedings of the 2018 ACM SIGSAC conference on Computer and Communications Security*, pages 713–728, 2018. doi:10.1145/3243734.3243737.

35 Sen Yang, Kartik Nayak, and Fan Zhang. Decentralization of ethereum's builder market. *arXiv preprint arXiv:2405.01329*, 2024. doi:10.48550/arXiv.2405.01329.

36 Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. Sok: Mev countermeasures: Theory and practice. *arXiv preprint arXiv:2212.05111*, 2022. doi:10.48550/arXiv.2212.05111.

37 Roi Bar Zur, Ameer Abu-Hanna, Ittay Eyal, and Aviv Tamar. Werlman: To tackle whale (transactions), go deep (RL). In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 93–110. IEEE, 2023. doi:10.1109/SP46215.2023.10179444.

Pool Formation in Oceanic Games: Shapley Value and Proportional Sharing

Aggelos Kiayias
University of Edinburgh, UK
Input Output Global (IOG), Edinburgh, UK

Elias Koutsoupias
University of Oxford, UK

Evangelos Markakis
Athens University of Economics and Business, Greece
Input Output Global (IOG), Athens, Greece

Panagiotis Tsamopoulos
Athens University of Economics and Business, Greece

Abstract

We study a game-theoretic model for pool formation in Proof of Stake blockchain protocols. In such systems, stakeholders can form pools as a means of obtaining regular rewards from participation in ledger maintenance, with the power of each pool being dependent on its collective stake. The question we are interested in is the design of mechanisms, i.e., "reward sharing schemes," that suitably split rewards among pool members and achieve favorable properties in the resulting pool configuration. With this in mind, we initiate a non-cooperative game-theoretic analysis of the well known Shapley value scheme from cooperative game theory into the context of blockchains. In particular, we focus on the *oceanic* model of games, proposed by Milnor and Shapley (1978), which is suitable for populations where a small set of large players coexists with a big mass of rather small, negligible players. This provides an appropriate level of abstraction for pool formation processes that occur among the stakeholders of a blockchain. We provide comparisons between the Shapley mechanism and the more standard proportional scheme, in terms of attained decentralization, via a Price of Stability analysis and in terms of susceptibility to Sybil attacks, i.e., the strategic splitting of a players' stake with the intention of participating in multiple pools for increased profit. Interestingly, while the widely deployed proportional scheme appears to have certain advantages, the Shapley value scheme, which rewards higher the most pivotal players, emerges as a competitive alternative, by being able to bypass some of the downsides of proportional sharing in terms of Sybil attack susceptibility, while also not being far from optimal guarantees w.r.t. decentralization. Finally, we also complement our study with some variations of proportional sharing, where the profit is split in proportion to a superadditive or a subadditive function of the stake, showing that our results for the Shapley value scheme are maintained in comparison to these functions as well.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases Shapley value, Nash equilibria, Price of Stability, Reward sharing schemes, Proof of Stake blockchains

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.21

Related Version *Full Version*: `https://arxiv.org/pdf/2505.04422` [19]

Funding This work has been supported by the H.F.R.I call "Basic research Financing (Horizontal support of all Sciences)" under the National Recovery and Resilience Plan "Greece 2.0" funded by the European Union – NextGenerationEU (H.F.R.I. Project Number: 15877).

Acknowledgements Part of this work' was done while the 4th author was a research intern at IOG. The authors would also like to acknowledge the use of resources of the "Theory, Economics and Systems Lab" at the Athens University of Economics and Business.

© Aggelos Kiayias, Elias Koutsoupias, Evangelos Markakis, and Panagiotis Tsamopoulos;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Permissionless blockchain protocols base participation on the resources that parties possess such as their *computational power* (as in Bitcoin, [26]), or their *stake* in the system, measured by the number of digital coins they own (as in Ethereum [8], see e.g., [9]). This resource-based operation opens up the possibility for pooling resources together and having multiple resource holders engage as a single entity in protocol operation. Pooling resources can have both positive effects for the system, such as reducing the variance of rewards awarded to participants, as well as negative ones, leading to centralization with a handful of large pools controlling the protocol. Importantly, viewing the blockchain protocol as a mechanism, the question that arises is what are the objectives of this mechanism design problem and how it is possible to realize them.

As a running paradigm, we will use throughout our work the pool formation process in Proof of Stake blockchains. In such systems, stakeholders are attracted to coalitions as a means of obtaining rewards from participation in block production. The power of such a (stake) pool is then dependent on the collective stake (up to some threshold) that its members possess, which affects the probability that a pool operator is selected as a validator or block producer. Creating or joining a pool can be done either explicitly via options that the protocol itself provides (referred to as *onchain pooling*, for example, in the Cardano blockchain) or by agreements among the stakeholders via smart contracts or even via informal agreements. In fact, for the Ethereum protocol, there even exist platforms[1] (e.g. Rocketpool or Lido) that facilitate the grouping of stakeholders into pools. Essentially these platforms match smaller stakeholders to larger ones, so as to collectively accrue the amount of 32 ETH, required for a block validator.

Once a pool forms, the central question of interest, which is the focus of our work as well, is how should the pool members split the received rewards among themselves. This has led to an interesting and currently active research agenda, with several (and sometimes conflicting) desiderata that one would like to satisfy. Among these, a significant desideratum in blockchain communities is to promote decentralization. Viewed as a game, this means that we would like the reward scheme to induce equilibria with a relatively high number of pools. At the same time, we would also like to achieve resilience to Sybil attacks [15], since the pseudonymity of blockchain systems may allow seemingly independent pools controlled by the same entity. Although completely eliminating this may be too much to hope for, cf. [21], one could aim for mechanisms that attempt to discourage Sybil behavior.

If we look at current implementations, both in onchain pooling and in mediating platforms, the deployed reward scheme is typically very simple: the pool operator may keep a certain amount (accounting for a profit margin and/or operational costs) and the remaining amount is split in a proportional manner, based on stake contribution. This puts forth the natural question of whether the proportional scheme is the best thing to do. To answer this, it seems reasonable to also consider alternative candidate solutions from economic theory. Concepts from cooperative game theory, like the core [18] or the Shapley value [30], to name a few, have been extensively studied in almost any setting that involves cost or profit sharing. Despite however the popularity of such notions within game theory and economics, there have been surprisingly very few attempts to incorporate them in models tailored to blockchain protocols, such as [23, 13] (discussed further in our related work section). More generally, there is only a limited number of works that provide comparisons among different reward schemes, with some notable exceptions being [7, 12, 10].

[1] To give an idea of their volume and popularity, their total market capitalization is currently over 30 billion USD.

1.1 Our contribution

The main question we are interested in is: *How does the choice of a reward sharing scheme within a pool affect performance w.r.t. decentralization, and w.r.t. resilience to Sybil attacks?*

To study this question, we consider the model of *oceanic games*, originally introduced by Milnor and Shapley [24] in the context of cooperative games. This model captures populations where (atomic) players with moderate or large stake coexist with smaller non-atomic players of negligible stake (the ocean). We believe this model is particularly suitable for certain blockchain environments, as also advocated in [22]. Empirical studies have shown that the stake distribution in many blockchain systems follows a power-law distribution. Therefore, the majority of users control only a small fraction of the total stake, and this aligns well with the oceanic game framework.

Within this model, we initiate a comparison between two reward schemes; the first one is based on the *Shapley value*, as one of the most prominent concepts of profit sharing from game theory. The second one is *proportional sharing*, used extensively in practice.

We analyze the corresponding *non-cooperative games*, induced by these reward schemes, where each player chooses which pool to join. We quantify the attained decentralization through a *Price of Stability* analysis on the set of Nash equilibria. The Price of Stability measures the quality of the *best* Nash equilibrium by comparing it to the ideal non-selfish solution. We employ the Price of Stability rather than the more pessimistic Price of Anarchy, as it is a more appropriate measure in our setting (where, as we will exhibit, some bad but quite unnatural equilibria may arise). The Price of Stability corresponds to an equilibrium that maximizes social welfare, and players will likely attempt to converge to it (or near it). Conversely, the Price of Anarchy corresponds to an equilibrium that minimizes welfare, and it is improbable that players would stabilize at such an equilibrium. With this in mind, we show that the proportional scheme always has an optimal equilibrium (i.e. the Price of Stability equals 1). Not far from this, the Shapley value scheme attains a constant Price of Stability bounded by 4/3. We complement this with an analysis for the purely atomic model, where the Shapley scheme deteriorates, albeit not by a lot, achieving a bound of 2.

We also consider the resiliency of such schemes against *Sybil attacks*, where the players can split their stake and contribute to multiple pools. Although none of the schemes can completely avoid Sybil attacks, we highlight a potential drawback of the proportional scheme, which can be avoided by the use of the Shapley value. In particular, we prove that under the equilibria of the Shapley scheme identified in the previous sections, the players have no incentive to perform Sybil attacks. On the contrary this is not true for proportional sharing, which is demonstrated with rather simple examples.

Finally, we include a study of some non-linear variations of the proportional scheme. Specifically, we consider the *proportional to square roots* and *proportional to squares* mechanisms, which use subadditive and superadditive reward functions, respectively. We exhibit that both have their drawbacks, regarding our desiderata. Namely the proportional to squares scheme has unbounded Price of Stability whereas the proportional to square roots scheme is more vulnerable to Sybil attacks.

Overall, while our results reveal certain advantages of the proportional scheme, they also discover important drawbacks (cf. Sybil attacks above). At the same time, the Shapley value emerges as a competitive alternative, that can bypass some of the negative aspects of proportional splitting, while maintaining comparable guarantees w.r.t. the Price of Stability for decentralization.

1.2 Related Work

In terms of the model we use, oceanic games were introduced by Milnor and Shapley [24]. In their work, they define the Shapley value and study its properties for a class of cooperative games, namely weighted voting games, when the set of players contains both atomic and non-atomic players. Especially for blockchain systems, the work of [22] revisited and advocated the use of oceanic games for mining. They also studied further aspects concerning the stability of the grand coalition of all players. In our work we consider a non-cooperative model of oceanic games, where we care to evaluate the Nash equilibrium outcomes (i.e., partitions into pools).

Our work is also related to the literature of weighted voting games in social choice theory. This is a class of cooperative games, where the value of a coalition is threshold-based, and reward sharing is viewed as determining the voting power of each player. The main solutions that have been proposed for such games are the core, the Shapley value and the Banzhaf index, and we refer to [11] for an overview of results. The main difference from our model is that we have a non-cooperative game where each strategy profile essentially induces a collection of weighted voting games, one per pool.

The use of solution concepts from cooperative game theory in blockchain applications has been limited so far. In [23] the notion of the core was studied for a model of mining in Bitcoin. They obtained a negative result that the core is empty and hence there is always a motive for some players to deviate. The Shapley value as a reward scheme in mining pool games has also been proposed in [13]. The focus of that work was however on its computational aspects and no game-theoretic analysis was given.

Some closely related works, along a similar spirit are [7] and [20], which present different reward sharing schemes for the Proof of Stake setting. Our difference is that they focus on how the system should allocate rewards to the pools, which are then split in a proportional manner among pool members (after the operator takes a cut). In our case we study an orthogonal question of how the pool should allocate the received rewards among its members. Hence it can be seen as complementary direction to [7].

There are also several papers discussing decentralization from various angles. The work of [4] introduces a model where the utility function of the players is decentralization conscious, in terms of effort exerted. In [6] a different model is analyzed where effort corresponds to committed stake, and is related to Tullock contests. Another attempt more tailored to Proof of Work systems is presented in [1], which highlights the heterogeneity of the costs invested in hardware. Finally, reward schemes are also studied in [16] under a dynamic model focusing on the "rich-get-richer" phenomenon in the long run over blockchain systems. The main difference of all these papers with our work is that they do not study explicitly any pool formation process, as they focus more on the individual effort or possessed resources in their proposed models.

One of the few works that formally studies the properties of proportional sharing for blockchain protocols is by [12]. This is an axiomatic study, tailored to Proof of Work protocols, and demonstrates why the proportional rule can have favorable performance (but in a model without pool formation). Other variants that are studied include the proportional to square roots and the proportional to squares schemes that we also consider. A follow up work with a further axiomatic study is provided in [10]. These models however do not consider any pool formation aspects, and there is no discussion on the performance of these schemes w.r.t. decentralization.

Finally, Sybil attacks have been a major concern for any environment where identity cannot be traced effectively [15]. In the context of auctions, it is often referred to as "false-name" bidding [3]. It is natural that blockchains also offer opportunities for such

manipulations, and general resilience against Sybil attacks seems a utopia [21]. Nevertheless some positive results have also been obtained towards having reward schemes that can potentially deter such behaviors, see e.g., [7].

2 Model and definitions

2.1 The oceanic model with atomic and non-atomic players

We consider a population of stakeholders, say N, in a Proof of Stake protocol, who will be referred to from now on as the players of the underlying game. We will mostly focus on the model of oceanic games, as defined by [24], where the population is split into two types of players. Namely, $N = N_a \cup N_s$, where $N_a = \{1, \ldots, n\}$ is a set of n atomic players. Each $i \in N_a$, possesses stake equal to a_i. The remaining players are *small* players, each with a tiny stake $\epsilon > 0$. We let $\epsilon \to 0$, and we can view N_s as a continuum of infinitesimally small, non-atomic players (i.e., the ocean). We denote by L the total stake possessed by the ocean, which equals the measure of N_s. For convenience we imagine N_s as arranged in the interval $[0, L]$. With this in mind, any subinterval $I \subseteq [0, L]$ corresponds to a mass of players possessing stake equal to the length of I. This model shares some characteristics with the atomic and non-atomic models in congestion/routing games [28].

Available strategies. Every player is considering either to operate a pool or join the pool of some other player. The corresponding game is as follows: every player i needs to choose some index $j \in N$. If she chooses herself, i.e., $j = i$, it means that she is starting her own pool, otherwise she joins someone else's pool. A strategy profile $\mathbf{x}$ specifies a choice of strategy x_i for each player $i \in N$. We say that a strategy profile is *valid* if whenever some player i chooses $x_i = j$, with $i \neq j$, we also have $x_j = j$, i.e., users who do not operate a pool do choose a valid pool to join. Technically, non-valid profiles could also arise in this model, but we will exclude them from our equilibrium analysis, as they lack a natural real life interpretation.

Pool rewards. A valid strategy profile induces a partition of the players into disjoint sets (pools), $\Pi = (S_1, \ldots S_k)$, for some k, so that $\cup_j S_j = N$. Given such a partition, each pool S_j receives a reward $\rho(S_j)$, determined by the execution of the blockchain protocol. In this work, we consider a simple reward function, as an attempt to model the rationale behind the operation of Ethereum or other protocols with a similar design. In Ethereum, a stake holder needs to have 32 ETH in order to register as a validator and claim rewards. This gives rise to a threshold-based scheme, where a pool can obtain rewards when its total stake exceeds a given threshold h. To define this formally, for a given pool $S \subseteq N$, let $m(S)$ be the total stake that it possesses. This consists of two terms; the total mass of stake by the non-atomic players, which equals $|S \cap N_s|$, and the stake of the atomic players, which is equal to $\sum_{i \in S \cap N_a} a_i$. Then, given the threshold h, the reward is equal to:

$$\rho(S) = \begin{cases} 1, & \text{if } |S \cap N_s| + \sum_{i \in S \cap N_a} a_i \geq h \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

We say that a pool $S \subseteq N$ is a winning pool if $\rho(S) = 1$.

▶ Remark 1. For the remainder of the paper, we make the assumption that $a_i < h$, for any atomic player $i \in N_a$. The main reason is that, as also considered in the initial model of oceanic games by [24], we want to focus on players who need to collaborate with other people in order to form a successful pool. If there exist players with $a_i > h$, given our

threshold-based reward function ρ, intuitively it makes sense for them to break their stake and run their own pools with avatars of size h each, and participate in our game with the remainder $a_i \ mod \ h$. For a further discussion on this, see the full version of this paper [19].

Reward sharing scheme within a pool. Once the pools are formed, the pool members need to split the obtained reward according to some agreed upon scheme.

▶ **Definition 2** (Reward sharing schemes). *Given a pool $S \subseteq N$ and its reward $\rho(S)$, a reward sharing scheme[2] specifies a payment allocation $p_i(S)$ to each pool member $i \in S$, dependent on the stake that i contributes to S, so that all the payments sum up to $\rho(S)$.*

▶ Remark 3. We focus on scenarios where the same reward scheme is applied to all formed pools. This is the case for example in protocols with onchain pooling (e.g., Cardano) but also in many of the platforms that facilitate pooling (e.g., for Ethereum). We do not consider secondary effects where pool members may re-negotiate their payments later on, among themselves.

In our model, we have not explicitly defined the cost that a pool may bear towards participating in the protocol. We essentially assume that $\rho(S)$ is the reward that is left for being shared, after subtracting the amount that the pool operator may claim to cover her operational costs.

2.2 Equilibria, Decentralization and Price of Stability

Under any strategy profile, we define the utility of each player to be the reward that she receives. For a non-valid profile, the reward of a player who does not end up in a pool is simply equal to 0. We are interested in profiles that induce a partition into winning pools, and we will denote by $u_i(\Pi)$ the utility of a player under a formed partition $\Pi = (S_1, \ldots, S_k)$.

▶ **Definition 4.** *Given a game G, a partition Π into winning pools is a Nash equilibrium if for every player $i \in N$, $u_i(\Pi) \geq u_i(\Pi')$, for any partition Π' that arises from Π if i moves to a different pool or opens a new pool on her own.*

A major concern in blockchain design is whether the actual execution of the protocol by selfish entities can result in a decentralized configuration. As a metric for this, we use the number of winning pools that are formed at an equilibrium. Given a game, the ideal scenario is that the players split into pools of size exactly h. Since this may not always be possible, we let $OPT(G)$, for a game G, be the maximum number of winning pools that can form over all possible partitions, i.e.,

$$OPT(G) = \max\{t : \exists\ \Pi = (S_1, \ldots S_t) \text{ s.t. } m(S_i) \geq h \ \ \forall i \in [t]\} \tag{2}$$

In the purely atomic case, when $N_s = \emptyset$, finding $OPT(G)$ is the dual version of the well known Bin Packing problem, referred to as dual Bin Packing in [2], and is easily seen to be intractable. An interesting question that arises is whether equilibria can lead to an approximate solution. In order to evaluate Nash equilibria, the usual metrics are the Price of Anarchy (PoA) and the Price of Stability (PoS). The Price of Anarchy is not an appropriate metric here as it is unavoidable to have completely centralized equilibria, which are however rather unrealistic. This is captured below.

[2] Given the form of $\rho(S)$, each pool can be seen as a weighted voting game [11] and the reward to each pool member can be also interpreted as her power within the pool.

▷ Claim 5. When $a_i < h$ for all $i \in N_a$, the grand coalition consisting of all players is an equilibrium under any reward scheme, and hence the Price of Anarchy is $\Omega(n)$.

We will therefore turn our attention to the Price of Stability, defined as the ratio between the quality of the optimal solution and the quality of the best Nash equilibrium. Given a partition Π, let $W(\Pi)$ be the number of winning pools that are formed under Π.

▶ **Definition 6.** *The Price of Stability (PoS) of a game G is computed as* $\min_{\Pi \in NE} \frac{OPT(G)}{W(\Pi)}$, *where NE is the set of all equilibrium partitions.*

The Price of Stability appears to be an appropriate quality benchmark for the equilibria of the games we consider. As the Price of Stability aligns with maximizing social welfare, the dynamics of threshold-based mechanisms would likely guide players towards more decentralized outcomes.

Finally, we note that all missing proofs in the remainder of the paper can be found in the full version of our work [19].

3 Analysis of the Shapley value reward sharing scheme

In this section, we study the scheme, where the reward in each winning pool is distributed using the Shapley value of each player. The Shapley value was introduced in [30] as the unique reward allocation rule that satisfies a particular set of axioms. Ever since, there is hardly any economic setting involving the sharing of costs or payoffs where the Shapley value has not been considered. Prominent examples are the design of truthful cost-sharing mechanisms [25] and the study of power indices in voting games [11]. Furthermore, despite the fact that the exact computation of the Shapley value is a #P-hard problem [14], there are very efficient sampling-based approximations that work quite well [5, 17]. Even further, the Shapley value has been deployed successfully in practice, in machine learning applications, as a way to identify important parameters in the training of neural networks [29].

3.1 Definition of the Shapley value in the oceanic model

The standard definition and use of the Shapley value is for the purely atomic model, when $N_s = \emptyset$. It is very instructive to recall first how the scheme works there, without any small players.

The atomic model. The rationale for the Shapley value is that within each pool, *each stakeholder gets their expected marginal contribution* if the members of the pool would arrive in a random order. Hence, one needs to take the average over all permutations. The Shapley value for a player i, when she belongs to a pool S is therefore given by the following formula:

$$\phi_i(S) = \sum_{T \subseteq S \setminus \{i\}} \frac{|T|!(|S| - |T| - 1)!}{|S|!} (\rho(T \cup \{i\}) - \rho(T)) \tag{3}$$

This is a valid scheme since it is shown already in Shapley's original work that $\sum_{i \in S} \phi_i(S) = \rho(S)$.

▶ **Example 7.** Consider a game G with atomic players and stake distribution $\mathbf{s} = (3, 1, 1, 1, 1, 1)$, and let the threshold be $h = 4$. The optimal pool formation is to have two pools, each with total weight 4. Thus $OPT(G) = 2$. However, this is not a Nash equilibrium, because the large player prefers to switch to the other pool: the reward of the

large player in the pool with one small player (with stake vector $(3, 1)$) is $1/2$, while if she joins the other pool with the 4 small players, so that the pool becomes $(3, 1, 1, 1, 1)$, it is $3/5$. Therefore, only the grand coalition can be a Nash equilibrium (which is a rather extreme case, since we will see that in bigger games other equilibria also arise). ◀

Extension to the oceanic model. We discuss now how to compute the Shapley value, when we include non-atomic players. Consider a pool with non-atomic players of total mass k and t atomic players with stakes $(a_1, \ldots, a_t)$; the total stake of the pool is $k + a_1 + \cdots + a_t$. For convenience, we will often refer to the atomic players as *large* players and the non-atomic ones as *small* players. Imagine that the small players are arranged in the interval $[0, k]$. Then a random arrival of the players corresponds to placing arrival times $L_{i_1}, L_{i_2}, \ldots, L_{i_t}$ for the atomic players on the interval $[0, k]$. The arrivals can be viewed as filling in the stake $k + a_1 + \cdots + a_t$, say starting at 0, from left to right, as follows: first, a mass of small players (possibly empty) arrives, filling in the interval $[0, L_{i_1}]$; then the first large player arrives, so that the stake of the current subset of players accrues to $L_{i_1} + a_{i_1}$. This is followed by another mass of small players; then a second large player arrives at L_{i_2}, accumulating now a total stake of $L_{i_2} + a_{i_1} + a_{i_2}$; and so on. For random arrivals, the points $L_{i_1}, L_{i_2}, \ldots, L_{i_t}$ are *uniformly distributed within the interval of length* k. The reward of each large player i is then the probability that the threshold h falls within the interval that makes player i pivotal. To be more precise, for a given arrival order, and for an atomic player i, let $P(i)$ be the set of other atomic players preceding i in the order. Then, if L_i is the random arrival time of player i, the Shapley value is given as follows:

$$\phi_i(S) = Pr\left[\sum_{j \in P(i)} a_j + L_i < h \leq \sum_{j \in P(i)} a_j + L_i + a_i\right] \tag{4}$$

After computing the reward of all large players, the remaining reward is distributed evenly among the small players. Hence, for the small players, all that matters is the reward per unit of stake, which is what we will consider in the game-theoretic analysis that will follow.

3.2 The Price of Stability in the oceanic model

For a pool with many large players along with a mass of small players, the exact formula for the Shapley reward might be too complex to derive analytically. For our study on the Price of Stability however, it will suffice to only consider equilibria in which every pool has at most one large player. Hence, when arguing about equilibria and deviations to other pools, it is sufficient to be able to compute the rewards of pools with at most 2 large players. This is done in the following lemma, which we will use repeatedly.

▶ **Lemma 8.** *Consider a pool with a mass of non-atomic players equal to k and total stake at least h.*

- *If $k \leq h$ and the pool has a single large player with stake a, the Shapley value of this player is $(k + a - h)/k$.*
- *If $k > h$ and the pool has a single large player with stake a, the Shapley value of this player is a/k.*
- *If $k \leq h$ and the pool has two large players with stakes a_1, a_2, respectively, the Shapley value of the player with stake a_1 is equal to (with an analogous formula for the second player)*

$$\frac{(h - a_2)^2 - (\max(0, h - a_1 - a_2))^2 + (\max(0, a_1 - h + k))^2}{2k^2}.$$

In all 3 cases above, the remaining reward is distributed equally to the non-atomic players.

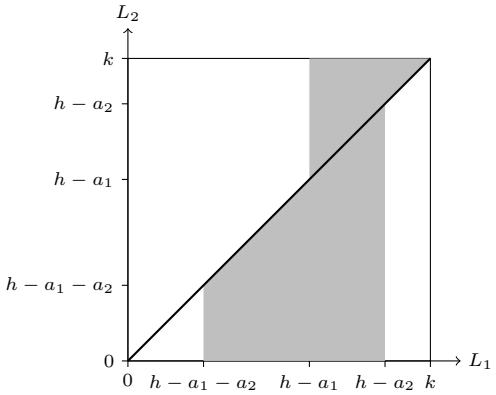


Figure 1 Calculation of the Shapley value for the first player in a winning pool of two large players with stakes a_1, a_2 and non-atomic mass of $k \leq h$. It is easy to verify that player 1 is pivotal when (L_1, L_2) is in the gray areas. The gray triangle shows the pivotal cases when $L_1 \leq L_2$, whereas the trapezoid corresponds to the cases with $L_1 \geq L_2$. The reward of player 1 is then the gray area (equal to $((h-a_2)^2 - \max(0, h-a_1-a_2)^2 + \max(0, a_1-h+k)^2)/2$) divided by the total area (equal to k^2). When $h - a_1 > k$, the gray triangle disappears. Similarly, when $h - a_1 - a_2 < 0$, the gray trapezoid becomes a triangle.

Proof. Consider a pool with a single large player and with $k \leq h$. We apply the procedure described in Section 3.1 on how to determine the reward of the large players. If L_1 is the random arrival of the single player within the interval $[0, k]$, then the player is pivotal, only when $L_1 \in [h - a, k]$. The probability of this event is $(k - (h - a))/k$, which proves the first part of the lemma.

Suppose now that $k > h$. In that case, the large player is pivotal, only when $L_1 \in [h-a, h)$, and this event occurs with probability a/k.

Finally, for the last part of the lemma, the formula for a pool of two large players, is a bit more complicated. If L_1, L_2 are their random arrival times, we distinguish two cases: $L_1 \leq L_2$ and $L_2 < L_1$. In the first case, player 1 arrives before the second player, and hence she receives a reward when $L_1 \in [h - a_1, k]$. In the second case, player 2 arrives before player 1, which means that for player 1 to be pivotal, it should hold that $L_1 + a_2 \in [h - a_1, h]$. This implies that $L_1 \in [h - a_1 - a_2, h - a_2] \cap [0, k]$. The ranges for the values of L_1, L_2, under which player 1 is pivotal, are depicted in Figure 1. The probability that a random point (L_1, L_2) falls into these areas gives the reward of player 1. ◀

We exhibit below an example to demonstrate how we can construct relatively good equilibria in this game.

▶ **Example 9.** Fix the threshold at $h = 3$ and consider a population where all large players have stake $a = 2$. Suppose also that there is enough mass of small players, so that we can create a partition into two types of pools. The first type has a single large player and non-atomic players with total mass $k = 2$ (hence each such pool has total stake equal to 4). The second type has only non-atomic players with mass $l = 4$. Let us argue that this partition is a Nash equilibrium.

The reward of each large player, as given by Lemma 8, is $(a - h + k)/k = 1/2$. If the player switches to a pool with only non-atomic players, her reward (by applying now the second part of Lemma 8) will be $a/l = 1/2$, so the player has no reason to switch. Also, if the player switches to a pool with another large player, her rewards will be at most $1/2$, since she will share the reward with another identical large player, and there is also a mass

of non-atomic players, who also receive some part of the profit. Furthermore, we can verify that for the non-atomic players, the reward per unit of stake is the same, and equal to 1/4, in both types of pools. Therefore this partition is a Nash equilibrium. Every pool at this equilibrium has a total stake equal to 4, which shows that for this instance the PoS is at most $4/h = 4/3$. ◀

The previous example also leads to a lower bound on the Price of Stability.

▷ Claim 10. The Price of Stability in the oceanic model, is at least $\frac{4}{3}$.

Proof. We consider a special case of Example 9, with $h = 3$. The set of players consists of only one atomic player with stake $a = 2$ and a mass of small players, equal to $m = 4t + 2$, where t is a large integer. The optimal partition for this game would be to have one pool including the atomic player and one unit of non-atomic players, and then have the remaining players form pools of size 3, except the last pool which will have one additional unit. Hence $OPT(G) = 1 + 4t/3$ (assuming that $4t$ is divisible by 3). By the construction of Example 9 above, we know that there exists an equilibrium in which a mass of size 2 forms a pool with the large player, and the remainder $4t$ units of small players form t pools of size 4 each.

For the sake of contradiction, assume that there is a better equilibrium formation that achieves a ratio better that $\frac{4}{3}$, compared to the optimal partition. In this equilibrium, the large player has to be paired with some mass, say of size k, of small players and all other pools have to consist only of small players of mass $l \geq h = 3$. The latter pools have to be of the exact same size otherwise the small players would have an incentive to move. In order for this formation to achieve a better than 4/3 Price of Stability, it has to be the case that $3 \leq l < 4$. Since, it is an equilibrium it must hold that the large player should not have a motive to move to a pool consisting only of small players, hence by Lemma 8, we have that $\frac{k-h+a}{k} \geq \frac{a}{l} \Leftrightarrow l(k-h+a) \geq a \cdot k \Rightarrow l(k-1) \geq 2k \Rightarrow k > 2$, where the implication holds from the fact that $l < 4$, $a = 2$ and $h = 3$. Moreover, no small agent who has been paired with the large player should want to move to a pool of only small players, and vice versa, so that $\frac{1-\frac{a-h+k}{k}}{k} = \frac{1}{l} \Rightarrow \frac{1}{k^2} = \frac{1}{l} \Leftrightarrow l = k^2$. Combining the facts that $k > 2$ and $l = k^2$, we obtain that $l > 4$ and thus we arrive to the desired contradiction.

Therefore, the Price of Stability is lower bounded by $\frac{4t/3+1}{t+1}$, which tends to 4/3 as $t \to \infty$. ◁

Our main result in this section is a matching upper bound, provided that all atomic players have a sufficiently large stake[3] and that the total population of small players is sufficiently large.

▶ **Theorem 11.** *In the oceanic model, and given any constant $\epsilon > 0$, the Price of Stability for the Shapley scheme, is at most $4/3 + \epsilon$, when all large players have stake in $(h/4, h)$ and there is a sufficiently large mass of non-atomic players (dependent on ϵ and the total stake of the large players).*

The remainder of this subsection is devoted to the proof of Theorem 11. Before we proceed, we comment on the type of equilibria that we construct to establish the proof. Following Example 9, we show that such games always possess equilibria where each large player forms a pool together with some mass of non-atomic players and with no other large

[3] The theorem requires large players to hold at least $h/4$ stake, which, coincidentally, is also the minimum stake requirement for validators in the Rocketpool platform.

players. We find such equilibria to be natural for the blockchain scenarios that we study (certainly more natural than the grand coalition equilibrium from Claim 5), in the sense that bigger players tend to avoid each other and form pools with less important users. They provide each big player with *skin in the game*, as each of them is the only entity with high enough stake in their pool. Furthermore, they reduce the question on the Price of Stability to upper bounding the mass of non-atomic players in each pool, as demonstrated in the sequel. Finally, we note that some games may also have other equilibria, with more than one large player in some pools, but the existence of more equilibria could only positively (if at all) affect the Price of Stability.

To prove Theorem 11, we start with the following lemma, that provides a characterization for the particular type of equilibria we are after.

▶ **Lemma 12** (Equilibrium conditions). *Consider a partition of the players into winning pools, where every large player $i \in N_a$ is in a pool with only small players of mass $k_i \leq h$, with $a_i + k_i \geq h$, and all remaining small players are in pools of total mass equal to l, with $l \geq h$. Such a partition is at a Nash equilibrium if and only if the following conditions are satisfied for any large players i, j :*

$$\frac{h - a_i}{k_i^2} = \frac{1}{l} \tag{5}$$

$$\frac{k_i + a_i - h}{k_i} \geq \frac{a_i}{l} \tag{6}$$

$$\frac{k_i + a_i - h}{k_i} \geq \frac{(h - a_j)^2 - (\max(0, h - a_i - a_j))^2 + (\max(0, a_i - h + k_j))^2}{2k_j^2} \tag{7}$$

Proof. We focus first on the non-atomic players. Since these players are infinitesimally small, a unilateral deviation of any such player to a different pool does not have any effect on the reward per unit of stake in the pool that she moves to. Therefore, at a Nash equilibrium, all pools with no large player must have the same stake, equal to l in the pools we consider, otherwise the non-atomic players will have an incentive to move to the smallest of these pools.

We consider now deviations of small players either from a pool with no large players to a pool with a large player or the opposite. Constraint (5) rules out such deviations. Indeed, by Lemma 8 the reward that is left over for the non-atomic players in the pool of player $i \in N_a$ is $1 - \frac{a_i - h + k_i}{k_i}$. Since the pool has a mass of size k_i of non-atomic players, the reward per unit of stake for them is equal to $(h - a_i)/k_i^2$. In pools with no large players, the corresponding reward per unit of stake is $1/l$. These two rewards must be equal for non-atomic players in either pool to have no incentive to switch.

Constraints (6) and (7) express the fact that an atomic player i has no reason to switch neither to a pool of non-atomic players nor to the pool of any other atomic player j, respectively. These follow directly by Lemma 8. ◀

We can now show that it is possible to construct such equilibria for a sufficiently large population of non-atomic players.

Proof of Theorem 11. Consider a game that satisfies the premises of the theorem. We will show that there exists a partition into winning pools that is a Nash equilibrium and in which every pool has at most one large player. We will establish that this equilibrium achieves $PoS \leq 4/3$.

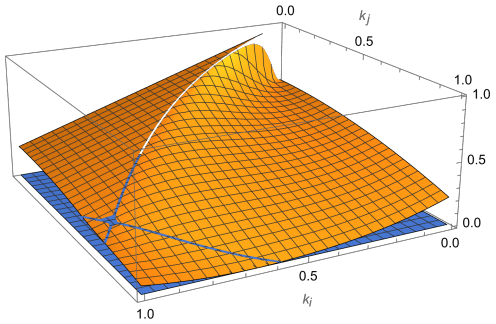


Figure 2 The function $f(k_i, k_j)$ of Equation (8). For $k_i, k_j \in [0, 1]$, the minimum value is 0 and it is achieved when $k_i = 2/3$ and $k_j = 1$.

Without loss of generality (simply by scaling), from now on we assume that $h = 1$. We will show that we can find numbers $k_1, \cdots, k_n$, so that we can fill in the pool of each large player $i \in N_a$, with a mass of non-atomic players, equal to k_i, and ensure that the equilibrium constraints of Lemma 12 are satisfied. Let us also make the simplifying assumption that the mass of small players is such that after allocating a mass of size k_i to the pool of each large player i, the remaining mass can be partitioned into pools without large players and with stake of exactly $l = 4/3$. We will revisit this assumption at the end of the proof.

Note that we need $k_i \leq h$ to be able to use Lemma 12. The first constraint (Constraint (5)) is an equality and tells us precisely how much k_i should be in terms of a_i. By solving for a_i, for each i, it should necessarily hold that: $a_i = h - k_i^2/l = 1 - 3k_i^2/4$. Since by the premises of the theorem, $a_i \in (1/4, 1)$, we get that $k_i \in (0, 1)$, i.e., $k_i \leq h$. An immediate implication of the assumption that $l \geq 4/3$ is that Constraint (6) is satisfied for every k_i because it reduces to $(3k_i/4 - 1/2)^2 \geq 0$.

It remains to show that Constraint (7) is also satisfied. By substituting a_i and a_j from Constraint (5), we can see that this last constraint becomes equivalent to $f(k_i, k_j) \geq 0$, where

$$f(k_i, k_j) = 1 - \frac{3}{4}k_i - \frac{9}{32}k_j^2 + \begin{cases} \frac{(3k_i^2+3k_j^2-4)^2}{32k_j^2} & \text{if } 3k_i^2 + 3k_j^2 \geq 4 \\ 0 & \text{otherwise} \end{cases} - \begin{cases} \frac{(3k_i^2-4k_j)^2}{32k_j^2} & \text{if } 3k_i^2 \leq 4k_j \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The following technical claim can then be established.

▷ Claim 13. For any $k_i, k_j \in (0, 1]$, we have $f(k_i, k_j) \geq 0$.

Figure 2 shows a plot of $f(k_i, k_j)$, which demonstrates that it is non-negative for all $k_i, k_j \in [0, 1]$. One can prove analytically Claim 13, using standard arguments and an exhaustive case analysis.

So far, we have established that we can have a Nash equilibrium, with the values of k_i set by (5). Under such a partition, the pools without a large player have size $l = 4/3$, while pools with one large player have size $k_i + a_i = 4/3 - (2 - 3k_i)^2/12 \leq 4/3$. Since the optimal partition, at its best would split the players into pools of total stake equal to $h = 1$, the Price of Stability is at most 4/3. A matching lower bound is given by Example 9 and Claim 10.

This completes the proof of the theorem when all pools of non-atomic players have stake of exactly $l = 4/3$. If the total stake is such that this is not possible, we can construct a Nash equilibrium with pools of stake $l = 4/3 + \epsilon$, where ϵ can be made arbitrarily small, provided there is a sufficiently large mass of non-atomic players. The preceding argument applies almost identically to this case as well. This complicates some expressions, regarding the function $f(k_i, k_j)$, but it does not affect the core logic of the proof. ◀

An interesting consequence of the theorem's proof is that large players receive more reward at the Nash equilibrium than their proportional share. Specifically, their reward is $(a_i - h + k_i)/k_i$, while their proportional share is $a_i/(a_i + k_i)$. Given that $a_i = 1 - 3k_i^2/4$, we get the following.

▶ **Fact 1.** *In the Nash equilibria of Theorem 11, when $h = 1$, the reward of a large player in a pool of mass of non-atomic players k_i is more than its proportional sharing by*

$$\frac{k_i(3k_i - 2)^2}{4(2 - k_i)(3k_i + 2)} \geq 0,$$

The extra reward varies between 0 (when $a_i = 2/3$) and 0.05 (when $a_i \approx 1/4$).

3.3 The Price of Stability in the atomic model

In this subsection, we provide an analogous analysis for the atomic case, where we could have again large and small players, but not infinitesimally small, i.e., $N_s = \emptyset$, and therefore $N = N_a$. We believe it is valuable to understand the performance under the purely atomic model as well. This provides a more complete picture for these games and intends to answer the question of whether a low Price of Stability is maintained even without infinitesimally small players. When there are only atomic players, the discrete nature of the problem makes the Price of Stability go up, albeit not by a lot in the cases we consider below. This is consistent with the analogous flavor of results in the analysis of congestion games with atomic and non-atomic players [28].

We start with the following easy observation, showing that we have a worse lower bound when there is no mass of non-atomic players.

▷ Claim 14. The analysis of Example 7 yields that in the atomic model, $PoS \geq 2$.

Assumption on the stake distribution. The analysis for the atomic model quickly becomes much more technical and challenging for an arbitrary stake distribution, due to the expression for the Shapley formula. But following the same spirit as with the oceanic model, we have managed to analyze a class of games that follow a separation of N into large and small players. In particular, we will consider a 2-valued stake distribution, so that for every player $i \in N$, either $a_i = 1$ modeling a small player or $a_i = a > 1$, with $a < h$, modeling a large player. We leave as an open problem the analysis for the case where the large players can have different stakes.

Let n_b, n_s be the number of large and small players respectively, so that $n = n_s + n_b$. Our main result of this subsection is the following theorem, where we give a matching upper bound to Claim 14, as long as the number of small players is sufficiently large, so that again the Price of Stability is bounded by a small constant.

▶ **Theorem 15.** *For instances on $n = n_s + n_b$ players, with $n_s \geq (2h - 1)^2 + n_b \cdot (h + 1)$, it holds that $PoS \leq 2$.*

To prove this, we will construct an equilibrium, with a similar pattern as in the proof of Theorem 11, where each large player forms a pool with only small players. In the atomic model, such partitions are captured by the following definition.

▶ **Definition 16.** *For 2 integers k, l, with $k + a \geq h$ and $l \geq h + 1$, we say that a partition of N is a (k, l)-partition if it consists only of pools of the following form (without necessarily having pools from all the three types below)*

- *Pools with exactly one big player and k small players (Type A pools),*
- *Pools with l small players (Type B pools),*
- *Pools with $l-1$ small players (Type C pools).*

A (k,l)-partition that is also a Nash equilibrium is referred to as a (k,l)-equilibrium. The reason we may need both Type B and Type C pools is that the total number of agents assigned to these pools may not be divisible by l or $l-1$. Furthermore, if we find equilibria where k and l are upper bounded by a small multiple of h, this yields directly a bound on the Price of Stability.

3.3.1 Games with one large player

For the atomic model, it is very instructive (and essentially the core of the proof) to provide first the analysis when we have only one large player with stake a (we fix this to be player 1). All the other players have stake equal to 1. As we will see, this is already a non-trivial case.

We first provide some useful properties that help us understand the Shapley value scheme as well as the structure of its equilibria. The next lemma is the analog of Lemma 8 and is derived by applying the Shapley formula (3).

▶ **Lemma 17.** *Consider a winning pool S, containing the large player and k small players.*

- *If $k < h$, then the Shapley scheme provides to player 1 a payoff of $\phi_1(S) = \frac{k-h+a+1}{k+1}$. Each of the other players receives a payoff of $\frac{h-a}{k\cdot(k+1)}$.*
- *If $k \geq h$, then player 1 has a payoff of $\phi_1(S) = \frac{a}{k+1}$. Each of the other players receives a payoff of $\frac{k+1-a}{k\cdot(k+1)}$.*

The following is straightforward by applying (3).

▶ **Fact 2** (Pools of Type B or C). *Consider a winning pool S consisting of only small players, with $|S| = t$, for some $t \geq h$. Then for every $i \in S$, $\phi_i(S) = 1/t$.*

The next lemma justifies why in Definition 16, it suffices to consider pools of Type B and Type C, regarding the pools containing only small players, in the construction of equilibria.

▶ **Lemma 18.** *Suppose there exists a Nash equilibrium containing in its partition 2 pools S_1, S_2, that consist only of small players. Then, it must hold that*

$$||S_1| - |S_2|| \leq 1$$

Therefore, for games with a single large player, when $a < h$, the equilibria of the game must have the format of a (k,l)-partition, for appropriate values of k and l. Each of the pools should have total stake that exceeds the threshold h, so that they are winning pools, and this is why in Definition 16, we enforced that $k + a \geq h$ and $l - 1 \geq h$ ($l \geq h$ some times also suffices if there are no pools of Type C). The next step is to understand how to set the parameters k and l so that a (k,l)-partition is a Nash equilibrium. The following lemma describes a set of sufficient conditions, which are also necessary when the partition has pools of all types (A, B and C). For our purposes, we will stick below to the case that $k \leq h-1$.

▶ **Lemma 19** (Sufficient conditions). *Consider a (k,l)-partition with $k < h$. When $k \leq h-2$, the partition is a Nash equilibrium, if the following conditions hold:*

1. $\frac{k(k+1)}{h-a} \leq l \leq \frac{(k+1)(k+2)}{h-a}$
2. $l \geq \frac{a(k+1)}{k-h+a+1}$

When $k = h-1$, the only change is that the upper bound in the first condition needs to be replaced by: $l \leq \frac{(k+1)(k+2)}{h+1-a} = \frac{h(h+1)}{h+1-a}$.

Based on the above, we prove below that for large enough games, there always exist Nash equilibria (characterized by the values of k and l), which are relatively good compared to the optimal formation.

▶ **Theorem 20.** *For instances with a single large player and with $n \geq (2h-1)^2 + h$, PoS ≤ 2.*

Proof. We only provide an outline here and omit the proofs of Lemmas 21 and 22, that are stated below, which are the most technical parts of the entire proof. We show that there exists a (k,l)-equilibrium profile, with $l \leq 2h$ and $k + a \leq 2h$. This would mean that all the pools at the equilibrium have stake at most $2h$. Since the optimal formation has pools with stake at least h, the Price of Stability is at most 2.

To ensure the existence of such an equilibrium, we exploit the sufficient conditions of Lemma 19, and we distinguish 2 cases, based on whether we will search for $k \leq h-2$ or for $k = h-1$.

Case 1: $h \leq a^2 - 2a + 2$. In this case we will identify a value for k so that $k \leq h-2$. The first condition of Lemma 19 defines an interval Δ for the possible values of l, which contains at least two integers. This is true since $\frac{(k+1)(k+2)}{h-a} - \frac{k(k+1)}{h-a} = \frac{2(k+1)}{h-a} \geq 2$, for any feasible k, since we are only looking for values of k with $k + a \geq h$. Hence, if the interval Δ belongs entirely to the region for l defined by the second condition of Lemma 19, there exists an equilibrium where both $l-1$ and l can take an integer value belonging to Δ. In order for this to happen, it must hold that: $\frac{k(k+1)}{h-a} \geq \frac{a(k+1)}{k-h+a+1} \Leftrightarrow k^2 + k(a-h+1) - a(h-a) \geq 0$.

We can proceed with solving the inequality of degree two, with respect to k. It is easy to check that the discriminant is positive. Hence, by carrying out the calculations, we eventually get that the inequality holds for the integer values that satisfy

$$k \geq k^* = \frac{1}{2}(\sqrt{-3a^2 + h^2 + 2ah - 2h + 2a + 1} - a + h - 1). \tag{9}$$

The remainder of the proof for this case now is to show how to set k and l appropriately.

▶ **Lemma 21.** *For any pair of integer values of a and h for which $2 \leq a \leq h-1$, and $h \leq a^2 - 2a + 2$, there are integer values k, l such that: $k \geq k^*$, $h - a \leq k \leq h - 2$ and $h + 1 \leq l \leq 2h$.*

Case 2: $h > a^2 - 2a + 2$. Now we cannot guarantee that $k \leq h-2$ and we need to follow a different (and in fact simpler) approach. We have again an interval Δ for the values of l, but as we will be looking for $k = h-1$, we need to use the analogous conditions from Lemma 19. The remaining part for handling Case 2 is by the lemma below.

▶ **Lemma 22.** *For any pair of integer values of a and h for which it holds that $2 \leq a \leq h-1$, $h > a^2 - 2a + 2$, there are integer values k, l such that: $h - a \leq k \leq h - 1$ and $h + 1 \leq l \leq 2h$.*

Given Lemmas 21 and 22, we can now describe the equilibrium construction for any $n \geq l^2 + k + 1$ (for the integer values k, l that were determined in Lemmas 21 and 22), and thus for any $n \geq (2h-1)^2 + 2h$. Since the values for k and l that we found satisfy the equilibrium conditions, all that remains to show is that we can actually partition the n players into such a (k,l)-partition. To do this, we will have $k+1$ players, including the large player, form one Type A pool. For the remaining $n - k - 1 \geq (l-1)^2$ players, we first create as many Type C pools with exactly $l-1$ players as possible. And then, each of the remaining $(n-k-1) \mod (l-1)$ players is added to a different Type C pool, converting it to a Type B pool. This completes the proof of Theorem 20. ◀

3.3.2 Games with more large players

Building upon the results of the previous subsection, we can address the more general case, where there are more large players. Suppose we have n_b large players and n_s small players so that $n = n_b + n_s$. As before, we show that there is again a (k, l)-partition that is an equilibrium and so that each pool has stake at most $2h$. To prove this, we need again to identify sufficient equilibrium conditions. But for this we can still utilize Lemma 19 with one additional condition: that large players have no incentive to deviate to another Type A pool.

3.3.3 Lower bounds for the atomic model with enough small players

For the atomic model, Claim 14 shows that Example 7 provides a lower bound of 2. However the example does not satisfy the premises of Theorem 15, that we have sufficiently many small players of stake 1. Under this assumption the best lower bound we have established is 1.5, as stated below. Therefore, for the atomic model, it remains an open problem whether the Price of Stability is lower than 2 when there is a sufficiently high population of small players or whether Theorem 15 is tight.

▶ **Theorem 23.** *There exist instances where the number of small players satisfies the assumption of Theorem 15, and where PoS* ≥ 1.5.

Proof. In order to prove this lower bound of $\frac{3}{2}$ we will consider a simple instance, where there is only one large player with stake $a = 2$, the threshold is equal to $h = 4$, and there is a large amount of small players with stake 1. In this instance, since there is only one "big" player and many small players, the best equilibrium is necessarily in the form of a (k, l)-equilibrium, with the lowest possible value for l. From the conditions of Lemma 19, we have that in a (k, l)-equilibrium it must hold that $\frac{k(k+1)}{h-a} \leq l \leq \frac{(k+1)(k+2)}{h-a}$ and $l \geq \frac{a(k+1)}{k-h+a+1}$. In order to find the best equilibrium, it suffices to find the smallest value of k, for which the two above conditions overlap. This happens for $k = 2$ and the two conditions above become $3 \leq l \leq 6$ and $l \geq 6$. Hence, the only feasible value for l to have a (k, l)-equilibrium is $l = 6 = \frac{3h}{2}$. Given that the optimal formation consists of pools with total stake at least equal to h, this leads us to the lower bound of $\frac{3}{2}$ for the Price of Stability. ◀

4 Optimality of the proportional sharing scheme

We come now to compare the Shapley scheme with the simplest possible scheme that one could apply in this setting, namely proportional sharing.

Recall that for a pool C, we have denoted by $m(C)$ the total stake of the pool, including atomic and non-atomic stake. For atomic players, the proportional share of an agent i is the percentage of her contribution over $m(C)$, where C is the pool that she belongs to. For the non-atomic players, the reward per unit of stake is equal to the profit divided by the total stake of the pool. This is a quite standard and popular rule, that is being used widely in several off-chain and on-chain agreements.

▶ **Definition 24.** *For a player $i \in N_a$, with stake a_i belonging to a pool C, the reward she receives by the proportional reward sharing scheme is $p_i(a_i, C) = \frac{a_i}{m(C)} \cdot \rho(C)$. For non-atomic players, the reward per unit of stake is $\rho(C)/m(C)$.*

For this scheme, there is always an equilibrium that induces an optimal formation, and this holds for any stake distribution, without the premises of Theorem 11.

▶ **Theorem 25.** *In the oceanic model, the proportional sharing scheme has Price of Stability equal to 1. This holds independently of the volume of non-atomic players (and hence it holds for the purely atomic model as well).*

Proof. Consider an optimal partition of the players into pools, as defined in (2). If there are multiple optimal partitions, consider the one that is as balanced as possible, i.e., a lexicographically optimal (leximin) one, where the stake of the smallest pool is maximized, and subject to that, the stake of the second smallest pool is maximum, and so on. Let $C_1, \ldots, C_t$ be the pools of this partition. It holds that $m(C_j) \geq h$ for any $j \in [t]$.

We note first that all the non-atomic players must belong to the pool (or pools) of minimum stake. If not, we could transfer some stake to the smallest pools and arrive at a lexicographically better one. Hence, for all non-atomic players, the reward per unit of stake is $\frac{1}{\min_j m(C_j)}$. This means that they have no incentive to switch to any other pool, where the reward per unit of stake is lower. Let us consider now the atomic players. If the partition is not an equilibrium, then there exists a player $i \in N_a$, belonging to some pool C_k, who becomes better off by moving to a pool C_r. But then this means that $\frac{s_i}{m(C_k)} < \frac{s_i}{s_i + m(C_r)}$. This implies that $m(C_k) > s_i + m(C_r)$, i.e., that $m(C_k \setminus \{i\}) > m(C_r)$. But then i is included in a pool C_k in the optimal partition that we considered, where even without i, C_k has more stake than C_r. This means that the partition we started with was not a lexicographically optimal partition, a contradiction. ◀

5 Sybil resilience

So far, it may appear that the proportional scheme is a dominating solution within the space of candidate reward schemes. In this section, we highlight one drawback of proportional sharing, which provides an advantage of Shapley, related to the resilience to Sybil strategies. By a Sybil strategy we refer to the act of an atomic player to split her stake into smaller portions so as to participate in different pools, disguised as a set of different players (which can be feasible, given the anonymity in blockchain environments).

5.1 Sybil resilience of equilibria

There are different ways to define Sybil attacks in the literature, depending on the model at hand and on the allowed or feasible behaviors from the players. For our model, we consider Sybil attacks with respect to some given configuration (which may have formed from previous actions that the players took). We also consider that only the atomic players are eligible to perform a Sybil attack.

▶ **Definition 26.** *Fix a strategy profile inducing a partition* Π*, of the players into pools. Given* Π*, a Sybil strategy for a player* $i \in N_a$ *with stake* a_i *is a strategy where the player splits her stake into* $s_{i_1}, \ldots, s_{i_t}$ *for some* t*, with* $\sum_{j=1}^{t} s_{i_j} = a_i$ *and chooses* t *pools to join among the ones formed by* Π*, contributing stake* s_{i_j} *to pool* j*, for* $j \in [t]$*. The payoff under such a strategy for player* i *is the sum of the payoffs from all the* t *pools.*

Based on the above definition of a Sybil attack, below we define the notion of Sybil-proofness (alternatively Sybil-resiliency).

▶ **Definition 27.** *Fix a partition* Π*. We say that a reward sharing scheme is Sybil-proof w.r.t. the partition* Π *if no player* i *can become strictly better off by switching to a Sybil strategy.*

Ideally, we would like to have reward schemes where no player has an incentive for Sybil attacks regardless of the configuration the game might be at. This is too much to ask for, but we start here with a positive result, that concerns Sybil resilience w.r.t. the equilibrium formations of Section 3. In particular, for the Shapley scheme, we show that even after enlarging the strategy space with Sybil strategies, the equilibrium partitions described in Section 3, continue to be at equilibrium.

▶ **Theorem 28.** *In the oceanic model, the Shapley value reward sharing scheme is Sybil-proof w.r.t. the equilibria identified in Theorem 11, and more generally w.r.t. any equilibrium as described by the conditions of Lemma 12.*

Proof. Consider an equilibrium formation satisfying the conditions of Lemma 12, with d pools $C_1, \ldots, C_d$. Each pool contains either an atomic player j with stake a_j with a mass of k_j small players (such that $a_j + k_j \geq h$), or a mass of $l > h$ small players.

Suppose that an atomic player i with stake a_i decides to split her stake into m different portions, $s_{i_1}, s_{i_2}, ..., s_{i_m}$ and commit s_{i_t} to pool C_t, where $\sum_{t\in[m]} s_{i_t} = a_i$. We distinguish the following cases.

Case 1. C_t is the pool that i belonged to initially. Then by Lemma 8, her reward is $\frac{k_i - h + s_{i_t}}{k_i}$.

Case 2. C_t is a pool, with a mass of $l \geq h$ small players. By contributing s_{i_t} to such a pool, Lemma 8 shows that player i receives a reward equal to $\frac{s_{i_t}}{l}$. This is at most $\frac{s_{i_t}}{k_i}$, due to the fact that $k_i < h = 1 \leq l$.

Case 3. C_t is a pool with another atomic player j with stake a_j and a mass of small players k_j. For ease of notation in this case, let $s_{i_t} = b$ and without loss of generality, let $h = 1$. We will establish that the reward player i gets is at most $\frac{b}{k_i}$. After the Sybil attack of player i, C_t contains player j with stake a_j, player i with stake b, and a mass of size k_j of small players. We need to discriminate between some subcases here, since the formula of the reward agent i receives varies. The reward agent i receives, by Lemma 8 is $\frac{(1-a_j)^2 - \max(0,1-b-a_j)^2 + \max(0,b-1+k_j)^2}{2{k_j}^2}$.

Since we are in an equilibrium formation, from the conditions in Lemma 12 it holds that $k_i = \sqrt{l \cdot (1-a_i)}$ and $k_j = \sqrt{l \cdot (1-a_j)}$. Also, since $k_i, k_j \leq h$, it holds that $a_i, a_j \in [1-\frac{1}{l}, 1)$. Note also that the function $\frac{b}{k_i} = \frac{b}{\sqrt{l\cdot(1-a_i)}}$, is increasing with respect to a_i. At the same time, for the reward function of player i within pool C_t, we have the following property, which can be easily verified:

▷ Claim 29. The function $\frac{(1-a_j)^2 - \max(0,1-b-a_j)^2 + \max(0,b-1+k_j)^2}{2{k_j}^2}$, is decreasing with respect to a_j, for $a_j \in [0, 1)$.

So in order to show that $\frac{b}{k_i} \geq \frac{(1-a_j)^2 - \max(0,1-b-a_j)^2 + \max(0,b-1+k_j)^2}{2{k_j}^2}$, it suffices to show that this holds for the lowest values that a_i and a_j can take, which is $1 - \frac{1}{l}$.

When $a_i = a_j = 1 - \frac{1}{l}$, it holds that $k_i = k_j = 1$. Given the form of the right hand side, we consider first the subcase that $1 - b - a_j < 0 \Rightarrow b > 1 - a_j = 1 - (1 - \frac{1}{l}) = \frac{1}{l}$. Also note that $b - 1 + k_j \geq 0$, since $k_j = 1$. In this scenario, the Sybil attacker gets reward from C_t equal to $\frac{(1-a_j)^2 + (b-1+k_j)^2}{2{k_j}^2}$. The desired inequality is equivalent to showing that $2b \geq (1 - (1 - \frac{1}{l}))^2 + b^2 = \frac{1}{l^2} + b^2$, which holds due to the fact that $b > \frac{1}{l} \Rightarrow b > \frac{1}{l^2}$, for any $l \geq 1$ and also $b \geq b^2$, since $b \leq a_i < 1$.

Next, we move to the subcase where $1 - b - a_j \geq 0 \Rightarrow b \leq 1 - a_j = \frac{1}{l}$. Again, $k_i = k_j = 1$, so the desired inequality becomes $2b \geq (1 - (1 - \frac{1}{l}))^2 - (\frac{1}{l} - b)^2 + b^2 = \frac{2b}{l} \Rightarrow 2b \geq \frac{2b}{l}$, which holds for any $l \geq 1$.

Putting everything together, the initial reward player i receives is $\frac{k_i-h+a_i}{k_i}$. Let C_1 be the pool where i belonged to initially, before the Sybil attack. We can then rewrite this as follows

$$\frac{k_i-h+a_i}{k_i} = \frac{k_i-h+(\sum_{t=1}^{m} s_{i_t})}{k_i} = \frac{k_i-h+s_{i_1}}{k_i} + \sum_{t=2}^{m} \frac{s_{i_t}}{k_i}$$

But now we can see that each term in the summation is at least as big as the reward received by each pool C_t analyzed in Cases 2 and 3. Also the leftmost term is at least as good as the reward received at Case 1. Hence player i earns more rewards by staking all her stake in her original pool. ◀

Next, we analyze the proportional scheme. In contrast to the Shapley value, and to our surprise, the proportional scheme has a disadvantage here. We show that the optimal equilibrium formation identified in Theorem 25 is not always Sybil-proof.

▶ **Theorem 30.** *There exist instances where the proportional reward sharing scheme is not Sybil-proof w.r.t. the optimal equilibrium of Theorem 25.*

Proof. We show first a simple example when there is no ocean of small players. Consider an instance with 3 large players, who all have stake $a = h/2 + 1$, and with $h/2 - 1$ additional players, with stake equal to 1. Obviously $OPT(G) = 2$, where the optimal formation is to have 2 pools C_1, C_2, with C_1 containing two large players together and C_2 containing the remaining players. This is an equilibrium, as follows by Theorem 25, and the payoff for each player in C_1 is $1/2$. Suppose now that one of the large players in C_1 decides to allocate 1 unit of stake to C_2 and keep the remaining stake in C_1. Her total payoff from this Sybil attack will be equal to $\frac{h/2}{h+1} + \frac{1}{h+1} > 1/2$.

Even when we have a large mass of non-atomic players, one can still have analogous constructions. The reason is that when the total stake of the game is not a multiple of h, the scheme then becomes vulnerable to Sybil attacks. To illustrate this, consider an instance with n atomic players and a mass of small players, such that the total stake is not a multiple of h. This means that in the optimal equilibrium formation of Theorem 25, the pool of maximum stake, say C_1 will have total stake $\ell_1 > h$. Suppose that C_2 is the pool of minimum stake, equal to ℓ_2, with $h \leq \ell_2 \leq \ell_1$. Let i be an atomic player in C_1 with stake a_i. There exists such a player, since by the proof of Theorem 25, we know that when $\ell_1 > \ell_2$, then C_1 has to contain atomic players (if $\ell_1 = \ell_2$, then we pick as C_1 any pool that contains an atomic player). We argue that agent i can gain more total reward by splitting her stake into two portions, specifically $a_i - x$ which she will commit to her initial pool C_1 and x (s.t. $\ell_1 - x \geq h$, which is feasible if x is small enough), which she will commit to pool C_2. By this splitting, agent i will receive a total reward of $\frac{a_i-x}{\ell_1-x} + \frac{x}{\ell_2+x}$, based on the proportional reward sharing scheme. Suppose now that $\ell_1 = \ell_2 + \delta$, for some $\delta \geq 0$. By simple calculations, we can see that the reward of i under this Sybil attack is greater than her initial reward of $\frac{a_i}{\ell_1}$, as long as

$$x \leq \frac{\ell_2(a_i+\delta)+\delta^2}{2\ell_2+2\delta-a_i}.$$ ◀

The next definition captures the stronger property that one could envision, i.e., Sybil resilience no matter what is the initial configuration at hand (equilibrium or not).

▶ **Definition 31.** *A reward sharing scheme is Sybil-proof if it is Sybil-proof w.r.t. any partition* Π.

This is obviously a quite demanding requirement, especially since some partitions may be very unlikely to form in practice. Theorem 30 shows already that proportional sharing does not satisfy it. For the Shapley scheme we can also construct non-equilibrium profiles where Definition 31 fails. To see this consider a partition (C_1, C_2), where C_1 contains 2 atomic players with stake $a > h/2$ and C_2 contains non-atomic players of mass $l = h$. Then it is easy to verify, using Lemma 8, that a large player from C_1 can keep a stake of $h/2$ in the first pool and contribute the remaining to C_2 resulting in a higher payoff for her.

Hence, when there is no significant cost of carrying out a Sybil attack, we cannot expect to avoid such strategies from any configuration Π, except with trivial schemes (e.g., zero payments). In fact such impossibilities have already been pointed out under a quite different model in [21]. We exhibit below that for our model, this holds beyond the Shapley and the proportional scheme, for a wide class of reward schemes.

▶ **Observation 32.** *Consider a reward scheme where the payment $p_i(a_i, S)$ of a player with stake a_i in a pool S, depends on a_i and on the stake distribution of S_{-i} but in an anonymous way. Suppose further that the payment is concave w.r.t. a_i, even in a subset of the entire range of $(0, h)$ (with at least one value of a_i where concavity holds with strict inequality). Then the scheme is not Sybil-proof.*

Proof. Consider a partition where player i ends up in a pool S, and where two other pools have formed S_1, S_2, such that $m(S_1) = m(S_2) = m(S \setminus \{i\})$ and the stake distribution of S_1 and S_2 is the same with the distribution of $S \setminus \{i\}$ (meaning that the composition in terms of the stakes of the atomic players and the volume of the non-atomic players is the same). Then suppose that player i splits her stake a_i into 2 portions of $a_i/2$ and commits it to S_1 and S_2. Then by the assumption of the theorem, regarding strict concavity, we would have that $p_i(a_i, S) < p_i(a_i/2, S_1) + p_i(a_i/2, S_2)$. ◀

5.2 Computation of Sybil strategies

We conclude this section by commenting on the computational problem of actually finding a good Sybil strategy. For this, fix a player i with stake a_i, and an arbitrary strategy profile of the other players $\mathbf{x}_{-i}$. Imagine that player i enters the game at a time where the strategies of the other players induce a partition $\Pi_{-i} = (C_1, \ldots, C_m)$, where each C_j is a winning pool (we ignore non-winning pools, if there are any, under the current partition). If player i wishes to maximize her utility, the problem that she has to solve is to compute how to split her stake and commit it to different pools in order to receive the maximum possible amount of reward. More formally, let $s_{i_1}, s_{i_2}, ..., s_{i_m}$, be the variables representing the portions of stake of a_i, that agent i will assign to each of the m pools. Then player i faces the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{m} p_i(s_{i_j}, C_j \cup \{i\}) \\ \text{subject to} \quad & \sum_{j=1}^{m} s_{i_j} \leq a_i \\ & s_{i_j} \geq 0 \quad \forall j \in [m] \end{aligned}$$

The constraints above are linear but this is not always an easy problem to solve, and its complexity is dependent on the reward scheme, since this affects the form of the objective function. For simpler schemes, such as the proportional one, we can have a polynomial

algorithm. In particular, under the proportional scheme, the objective function is a sum of concave functions (and hence a concave function itself), equal to $\sum_{j=1}^{m} \frac{s_{i_j}}{m(C_j)+s_{i_j}}$. This means that the optimization problem is equivalent to minimizing a convex function subject to linear constraints, which can be done efficiently. In fact, for the proportional scheme, we can have an even simpler, greedy algorithm, based on a water-filling argument. Hence, we can conclude with the following.

▶ **Theorem 33.** *Under the proportional scheme, the optimal solution of the above problem, can be computed in polynomial time.*

Proportional vs Shapley w.r.t. complexity of Sybil attacks. The above algorithm shows that for the proportional scheme, it is relatively easy to find a profitable Sybil attack (whenever one exists). On the other hand, this is not known to be applicable for the Shapley scheme. For the restriction to integer stakes, solving the aforementioned optimization problem is PP-hard [27]. Without integrality constraints, we are not aware of any efficient algorithm. Even if we have access to an approximation algorithm for the Shapley value of a pool, it does not seem obvious how to find an approximately optimal Sybil strategy. Hence the overall computational difficulty of the Shapley value could serve as a deterrent for potential manipulators.

6 Alternative variations of proportional schemes

We conclude our study with some further ideas that have been proposed in the literature. In [12], some alternative schemes are considered, based on variations of proportional sharing. This involves functions that are superadditive in a player's stake. For illustration, we examine the proportional-to-squares reward sharing scheme. Qualitatively, similar results hold if we use other functions of the stake in the form a_i^β with $\beta > 1$.

▶ **Definition 34.** *For a player i, with stake a_i belonging to a pool C, the reward by the proportional-to-squares reward sharing scheme is* $p_i(a_i, C) = \frac{a_i^2}{\sum_{j \in C} a_j^2} \cdot \rho(C)$.

On the opposite direction, we could have a subadditive function of each player's stake.

▶ **Definition 35.** *For a player i, with stake a_i belonging to a pool C, the reward she receives by the proportional-to-square-roots scheme is* $p_i(a_i, C) = \frac{\sqrt{a_i}}{\sum_{j \in C} \sqrt{a_j}} \cdot \rho(C)$.

We note that we have presented the definition above only for the purely atomic model, as taking the square or the square root of infinitesimally small quantities leads to uninteresting or trivialized payment rules in the limit. E.g., for a pool with one atomic player and a mass of non-atomic players, the proportional to squares rule would allocate the entire reward to the atomic player.

Even though such schemes can have their merits, we argue here that in our context they also have their drawbacks related to the metrics of the previous sections. We first exhibit that the proportional-to-squares scheme can lead to very bad equilibria, attaining very high Price of Stability.

▶ **Theorem 36.** *For any h, there exist instances where the proportional-to-squares scheme has Price of Stability at least $\frac{h-1}{2} = \Omega(h)$, and thus unbounded.*

When it comes to the proportional-to-square-roots scheme, the picture is quite different. We can show that the Price of Stability is comparable to the Shapley scheme.

▶ **Theorem 37.** *Consider instances with n players ($n = n_s + n_b$), so that $n_s \geq h^2 + n_b(\lceil h - \sqrt{a} + 1 \rceil)$. Then, for the proportional-to-square-roots reward sharing scheme, we have that PoS = 2.*

However, the proportional-to-square-roots scheme is not Sybil-proof w.r.t. its equilibria, in contrast to the Shapley value.

▶ **Theorem 38.** *There exist instances where the proportional-to-square-roots scheme is not Sybil-proof w.r.t. (k, l)-equilibria, as given in Defintion 16.*

Hence we can conclude that for the family of all considered variants of proportional sharing, we either have vulnerability to Sybil attacks or high Price of Stability.

7 Conclusions

We have studied a model of pool formation over blockchains with large, influential players and players with negligible stake. The conclusions regarding the schemes we considered are as follows.

1. The Shapley scheme has constant Price of Stability bounded by 4/3 in the oceanic model and by 2 in the atomic model. It is also Sybil-proof w.r.t. the constructed equilibria. Furthermore it is not always easy to find the best possible Sybil strategy.
2. The proportional scheme always has an optimal equilibrium (PoS = 1). But it is not Sybil-proof w.r.t. its equilibria. Moreover, Sybil attacks can be computed relatively easily.
3. The proportional to squares scheme has unbounded Price of Stability.
4. The proportional to square roots scheme has constant Price of Stability bounded by 2. But it is not Sybil-proof w.r.t. its equilibria.

We view this work as a promising initial step towards demonstrating the applicability of cooperative game theory concepts, such as the Shapley scheme, in the context of reward sharing for blockchain protocols. While not exhaustive, our findings provide a foundation for further exploration and development in this area.

7.1 Future work

There are several interesting avenues for future research. An immediate direction is to consider a different function $\rho(S)$ for the pool rewards so as to capture other scenarios or protocols. Our focus for $\rho(\cdot)$ in this paper was a threshold function inspired of validator rewards in Ethereum. However, other schemes are possible for instance, we can consider a model where $\rho(S)$ is a linear (up to a threshold) function of $\sum_{i \in S} a_i$, or other variants that provide additional reward to pools with large pool operators so that some "skin in the game" is present in pool formation, cf. [7]. With respect to this latter property, it would be also interesting to identify skin in the game as a separate or alternative objective (i.e., separate from welfare and the number of independent pools involved at equilibrium). For instance, having skin in the game is reflected in design choices of pool formation contracts like Rocketpool to impose a minimum threshold of 8 ETH to the operator for each pool spun by the system. It is easy to capture such considerations in the oceanic model by observing that skin in the game is not provided by non-atomic players and hence a pool configuration can

only derive skin in the game from the atomic players. Comparing equilibria and the Price of Stability under this objective would be an interesting parallel to the results presented herein, while considering both objectives simultaneously (equilibria exhibiting high decentralization and high skin in the game at the same time) points to an interesting Pareto optimization question.

Another direction is to consider further concepts from game theory, such as the Banzhaf index [11]. This has played a prominent role in weighted voting games, and it would be interesting to explore if it would lead to a performance comparable to the Shapley scheme. Finally, it is also important to investigate the dynamics for the iterated version of pool formation games and the speed of convergence to good equilibria.

References

1 Nick Arnosti and S. Matthew Weinberg. Bitcoin: A natural oligopoly. *Management Science*, 68(7):4755–4771, 2022. doi:10.1287/mnsc.2021.4095.

2 S. F. Assmann, D. S. Johnson, D. J. Kleitman, and J. Y. Leung. On a dual version of the one-dimensional bin packing problem. *Journal of Algorithms*, 5:502–525, 1984. doi:10.1016/0196-6774(84)90004-X.

3 L. M. Ausubel and P. Milgrom. The lovely but lonely vickrey auction. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*, pages 18–40. MIT Press, 2006.

4 Sarah Azouvi and Alexander Hicks. Decentralisation conscious players and system reliability. In *26th International Conference on Financial Cryptography and Data Security, FC 2022*, pages 426–443. Springer, 2022. doi:10.1007/978-3-031-18283-9_21.

5 Yoram Bachrach, Evangelos Markakis, Ezra Resnick, Ariel D. Procaccia, Jeffrey S. Rosenschein, and Amin Saberi. Approximating power indices: theoretical and empirical analysis. *Auton. Agents Multi Agent Syst.*, 20(2):105–122, 2010. doi:10.1007/s10458-009-9078-9.

6 Maryam Bahrani, Pranav Garimidi, and Tim Roughgarden. Centralization in block building and proposer-builder separation, 2024. arXiv:2401.12120. doi:10.48550/arXiv.2401.12120.

7 Lars Brünjes, Aggelos Kiayias, Elias Koutsoupias, and Aikaterini-Panagiota Stouka. Reward sharing schemes for stake pools. In *2020 IEEE european symposium on security and privacy (EuroS&p)*, pages 256–275. IEEE, 2020. doi:10.1109/EuroSP48549.2020.00024.

8 Vitalik Buterin. A next-generation smart contract and decentralized application platform, 2013. URL: https://github.com/ethereum/wiki/wiki/White-Paper.

9 Vitalik Buterin, Daniël Reijsbergen, Stefanos Leonardos, and Georgios Piliouras. Incentives in ethereum's hybrid casper protocol. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2019, Seoul, Korea (South), May 14-17, 2019*, pages 236–244. IEEE, 2019. doi:10.1109/BLOC.2019.8751241.

10 Burak Can, Jens Leth Hougaard, and Mohsen Pourpouneh. On reward sharing in blockchain mining pools. *Games Econ. Behav.*, 136:274–298, 2022. doi:10.1016/j.geb.2022.10.002.

11 G. Chalkiadakis and M. Woolridge. Weighted voting games. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia, editors, *Handbook of Computational Social Choice*, pages 377–396. Cambridge University Press, 2016. doi:10.1017/CBO9781107446984.017.

12 Xi Chen, Christos Papadimitriou, and Tim Roughgarden. An axiomatic approach to block rewards. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 124–131, 2019. doi:10.1145/3318041.3355470.

13 Zhihuai Chen, Xiaoming Sun, Xiaohan Shan, and Jialin Zhang. Decentralized mining pool games in blockchain. In *IEEE International Conference on Knowledge Graph, ICKG 2020*, pages 426–432, 2020. doi:10.1109/ICBK50248.2020.00067.

14 Xiaotie Deng and Christos Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994. doi:10.1287/moor.19.2.257.

15 John R. Douceur. The sybil attack. In *Peer-to-Peer Systems, First International Workshop, IPTPS 2002*, pages 251–260, 2002. doi:10.1007/3-540-45748-8_24.

16 Giulia Fanti, Leonid Kogan, Sewoong Oh, Kathleen Ruan, Pramod Viswanath, and Gerui Wang. Compounding of wealth in proof-of-stake cryptocurrencies. In *23rd International Conference on Financial Cryptography and Data Security, FC 2019*, pages 42–61. Springer, 2019. doi:10.1007/978-3-030-32101-7_3.

17 Shaheen S Fatima, Michael Wooldridge, and Nicholas R Jennings. A linear approximation method for the Shapley value. *Artificial Intelligence*, 172(14):1673–1699, 2008. doi:10.1016/j.artint.2008.05.003.

18 Donald Bruce Gillies. *Some theorems on n-person games*. Princeton University, 1953.

19 Aggelos Kiayias, Elias Koutsoupias, Evangelos Markakis, and Panagiotis Tsamopoulos. Pool formation in oceanic games: Shapley value and proportional sharing, 2025. doi:10.48550/arXiv.2505.04422.

20 Aggelos Kiayias, Elias Koutsoupias, Francisco Marmolejo-Cossio, and Aikaterini-Panagiota Stouka. Balancing participation and decentralization in proof-of-stake cryptocurrencies. In *17th International Conference on Algorithmic Game Theory, SAGT 2024*, pages 333–350, 2024. doi:10.1007/978-3-031-71033-9_19.

21 Yujin Kwon, Jian Liu, Minjeong Kim, Dawn Song, and Yongdae Kim. Impossibility of full decentralization in permissionless blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 110–123, 2019. doi:10.1145/3318041.3355463.

22 Nikos Leonardos, Stefanos Leonardos, and Georgios Piliouras. Oceanic games: Centralization risks and incentives in blockchain mining. In *1st International Conference on Mathematical Research for Blockchain Economy, MARBLE 2019*, Springer Proceedings in Business and Economics, pages 183–199. Springer, 2019. doi:10.1007/978-3-030-37110-4_13.

23 Yoad Lewenberg, Yoram Bachrach, Yonatan Sompolinsky, Aviv Zohar, and Jeffrey S. Rosenschein. Bitcoin mining pools: A cooperative game theoretic analysis. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015*, pages 919–927. ACM, 2015. URL: http://dl.acm.org/citation.cfm?id=2773270.

24 J. W. Milnor and L. S. Shapley. Values of large games ii: Oceanic games. *Mathematics of Operations Research*, 3(4):290–307, 1978. doi:10.1287/moor.3.4.290.

25 Herve Moulin and Scott Shenker. Strategyproof sharing of submodular costs: Budget balance vs efficiency. *Economic Theory*, 18:511–533, 2001.

26 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: https://bitcoin.org/bitcoin.pdf.

27 Anja Rey and Joerg Rothe. False-name manipulation in weighted voting games is hard for probabilistic polynomial time. *Journal of Artificial Intelligence Research*, 50:573–601, 2014. doi:10.1613/jair.4293.

28 Tim Roughgarden. *Twenty Lectures on Algorithmic Game Theory*. Cambridge University Press, 2016.

29 Benedek Rozemberczki, Lauren Watson, Péter Bayer, Hao-Tsung Yang, Oliver Kiss, Sebastian Nilsson, and Rik Sarkar. The Shapley value in machine learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022*, pages 5572–5579, 2022. doi:10.24963/ijcai.2022/778.

30 Lloyd Shapley. A value for n-person games. In H. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 2, pages 307–317. Princeton University Press, 1953.

Single-Token vs Two-Token Blockchain Tokenomics

Aggelos Kiayias
University of Edinburgh, UK
IOG, Edinburgh, UK

Philip Lazos
London, UK

Paolo Penna
IOG, Zurich, Switzerland

Abstract

We study long-term equilibria that arise in the token monetary policy, or *tokenomics*, design of proof-of-stake (PoS) blockchain systems that engage utility maximizing *users* and *validators*. Validators are system maintainers who get rewarded with tokens for performing the work necessary for the system to function properly, while users compete and pay with such tokens for getting a desired portion of the system service.

We study how the system service provision and suitable rewards schemes together can lead to equilibria with the following desirable characteristics (1) viability: the system keeps parties engaged, (2) decentralization and skin-in-the-game: multiple sufficiently invested validators are participating, (3) stability: the price path of the underlying token used to transact with the system does not change widely over time, and (4) feasibility: the mechanism is easy to implement as a smart contract, e.g., it does not require a fiat reserve on-chain to perform token *buybacks* or to perform bookkeeping of exponentially growing token holdings.

Our analysis enables us to put forward a novel generic mechanism for blockchain monetary policy that we call *quantitative rewarding* (QR). We investigate how to implement QR in single-token and two-token proof of stake (PoS) blockchain systems. The latter are systems that utilize one token for the users to pay the transaction fees and a different token for the validators to participate in the PoS protocol and get rewarded. Our approach demonstrates a concrete advantage of the two-token setting in terms of the ability of the QR mechanism to be realized effectively and provide good equilibria. Our analysis also reveals an inherent limitation of the single token setting in terms of implementing an effective blockchain monetary policy – a distinction that is, to the best of our knowledge, highlighted for the first time.

2012 ACM Subject Classification Theory of computation → Solution concepts in game theory

Keywords and phrases Blockchain, tokenomics, buyback, equilibria, price path, stable price, discounted game, dual-token, proof-of-stake, validator

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.22

Related Version *Full Version*: `https://arxiv.org/abs/2403.15429` [15]

1 Introduction

Blockchains create value by offering services in a fully decentralized manner, wherein *users* pay fees to access these services, while the functioning and security of the system is guaranteed by a set of nodes or *validators* who receive rewards for performing the necessary computations required by the protocol. These payments are issued in the system's native *token* and the mechanism that mints and distributes these tokens to the relevant participants determines the "tokenomics policy" of the blockchain. Designing such policies with good properties is pivotal to ensuring the success of blockchain systems in the long term.

The token's value or *price*, denominated in standard (fiat) currency, crucially determines the actual costs for users and the compensation for validators. An upward or downward fluctuation in the token's price can make the system less attractive for either type of party.

© Aggelos Kiayias, Philip Lazos, and Paolo Penna;
licensed under Creative Commons License CC-BY 4.0

While the system cannot directly control its token price in the market, it can implement various monetary policies (such as increasing token minting, burning transaction fees, adjusting the level of transaction fees, change the validator level of rewards, and others) to achieve a long-term equilibrium with the desired price without compromising the system's viability and decentralization.

This motivates the study of tokenomics design achieving the following important desiderata:

1. **Viability**. Preconditioned on a positive system value, the system keeps all involved parties actively engaged: validators to guarantee the protocol being live and securely running, and the users to guarantee enough fees are collected at all time.
2. **Decentralization and "Skin-in-the-game."** A set of validators each equally engaged in the protocol – maximal decentralization [20], receiving adequate rewards to offset their costs, while also having significant stake in the system operation (e.g. in the case of proof-of-stake (PoS) systems, validators "staking" high enough amount of tokens).
3. **Stability**. Reasonably stable token prices, meaning that the price of the token required to issue transactions does not significantly change over time.
4. **Feasibility**. The tokenomics policy should be feasible to implement on-chain algorithmically. In particular, this means that the policy avoids using features that are incompatible or difficult to implement in the form of smart contracts, integrate into the blockchain accounting model or they are antithetical to blockchain decentralization. For example, the policy should minimize the use of off-chain entities that need to intervene actively to adjust policy (e.g., a central bank type of actor), or the use of techniques such as buybacks which would require a fiat reserve to facilitate. Furthermore, all the numerical quantities that are accounted in the ledger (e.g., the number of tokens held by a validator) should *not* grow exponentially over time.

The conditions under which there is a tokenomics policy achieving *all* these requirements is a fundamental question which we address in this work.

1.1 Main contributions

We explore tokenomics policies in the PoS setting, i.e., the setting where validators have to acquire tokens and stake them in order to receive rewards and perform the necessary system maintenance to further the system's operation. We put forth a model where a set of m users and n validators engage with the system in discrete time steps. At each time step, users and validators buy or sell tokens in order to accommodate their objectives that include issuing transactions and staking tokens to provide the service.

Modeling framework. The model takes as given two exogenous parameters. The discount factor δ, a conventional element in economic models, reflects players' preference for immediate utility over future gains, and is tied to the risk-free rate. The value of the service $S^{(t)}$ captures the usefulness of the system for users in each round t and may fluctuate unpredictably due to external shocks (Remark 3). The system can set rewards $R^{(t)}$ to be distributed in each round t to the validators according to their current stake at round t; each validator also faces a fixed cost $v > 0$ associated with maintaining the required secure node configuration. As our analysis shows, the monetary policy, $R^{(t)}$, influences the equilibrium and the token price. One of our key findings (see below) is that suitably defined monetary policies can "absorb" shocks and avoid undesirable price fluctuations.

The equilibria in our model come from players who aim to maximize their own discounted utility, thus choosing strategies with time-dependent parameters, such as how many tokens to sell on the market or retain for future rounds (users need to spend tokens to access the

system, while validators need to stake tokens to receive rewards). These equilibria, in turn, govern the key endogenous variables of the system, namely the staking levels of the validators, the holdings of user tokens, the fees paid to the system, the demand and supply of tokens in the spot market and the resulting trajectory or price path of the token (the market-clearing price in equilibrium as in [11]).

Equilibria analysis and quantitative rewarding. We give both boundary conditions and actual policies (mechanisms) that implement equilibria with all desired features: viability, decentralization, stability, and feasibility. Specifically, our analysis leads us to put forward a novel generic mechanism for blockchain "monetary policy":

> **Quantitative Rewarding (QR).** The amount of rewards the system gives to the validators is suitably adjusted at each step to anticipate changes in the value of the service.

This QR concept arises from our equilibrium analysis, focusing on how market demand and supply respond to changes in service value $S^{(t)}$ and rewards $R^{(t)}$. In all equilibria implementing certain desired price paths, the following conditions hold at every round (cf. Theorem 9):

- Users spend an amount of tokens whose value is a constant fraction of the current service value $S^{(t)}$.
- Validators stake an amount of tokens that is a constant fraction of current rewards $R^{(t)}$.

As these conditions and equilibria do not rule out undesirable mechanisms such as buybacks, our goal is to use QR to induce equilibria where token demand and supply match–so no buybacks are needed. Observe that if, at any point in time, the value of the service decreases, users will buy fewer tokens, thus *reducing demand* and threatening price stability. Instead of relying on buybacks, our QR mechanism, perhaps somewhat counterintuitively, *increases the rewards* allocated to validators, who react by staking more tokens, which must be acquired from the market. This action effectively restores the demand for tokens to align with the supply. The reverse effect occurs when the value of the service increases.

We show that implementing QR in single-token systems to avoid buybacks is highly non-trivial, as it requires anticipating future shocks *far* in advance. In contrast, for two-token systems, QR can be implemented using only a one-step lookahead of the service value. This conclusion stems from equilibrium analysis, but can be intuitively explained as follows: In single-token systems, validators must retain some tokens to stake and sell only a portion to the market to satisfy demand. In two-token systems, one token is used for staking, while the other – distributed as rewards in our mechanism – is used by users. Validators can sell all of these reward tokens to the market, making it easier to align supply with demand. Consequently, any equilibrium where rewards match the next round's service value (current users' demand for new tokens) avoids buybacks.

Decoupling users from validators via distinct tokens enables the system to absorb service value fluctuations more effectively, while keeping the user token's price stable and preserving other desiderata. At equilibrium, the price of the stake token rises gradually, and validators stake a fixed amount, trading only the other token, which they receive as a reward. In contrast, a single-token setup may trigger a self-reinforcing loop of increasing validator rewards, even after service levels stabilize, underscoring a fundamental distinction between the single-token and two-token configurations.

Implementing QR and its challenges. In Section 3.3, we investigate how to implement QR via a mechanism which adjusts the rewards at every round based on the current rewards and the next round service value using a single token for accounting. An important challenge that arises in this setting is that the rewards formula is subject to a recursive condition (see the lower bound in Theorem 13 and Corollary 14) which makes rewards subject to a "feedback loop" effect that can lead to rewards explosion.

As a result, while maintaining certain (minimal) rewards to ensure decentralization and security, it is possible that the system can trigger an uncontrolled growth in the amount of rewards distributed and in the staked amounts of tokens (see Section 3.3.1 for conditions that trigger such explosion). In Section 3.4 we describe in detail the implementation of our mechanism for the case of stable prices. This implementation includes the strategies for users and validators, showing that all information needed is service value at the next round (despite their simplicity, these strategies still constitute an equilibrium in the underlining repeated game, i.e., when more complex time-dependent strategies are in principle possible).

The above results apply to the single token setting which is the most common in PoS systems (e.g., Ethereum and Cardano operate in this way, for example, as PoS cryptocurrencies) and paint a rather negative picture: one has to either accept tokenomics with widely fluctuating token prices (the most common outcome that is observed in these systems as deployed in the real world), implement drastic measures such as token buybacks, or suffer a potential explosion in rewards accounting (unless future shocks in service value can be predicted – a rather unreasonable assumption). Note that QR mechanisms are, in general, more complex than standard "fee burning" policies used in practice (see Remark 16).

Two-token systems and QR. Poised to obtain a more favorable outcome, we next explore the implementation of QR in two token PoS systems in which one token is used to get the service and another token is used for staking (and governance). While less common, this type of setup has been adopted in a few occasions: e.g., in the NEO cryptocurrency [19] and can be implemented in other systems as well (e.g., in the Cosmos SDK [8]). To investigate the potential of QR in this setting, we adapt our model to the two tokens setting (Section 4). In order to maintain feasibility, while maintaining desirable prices, we identify the necessary conditions that equilibria (mechanism) must guarantee in terms of rewards.

The two token mechanism for QR is presented in Section 4.3. The main advantage of the resulting rewards formula is that it is simpler than in the single token setting and the mechanism does not need to know "global information" about the service value time series – merely a one step lookahead suffices (cf. Theorem 24). As an important corollary, the mechanism achieves our objectives and is capable of avoiding the feedback loop effect of the single token case that leads to an exponential explosion for rewards.

1.2 Related work

In the single-token setting, our results build on the model of [11]. In [11], the author considers a *platform design* framework in which the system designer selects both time-dependent rewards $R^{(t)}$ and service quality $S^{(t)}$. There, $S^{(t)}$ is *not* influenced by external shocks; rather, it is deterministically set by the system, which incurs a cost depending on the chosen service quality. Market frictions are captured through shocks occurring at the user level: with a fixed probability, token holdings are reset to zero at each round. The objective is to maximize system profit, and the platform is permitted to engage in the spot market, either by selling or buying back tokens. Compared to that prior work, our results accommodate a more general class of equilibria as [11] uses a "rigid" policy $R^{(t)}$ that needs

buybacks (see full version [15] for details). To the best of our knowledge, ours is the first analytical model for such systems to study their equilibria and related token price paths achieving all four desiderata we set out in the beginning of this section. Other prior work includes the effects of tokens regarding user adoption and equilibrium selection [1, 2, 18], or using token supply to promote optimal growth and service provision [6], also allowing (costly) buybacks [7] that cause underinvestment in productivity. Different aspects of the monetary characteristics of tokens have been studied in [25, 21, 22], while [16] considered token burning; notably one can view QR as a generalization of the token burning mechanism that is more versatile, cf. 16. In prior work, typically users receive the most attention, but [4] expand on the strategy space of validators and their alternative uses for tokens. Further work modeling validators' staking strategies are [5, 13, 24, 14]. While all previous work has focused on the single token design, [9] considered two-token systems and proposed several economic indicators (most of them depending on the prices of the two tokens, and thus our price analysis at equilibrium can be combined with these proposed metrics).

2 Model description and notation (single token)

We consider a Lagos-Wright ([17]) type of model, in which there are two kinds of players: users and validators, as in [16, 11]. We have a repeated game where each round t, consists of two phases or *subrounds*. Players first interact with the system (users pay the system to get some service, and validators stake tokens to access rewards for performing some task), and then they interact with a "spot market" (players buy or sell their tokens at the end of the round in order to hold the amount of tokens they need at the next round). In order to describe the model in detail, we introduce some notation in Figure 1. The two subrounds are as follows:

Subround I (pay & stake)

Each user i and each validator j has $\mathbb{TK}_i^{(t)}$ and $\mathbb{TK}_j^{(t)}$ tokens, respectively. Then, given some (total) value of the service $S^{(t)}$ and the total rewards $R^{(t)}$ for validators, we have

1. Each validator j performs some work, incurs a cost v, and receives an amount $R_j^{(t)}$ of additional tokens according to Equation (1), given the total reward for validators $R^{(t)}$.
2. Each user i uses some of her currently available tokens to pay for the service, and receives the corresponding service value $S_i^{(t)}$ according to Equation (2), given the value of the service $S^{(t)}$.

Subround II (buy or sell)

Each user i and each validator j has $\mathbb{TK}_i^{(t)} - u_i^{(t)}$ and $\mathbb{TK}_j^{(t)} + R_j^{(t)}$ tokens, respectively. Both type of players (users and validators) can buy any amount of new tokens or sell (part or all of the tokens they currently have) at the current market price. In detail:

1. Each validator j sells $s_j^{(t)}$ tokens to the market and receives $s_j^{(t)} \cdot \mathbb{PRICE}^{(t)}$ units of money in return. Note that we have a validator's selling constraint

$$s_j^{(t)} \leq \mathbb{TK}_j^{(t)} + R_j^{(t)} \ . \tag{5}$$

   Any *negative* $s_j^{(t)}$ is allowed, meaning that j is actually *buying* tokens from the market at the current price.

Notation (single token model)

- m = number of users (fixed and constant over time).
- n = number of validators (fixed and constant over time).
- $\mathbb{TK}_p$ = token holding of a generic player p (user or validator).
- $\mathbb{PRICE}$ = price of the token.
- $f^{(t)}$ the value of a generic quantity f at time t.
- $f^{(\delta,\infty)}$ = discounted version of quantity $f^{(t)}$, that is, $f^{(\delta,\infty)} = \sum_{t=0}^{\infty} \delta^t f^{(t)}$.
- $f^{(\delta,\infty|t)}$ = generalization of the previous definition in which we start at t and discount accordingly, that is, $f^{(\delta,\infty|t)} = \sum_{\tau=t}^{\infty} \delta^{\tau-t} f^{(\tau)} = f^{(t)} + \delta f^{(\delta,\infty|t+1)}$.
- $s_j^{(t)}$ = validator j's selling strategy at time t (negative values are possible, i.e., buying).
- $b_i^{(t)}$ = user i's buying strategy at time t (negative values are possible, i.e., selling).
- $u_i^{(t)}$ = amount of tokens that user i pays for the service at time t.
- v = cost incurred by each validator (identical for all validators and all t).
- $R^{(t)}$ is the total reward for validators at time t, which is further distributed to each validator according to a reward sharing scheme $r(\cdot)$, meaning that validator j receives an amount of tokens equal to

$$R_j^{(t)} := R^{(t)} \cdot r(\mathbb{TK}_j^{(t)}, \mathbb{TK}_{-j}^{(t)}) \ . \tag{1}$$

  where $\mathbb{TK}_{-j}^{(t)}$ are the token holding of all validators but j.
- $S^{(t)}$ is the service value at time t, which is further divided among the users according to some scheme $s(\cdot)$, meaning that each user i receives a service value

$$S_i^{(t)} := S^{(t)} \cdot s(u_i^{(t)}, u_{-i}^{(t)}) \ . \tag{2}$$

  where $u_{-i}^{(t)}$ are the used tokens of all users but i.
- $g(n)$ = generic non-decreasing decentralization factor, where n is the number of validators.
- $U_i^{(t)}$ = instantaneous utility of user i, given by

$$U_i^{(t)} = S_i^{(t)} \cdot g(n) - b_i^{(t)} \cdot \mathbb{PRICE}^{(t)} \ . \tag{3}$$

- $V_j^{(t)}$ = instantaneous utility of validator j, given by

$$V_j^{(t)} = s_j^{(t)} \cdot \mathbb{PRICE}^{(t)} - v \ . \tag{4}$$

Figure 1 Notation and symbols.

2. Each user i buys $b_i^{(t)}$ additional tokens from the market and pays $b_i^{(t)} \cdot \mathbb{PRICE}^{(t)}$ units of money for that. Any *negative* $b_i^{(t)}$ is allowed, meaning that i is *selling* tokens to the market at the current price. Note that we have a users's selling constraint

$$-b_i^{(t)} \leq \mathbb{TK}_i^{(t)} - u_i^{(t)} \ . \tag{6}$$

At the end of subround II of step t we get the token holdings for the next step, $t+1$, for each user i and each validator j, respectively

$$\mathbb{TK}_i^{(t+1)} = \mathbb{TK}_i^{(t)} - u_i^{(t)} + b_i^{(t)} \overset{(6)}{\geq} 0 \qquad\qquad \mathbb{TK}_j^{(t+1)} = \mathbb{TK}_j^{(t)} + R_j^{(t)} - s_j^{(t)} \overset{(5)}{\geq} 0 \ . \tag{7}$$

Each user i and each validator j aims at maximizing her own *discounted* utility, respectively

$$U_i^{(\delta,\infty)} \overset{(3)}{=} \sum_{t=0}^{\infty} \delta^t \cdot [S_i^{(t)} \cdot g(n) - b_i^{(t)} \cdot \mathbb{PRICE}^{(t)}] \tag{8}$$

$$V_j^{(\delta,\infty)} \overset{(4)}{=} \sum_{t=0}^{\infty} \delta^t \cdot [s_j^{(t)} \cdot \mathbb{PRICE}^{(t)} - v] \tag{9}$$

subject to their respective selling constraints in (6) and in (5). As usual, the discount factor δ above is $\delta = 1/(1+r)$ where r is the risk free rate.[1]

▶ **Definition 1** (symmetric equilibrium). *Consider a system policy given by* (1) *and* (2)*, and a triple of users' strategies, validators' strategies, and prices*

$$(u_i^{(t)}, b_i^{(t)}) \qquad\qquad s_j^{(t)} \qquad\qquad \mathbb{PRICE}^{(t)}$$

such that the corresponding selling constraints (6) *and* (5) *are satisfied. We say that such a triple is an* equilibrium *if*

1. *Strategy $s_j^{(t)}$ maximizes the discounted utility* (9) *of validator j, given all other strategies, for each validator j;*
2. *Strategy $(u_i^{(t)}, b_i^{(t)})$ maximizes the discounted utility* (8) *of user i, given all other strategies, for each user i;*
3. *The resulting two sequences of token holdings* (7) *are strictly positive, that is, $\mathbb{TK}_i^{(t)} > 0$ and $\mathbb{TK}_j^{(t)} > 0$.*

Moreover, we say that such an equilibrium is symmetric *if the token holdings of all users are the same, and similarly, if all token holdings of all validators are the same, correspond to sequences of strictly positive token holdings*

$$\mathbb{TK}_i^{(t)} = \mathbb{TK}_U^{(t)} > 0 \qquad\qquad \mathbb{TK}_j^{(t)} = \mathbb{TK}_V^{(t)} > 0 \ . \tag{10}$$

for all users i and for all validators j and for all t.

We should point out that, in addition to being natural, the symmetry conditions (10) are often without loss of generality (see full version [15] for details).

▶ Remark 2 (viability and equilibria). The condition that tokens holding must be strictly positive at all time steps implies that the selling/buying strategies at such an equilibrium satisfy the selling constraints (6) and (5) with a strict inequality. This condition captures our *viability* requirement of keeping all parties engaged.

[1] Intuitively, the system is analyzed as investment "against" the risk-free investment which is captured abstractly by the risk-free rate. Also, at equilibrium validators do not necessarily break even (as e.g., in prior work [11]) but they can have positive utility at each step, cf. Section 4.3.

In order to establish the existence of equilibria, we need a few (technical) assumptions, regarding the service value (Assumption 1) and about the rewards and service sharing functions (Assumption 2 below).

▶ **Assumption 1** (service value)**.** *The service value $S^{(t)}$ is upper bounded by some (arbitrarily large) constant independent of t.*

The value of the constant in Assumption 1 does not affect the results and bounds in any way, and is only needed to establish the existence of equilibria. Other than this, we make no other assumptions, and allow $S^{(t)}$ to increase or decrease arbitrarily within this range.

▶ Remark 3 (service value fluctuations – shocks). The service value $S^{(t)}$ represents the overall value the system provides to users. Several factors (technological improvements, competitors, regulations, market conditions) influence $S^{(t)}$. Consequently, the system may neither fully control nor predict all future values of $S^{(t)}$, as these factors can cause unforeseen shocks.

3 Single token mechanisms

3.1 Analysis of price path

In this section we provide the analysis of the prices for the following family of service allocation and rewards:

$$S_i^{(t)} = S^{(t)} \cdot s\left(\frac{u_i^{(t)}}{\sum_k u_k^{(t)}}\right) \qquad\qquad R_j^{(t)} = R^{(t)} \cdot r\left(\frac{\mathbb{TK}_j^{(t)}}{\sum_v \mathbb{TK}_v^{(t)}}\right) \tag{11}$$

where $s(\cdot)$ and $r(\cdot)$ are arbitrary differentiable functions in $(0,1)$, and where the indexes k and v in the summations range over all users and all validators, respectively. In the following, we denote by $r'(\cdot)$ and $s'(\cdot)$ the first derivatives of $r(\cdot)$ and $s(\cdot)$, respectively. We make the following assumption about $r(\cdot)$ and $s(\cdot)$.

▶ **Assumption 2.** *We generalize [11] and assume that the functions $r(\cdot)$ and $s(\cdot)$ in (11) are both concave. Moreover, for the number $m \geq 2$ and $n \geq 2$ of users and validators under consideration, they never allocate more than the total rewards or service value available, $r(1/n) \leq 1/n$ and $s(1/m) \leq 1/m$, and the first derivatives satisfy $r'(1/n) > 0$ and $s'(1/m) > 0$.*

▶ **Lemma 4.** *In any symmetric equilibrium, it holds that*

$$\mathbb{PRICE}^{(t-1)} = \mathbb{PRICE}^{(t)} \cdot \delta \cdot \mathcal{R}^{(t)} \ , \qquad \mathcal{R}^{(t)} = 1 + \frac{R^{(t)}}{n\mathbb{TK}_V^{(t)}} \cdot \frac{n-1}{n} \cdot r'(\frac{1}{n}) \ . \tag{12}$$

Moreover

$$\mathbb{PRICE}^{(t-1)} = \delta \cdot \begin{cases} \mathcal{S}^{(t)} & \text{if } u_i^{(t)} = \mathbb{TK}_U^{(t)} \\ \mathbb{PRICE}^{(t)} & \text{if } u_i^{(t)} < \mathbb{TK}_U^{(t)} \end{cases} \ , \quad \mathcal{S}^{(t)} = \frac{S^{(t)}}{m\mathbb{TK}_U^{(t)}} \cdot g(n) \cdot \frac{m-1}{m} \cdot s'(\frac{1}{m}). \tag{13}$$

Hence, if $\mathcal{R}^{(t)} \neq 1$, then $u_i^{(t)} = \mathbb{TK}_U^{(t)}$ and $b_i^{(t)} = \mathbb{TK}_U^{(t+1)}$ for all users i, and the following identity must hold:

$$\mathbb{PRICE}^{(t)}\mathcal{R}^{(t)} = \mathcal{S}^{(t)}. \tag{14}$$

A simple class of schemes satisfying our assumptions is the following one from [11]:

$$S_i^{(t)} = S^{(t)} \cdot \left(\frac{u_i^{(t)}}{\sum_k u_k^{(t)}} \right) \qquad R_j^{(t)} = R^{(t)} \cdot \left(\frac{\mathbb{TK}_j^{(t)}}{\sum_v \mathbb{TK}_v^{(t)}} \right) . \tag{15}$$

Example 5 below provides another possible reward sharing function $r(\cdot)$ other than the one in (15). We do not claim that this alternative $r(\cdot)$ provides any particular improvement, but simply point out that changing $r(\cdot)$, and similarly $s(\cdot)$, does affect the equilibria as described by Lemma 4 (see full version [15] for further examples and discussion).

▶ **Example 5.** For any parameter $\ell > 1$, the following reward sharing function $r(x) = x - x^\ell$ satisfies $r(1/n) = 1/n - 1/n^\ell > 0$, meaning that for smaller number of validators a smaller fraction of the total allocated rewards is actually distributed. Since $r'(1/n) = 1 - \ell/n^{\ell-1}$, we have $r'(1/n) > 0$ for sufficiently large n.

▶ Remark 6. Assumption 2 implies that $\mathcal{R}^{(t)} > 1$ whenever $R^{(t)} > 0$, thus implying that the largest possible price growth must satisfy $\mathbb{PRICE}^{(t)} < \mathbb{PRICE}^{(t-1)}/\delta = (1+r) \cdot \mathbb{PRICE}^{(t-1)}$ where $\delta = 1/(1+r)$ and r is the risk-free rate. Intuitively, the return rate for just holding the token cannot beat the risk-free rate, as one might expect when looking for equilibria that keep all users and validators engaged (viability). Whether blockchains can achieve return rates competitive with the inflation is studied in [10] in relation to policies adopted by some of the current blockchains, though without providing analytical results.

3.2 Generic symmetric equilibria

In the following we focus on the interesting case of prices having a constant multiplicative growth, which includes stable prices as a special case.

▶ **Definition 7** (γ-stable prices)**.** *We say that the prices (of some equilibrium under consideration) are γ-stable if, for all $t \geq 1$, they satisfy*

$$\mathbb{PRICE}^{(t)} = \frac{1}{\gamma} \cdot \mathbb{PRICE}^{(t-1)} \qquad \gamma \neq \delta,\ \gamma > 0 . \tag{16}$$

Note that stable prices satisfy the above condition with $\gamma = 1$, while $\gamma > 1$ and $\gamma < 1$ correspond to decreasing and increasing prices, respectively.

We next define a class of generic symmetric equilibria which, as we prove below, captures all symmetric equilibria with γ-stable prices.

▶ **Definition 8** (generic symmetric equilibrium)**.** *A symmetric equilibrium is generic if the following conditions hold for all $t \geq 1$ and constants $\kappa_S, \kappa_R, \mathrm{Rew2Stake}, \mathrm{Ser2Fees} \in \mathbb{R}$:*

1. *The monetary amount of tokens that users hold (and use) is proportional to the value of the service offered by the system. That is, the service to fees ratio is constant,*

$$\mathrm{Ser2Fees}^{(t)} := \frac{S^{(t)}}{m\mathbb{TK}_U^{(t)} \cdot \mathbb{PRICE}^{(t)}} = \mathrm{Ser2Fees} .$$

2. *The amount of tokens staked by validators is proportional to the rewards offered by the system. That is, the rewards to stake ratio is constant,*

$$\mathrm{Rew2Stake}^{(t)} := \frac{R^{(t)}}{n\mathbb{TK}_V^{(t)}} = \mathrm{Rew2Stake} .$$

3. *The prices satisfy*

$$\begin{aligned}\mathbb{PRICE}^{(t-1)} =& \delta \cdot \mathbb{PRICE}^{(t)} \cdot (1 + \text{Rew2Stake} \cdot \kappa_R) \\ =& \delta \cdot \mathbb{PRICE}^{(t)} \cdot \text{Ser2Fees} \cdot \kappa_S\end{aligned}$$

where constants κ_R and κ_S depend only on the number of users m, the number of validators n, the rewards sharing scheme, and on the service fee scheme.

4. *Each user i starts round t with the same token holding $\mathbb{TK}_U^{(t)}$, uses all its token current holding for the service ($u_i^{(t)} = \mathbb{TK}_U^{(t)}$), and buys new tokens needed for the next round accordingly ($b_i^{(t)} = \mathbb{TK}_U^{(t+1)}$).*

Note that generic symmetric equilibria provide additional structure to the definition of symmetric equilibria. The following theorem says that, when aiming at γ-stable prices (Definition 7), we can restrict ourselves to generic symmetric equilibria without loss of generality. In particular, this holds true for stable prices.

▶ **Theorem 9.** *Any symmetric equilibrium for the reward and service fee schemes in* (11) *with γ-stable prices is a generic symmetric equilibrium with constants*

$$\kappa_R = \frac{n-1}{n} \cdot r'(1/n) \qquad \textit{and} \qquad \kappa_S = g(n) \cdot \frac{m-1}{m} \cdot s'(1/m) \ .$$

This in particular holds true for the case of stable prices.

Note that stable prices require the following *specific* constants in the two ratios involving the tokens:

$$\text{Ser2Fees} = 1/(\delta\kappa_S) \qquad \text{Rew2Stake} = (1-\delta)/(\delta\kappa_R) \tag{17}$$

thus implying that it must hold $\kappa_R > 0$ and $\kappa_S > 0$. Theorem 9 then implies the following.

▶ **Corollary 10.** *Any generic symmetric equilibrium (and thus any symmetric equilibrium for the reward and service fee schemes in* (11)*) with stable prices must have the following token holdings:*

$$\mathbb{TK}_U^{(t)} = \frac{S^{(t)}}{m} \cdot \delta \cdot \kappa_S > 0 \qquad \textit{and} \qquad \mathbb{TK}_V^{(t)} = \frac{R^{(t)}}{n} \cdot \frac{\delta}{1-\delta} \cdot \kappa_R > 0$$

for any nonnegative $S^{(t)}$ and $R^{(t)}$.

▶ Remark 11 (higher rewards improve PoS security). Security of PoS protocols relies on the total amount of staked tokens – the higher, the better, as an attacker needs at least a fraction of this amount to succeed. The results above and Item 2 in Definition 8 indicate that the amount of staked tokens at equilibrium is *proportional* to the rewards. Therefore, increasing rewards improves the system's security, a strategy even suggested in some implementations, such as Polkadot [3].

3.3 The implications of no buybacks and Quantative Rewarding

In the model considered so far, the system is allowed to buy or to sell the additional tokens required during each subround II to match demand with supply. In this section, we refine a concept of equilibrium, by requiring that the system does not have to buyback tokens (and perhaps does not sell tokens either if demand and supply match at equilibrium). We stress that this refers only to the "spot market" (subround II) where tokens are exchanged

for *money*. Indeed, these Lagos-Wright "two-stage" models have a simple implementation (subround I): The system either burns some fees (when rewards are less than the paid fees) or mints additional new tokens for rewards (when rewards are more than paid fees). Note that *no monetary reserve* is needed for these operations.

▶ **Definition 12** (no buyback). *We say that a symmetric equilibrium satisfies the no buyback condition if no additional tokens are bought at any round by the system, that is,* $mb_U^{(t)} \geq ns_V^{(t)}$ *for all* t*. Additionally, we say that the no buyback condition holds tightly if demand matches supply, that is,* $mb_U^{(t)} = ns_V^{(t)}$*.*

Note that the quantities $m \cdot b_U$ and $n \cdot s_V$ correspond to the token *demand* and *supply* in the "market" subround II, respectively. As we formally prove below, the demand and supply are determined by the service value and rewards, respectively:

- *Demand* ($mb_U^{(t)}$ from users) decreases if the next round's *service value* decreases (Equation (19) below).
- *Supply* ($ns_V^{(t)}$ from validators) decreases with the next round's *rewards* (Equation (20) below).

This naturally suggests the following *quantitative rewarding* approach to avoid buybacks:

1. (*demand decreases* ⇒ *increase rewards.*) Whenever the next round's service value "decreases significantly", in order to adjust supply to the decreased demand, we *increase* the next round's rewards. This will make rewards more attractive for the validators, who then stake more tokens (and thus sell less on the market).
2. (*demand increases* ⇒ *decrease rewards.*) Whenever the next round's service value "increases significantly," we can *decrease* the next round's rewards. This will have the opposite effect on validators, who will sell more tokens on the market.

The first case is necessary to avoid buybacks (satisfy Definition 12). The second operation turns out to be useful for (i) restoring payments to a lower value when possible and (ii) satisfying the no buybacks condition *tightly*, meaning that demand never exceeds supply ($mb_U = ns_V$). This stronger condition has the advantage that the system does not even need to implement a minting mechanism for users to buy tokens (see Remark 15 below).

The following theorem establishes an equivalence between the no buybacks condition and the above quantitative rewarding (rewards increase). In particular, it quantifies the minimum rewards for which it is possible to avoid buybacks at a given round.

▶ **Theorem 13.** *In any generic symmetric equilibrium, the no buyback condition holds at round* t *if and only if the rewards satisfy the following condition*

$$R^{(t+1)} \geq R^{(t)} \cdot (1+a) - b \cdot S^{(t+1)} \cdot (\delta\mathcal{R})^{t+1} \tag{18}$$

where

$$a = \text{Rew2Stake} \cdot n \cdot r(1/n) \ , \quad b = \frac{\text{Rew2Stake}}{\text{Ser2Fees}} \cdot \frac{1}{\mathbb{PRICE}^{(0)}} \ , \quad \mathcal{R} = 1 + \text{Rew2Stake} \cdot \kappa_R \ .$$

Proof. Let us first observe that the total amount of tokens bought by the users (demand) is

$$mb_U^{(t)} = m\mathbb{TK}_U^{(t+1)} = \frac{S^{(t+1)}}{\mathbb{PRICE}^{(t+1)} \cdot \text{Ser2Fees}} = \frac{S^{(t+1)} \cdot (\delta\mathcal{R})^{t+1}}{\mathbb{PRICE}^{(0)} \cdot \text{Ser2Fees}} \tag{19}$$

where $\mathcal{R} = 1 + \text{Rew2Stake} \cdot \kappa_R$. The total amount of tokens sold by the validators (supply) is

$$\begin{aligned} ns_V^{(t)} &= n \cdot \left(\mathbb{TK}_V^{(t)} - \mathbb{TK}_V^{(t+1)} + R^{(t)} \cdot r(1/n) \right) \\ &= \frac{R^{(t)}}{\text{Rew2Stake}} - \frac{R^{(t+1)}}{\text{Rew2Stake}} + R^{(t)} \cdot n \cdot r(1/n) \ . \end{aligned} \tag{20}$$

Hence, the no buyback condition $mb_U^{(t)} \geq ns_V^{(t)}$ is equivalent to

$$\frac{R^{(t)}}{\text{Rew2Stake}} - \frac{R^{(t+1)}}{\text{Rew2Stake}} + R^{(t)} \cdot n \cdot r(1/n) \leq \frac{S^{(t+1)} \cdot (\delta\mathcal{R})^{t+1}}{\mathbb{PRICE}^{(0)} \cdot \text{Ser2Fees}}$$

that is

$$R^{(t)} - R^{(t+1)} + R^{(t)} \cdot n \cdot r(1/n) \cdot \text{Rew2Stake} \leq \frac{\text{Rew2Stake}}{\mathbb{PRICE}^{(0)} \cdot \text{Ser2Fees}} \cdot S^{(t+1)} \cdot (\delta\mathcal{R})^{t+1} \ .$$

By rearranging the terms, the theorem follows. ◀

The above theorem provides a recursive (algorithmic) formula for the rewards, which leads to our mechanism described in Section 3.4 below. By unfolding the recursion (18) in same theorem, we get the following bound on the growth of the rewards necessary to guarantee the no buyback.

▶ **Corollary 14.** *The minimal rewards satisfying the no buyback condition (tightly) are equal to*

$$R_{single}^{(t+1)} = (1+a)^t \cdot \left(R^{(0)} - b \sum_{\tau=1}^{t} S^{(\tau)} \cdot \left(\frac{\delta\mathcal{R}}{1+a} \right)^{\tau} \right) \ , \qquad t \geq 1$$

for any initial reward $R^{(0)}$, and for constants a, b, and $\mathcal{R}$ as in Theorem 13.

We stress that the *minimal* rewards satisfying the no buyback condition must satisfy the no buyback condition *tightly*, and thus must coincide with the rewards $R_{single}^{(t)}$ in Corollary 14.

▶ Remark 15 (avoid users minting with tight no buyback). Observe that the no buyback condition only requires supply to not exceed demand (Definition 12). Hence, it is possible that (users') demand exceeds (validators') supply ($mb_U^{(t)} - ns_V^{(t)} > 0$). In this case, the system has to provide this difference of tokens to the market (users). On the one hand, it allows the system to make some further profit. On the other hand, there are scenarios in which this might be difficult or undesirable (typically, it requires the system to implement a "special" minting mechanism that allows users to buy tokens). In such cases, we require a *tight* version of the no buyback condition (Definition 12) in which demand matches supply. The analysis of this tight no buyback condition is given by Corollary 14, while the formula in Theorem 13 with equality gives the recursion for the rewards.

As we discuss below, implementing quantitative rewarding can be difficult due to the possibility of an uncontrolled rewards growth. Indeed, Corollary 14 states that the initial rewards $R^{(0)}$ cannot exceed a certain value unless rewards grow exponentially. This value, critically, depends on future service values. In Section 3.4, we describe an implementation under the assumption that the initial rewards $R^{(0)}$ can be set properly by the system (either having full knowledge about future service values, a good estimate, or some partial control over them).

▶ Remark 16 (QR vs fees burning). In practice, the system collects fees paid in round t and uses part of them as rewards in the same round. Validators receive $R^{(t)} \cdot n \cdot r(1/n)$ tokens in total, defining the "rewards-over-fees" ratio as $ROF^{(t)} := \frac{R^{(t)} \cdot n \cdot r(1/n)}{m\mathbb{TK}_U^{(t)}}$. Systems that burn a constant fraction of fees and set the remainder as rewards maintain a constant $ROF^{(t)} < 1$. Quantitative rewarding is more general: (1) $ROF^{(t)}$ can vary over time, implying a nonconstant burn rate, and (2) $ROF^{(t)} > 1$ allows monetary expansion, introducing extra reward tokens (still avoiding user minting according to Remark 15).

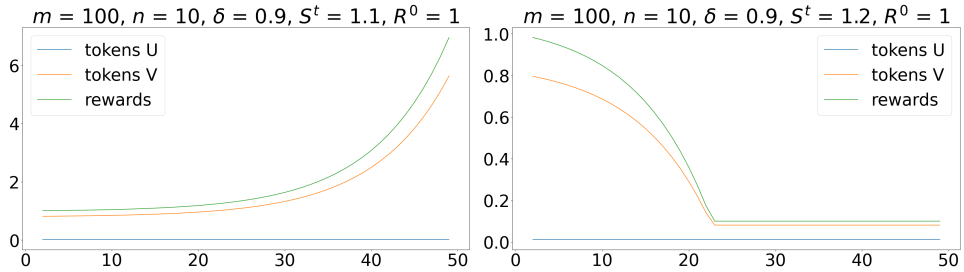


Figure 2 The effect of service values on rewards growth when no buyback condition: a small service value (left) can trigger an exponential increase in both rewards and validators staked amounts, while a slightly bigger service value (right) can remove this growth.

3.3.1 Main dilemma: Uncontrolled growth of rewards

The bound in Corollary 14 suggests that an *exponential* growth of the rewards over time is necessary if the system starts with a too high initial rewards or the service value is too low. Hence, although larger rewards might be desirable to improve security (Remark 11), some care is needed in case of "shocks" in the service value.

In the following experiment, we consider constant service value and rewards that are set to the minimal value necessary in order to have no buyback (Theorem 13), and also impose the rewards to not be below some minimum value (set to 0.1 for the sake of exposition). For the sake of simplicity, we consider the rewards and cost sharing schemes in (15) and $g(n) = 1$, and observe the following:

- Sufficiently high service value can avoid rewards explosion (Figure 2 compares two different service values under the same conditions). The necessary increase in the service value may simply be not possible due to inherent technological limitations and other external factors.
- Sufficiently small initial rewards also avoid the rewards explosion (Figure 3 compares two initial rewards under the same conditions). Rewards however cannot be arbitrarily small since they must cover the costs of validators and be competitive against other sources of investments.
- A higher risk free rate may also trigger an exponential growth (Figure 4 shows that for smaller δ we need a smaller initial reward, or a larger service value, to avoid this explosion).
- Allowing the token price to decrease does avoid the rewards explosion that instead occur with stable prices, once we consider the rewards in money (Figure 5). Decreasing prices are however not desirable, and perhaps one may want increasing prices, which turn out to make the monetary reward explosion even more severe.

The above uncontrolled growth represents a potential "death spiral" stemming from necessary equilibrium conditions. First, it leads to impractical implementation due to numerical explosion in ledger validators accounts (thus violating feasibility). Second, it may even render such equilibria nonexistent for some parameter combinations. Note that adapting the rewards requires knowledge about shocks far in the future. Indeed, Theorem 13 states that rewards cannot be reduced arbitrarily from one step to the next. Such powerful oracles are generally not available, and solutions based on them also violate feasibility.

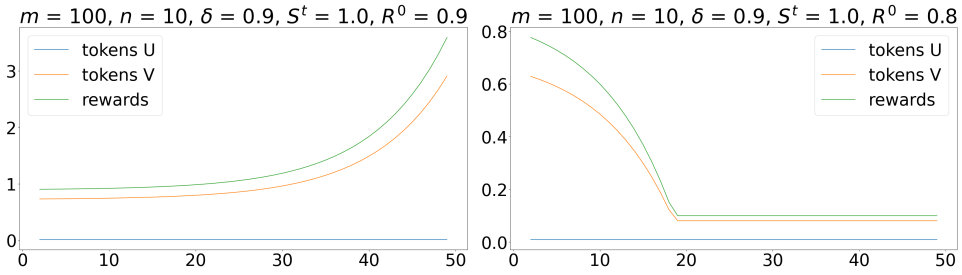


Figure 3 A too high initial rewards (left) causes an exponential growth, as opposed to a smaller initial rewards (right) leading to stable rewards and staked tokens (note the different scale of the y-axis in the two plots). In both cases, we fix a minimum value of 0.1 for the rewards, which is the flat part of the green line in the right plot.

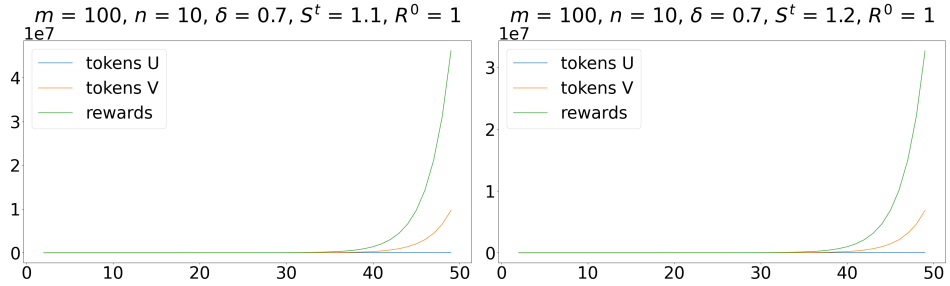


Figure 4 The effect of risk free rate (δ) on rewards growth when no buyback condition: a smaller δ might trigger an exponential increase in both rewards and validators staked amounts (compare the right picture here with the right picture in Figure 2).

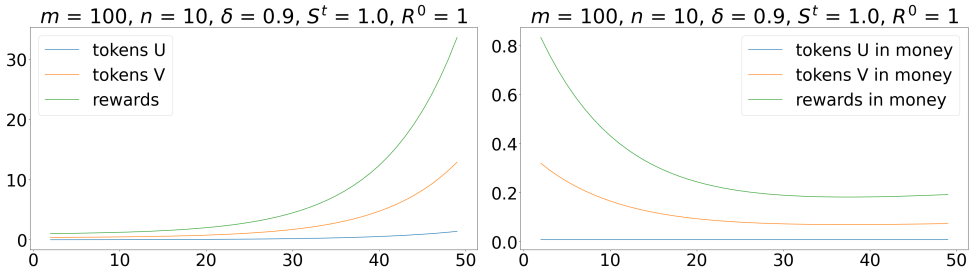


Figure 5 The effect of non-stable prices on rewards and validators staked tokens. We set decreasing prices according to the price growth parameter $\delta\mathcal{R} = 0.9$. Though the rewards and staked tokens increase exponentially (left) their value expressed in money – rewards and staked tokens market cap – does not (right).

3.4 Policy that accommodates and implements the stable price equilibrium

In this section, we consider the fundamental question of how one can compute and implement an equilibrium with stable prices satisfying the no buyback condition. Stable prices require to keep a precise amount of tokens of users (on one side) and of tokens of validators (on the other side) given by Corollary 10. The no buyback condition requires the system to set the rewards so to satisfy the condition in Theorem 13. For the sake of exposition, we next describe an implementation for the reward and service fee schemes in (11). We should point out that this implementation does not specify how to set the initial rewards $R^{(0)}$ which is the crucial point to avoid rewards explosion (Corollary 14).

Given parameters and constraints.

1. The risk free rate r and thus the discount factor $\delta = 1/(1+r)$.
2. The total number m of users and the total number n of validators (both constant over time).
3. There is no need for the system to buyback tokens (Definition 12).

System parameters. The system design depends on essentially two parameters:

1. Total value of the service $S^{(t)}$ which is divided among the users at each step t.
2. Total amount of rewards $R^{(t)}$ which is distributed to the validators at each step t.

Suggested equilibrium description and properties. At each round $t \geq 0$,

1. Users strategies:
   a. Start with $\mathbb{TK}_U^{(t)} = \frac{S^{(t)}}{m} \cdot \delta \cdot \kappa_S$ tokens, where $\kappa_S = g(n) \cdot \frac{m-1}{m} \cdot s'(1/m)$.
   b. Spend all these tokens to get a service value equal to $S^{(t)} \cdot s(1/m)$.
   c. Given the next round service value $S^{(t+1)}$, buy $b_U^{(t)} = \mathbb{TK}_U^{(t+1)} = \frac{S^{(t+1)}}{m} \cdot \delta \cdot \kappa_S$ tokens required for the next round according to Item 1a.
2. Validators strategies:
   a. Start with $\mathbb{TK}_V^{(t)} = \frac{R^{(t)}}{n} \cdot \frac{\delta}{1-\delta} \cdot \kappa_R$ tokens, where $\kappa_R = \frac{n-1}{n} \cdot r'(1/n)$.
   b. Stake all these tokens to get $R^{(t)} \cdot r(1/n)$ new tokens as reward.
   c. Given the rewards $R^{(t+1)}$ of next round, sell this amount of tokens (if negative, buy tokens): $s_V^{(t)} = \left(R^{(t)} - R^{(t+1)}\right) \cdot \frac{\delta}{1-\delta} \cdot \kappa_R + R^{(t)} \cdot r(1/n)$, which yields the required $\mathbb{TK}_V^{(t+1)} = \frac{R^{(t+1)}}{n} \cdot \frac{\delta}{1-\delta} \cdot \kappa_R$ tokens for next round $t+1$ according to Item 2a.
3. System actions:
   a. Given the next round service value $S^{(t+1)}$, set the next total rewards $R^{(t+1)}$ so that

$$m \cdot S^{(t+1)} \cdot \delta \cdot \kappa_S \geq n \cdot \left(R^{(t)} - R^{(t+1)}\right) \cdot \frac{\delta}{1-\delta} \cdot \kappa_R + n \cdot R^{(t)} \cdot r(1/n)$$

   which guarantees the no buyback condition, that is, $m \cdot b_U^{(t)} \geq n \cdot s_V^{(t)}$.
   b. Announce the next rewards $R^{(t+1)}$ before the token buy or sell (Subround II) begins.
   c. Sell the required amount of tokens, $m\mathbb{TK}_U^{(t+1)} - ns_V^{(t)}$, to match demand during Subround II.
4. Prices: all tokens are exchanged at a stable price $\mathbb{PRICE}^{(t)} = 1$.

System	token **A**	purpose	token **B**	purpose
DFINITY ([27])	ICP	staking & rewards	Cycle	computation (service)
NEO ([19])	NEO	staking	GAS	pay transactions
Axie Infinity ([12])	AXS	governance and staking	SLP	play (breeding Axie)
IOTA ([23])	IOTA	staking & rewards	Mana	access ledger

Figure 6 Examples of two-token systems.

- $R^{(t)} = (R^{(t)}_{\mathbf{A}}, R^{(t)}_{\mathbf{B}})$ is the reward at time t.
- $s^{(t)}_j = (s_j{}^{(t)}_{\mathbf{A}}, s_j{}^{(t)}_{\mathbf{B}})$ is the selling strategy of validator j.
- $b^{(t)}_i = (b_i{}^{(t)}_{\mathbf{A}}, b_i{}^{(t)}_{\mathbf{B}})$ is the buying strategy of user i.
- $u^{(t)}_i$ is the amount of token **B** used by user i to pay for the service.
- $\mathbb{TK}^{(t)}_p = (\mathbf{A}^{(t)}_p, \mathbf{B}^{(t)}_p)$ are the token holdings of a generic player p (a user or a validator).
- $\mathbb{PRICE}^{(t)} = (\mathbb{PRICE}^{(t)}_{\mathbf{A}}, \mathbb{PRICE}^{(t)}_{\mathbf{B}})$ are the prices of the two tokens.

Figure 7 Notation for the two-token model.

4 A two-token model

In this section, we consider a natural variant of the previous (single-token) model to the case in which we have two tokens, token **A** and **B**, whose use and purpose is as follows:

- token **B** is used by the users to pay and get the service, and
- token **A** is used by the validators for staking to get rewarded with more tokens (of both types).

A number of existing systems follow a similar two-token scheme (see Figure 6 for some examples) and a model based on the similar assumptions has recently been proposed by [9]. Both tokens can be exchanged in the spot market as before, and thus we have a price for each token (see Figure 7 for some additional notation for the two-token model). The corresponding utilities are naturally given by

$$U_i^{(\delta,\infty)} = \sum_{t=0}^{\infty} \delta^t \cdot U_i^{(t)} \ , \qquad U_i^{(t)} = S_i^{(t)} \cdot g(n) - b_i{}^{(t)}_{\mathbf{A}} \cdot \mathbb{PRICE}^{(t)}_{\mathbf{A}} - b_i{}^{(t)}_{\mathbf{B}} \cdot \mathbb{PRICE}^{(t)}_{\mathbf{B}} \ . \tag{21}$$

$$V_j^{(\delta,\infty)} = \sum_{t=0}^{\infty} \delta^t \cdot V_j^{(t)} \ , \qquad V_j^{(t)} = s_j{}^{(t)}_{\mathbf{A}} \cdot \mathbb{PRICE}^{(t)}_{\mathbf{A}} + s_j{}^{(t)}_{\mathbf{B}} \cdot \mathbb{PRICE}^{(t)}_{\mathbf{B}} - v \ . \tag{22}$$

For the sake of exposition, we consider the simpler reward sharing and payment schemes (15), where token **A** is used for staking, and each validator j is rewarded with quantities $R^{(t)}_{\mathbf{A}j}$ and $R^{(t)}_{\mathbf{B}j}$ of tokens **A** and **B**, respectively. Users get a service value depending on the amount of tokens **B** they use, i.e., they pay to the system. Thus, we have

$$S_i^{(t)} = S^{(t)} \cdot \left(\frac{u_i^{(t)}}{\sum_k u_k^{(t)}} \right) \ , \quad R^{(t)}_{\mathbf{A}j} = R^{(t)}_{\mathbf{A}} \cdot \left(\frac{\mathbf{A}^{(t)}_j}{\sum_v \mathbf{A}^{(t)}_v} \right) \ , \quad R^{(t)}_{\mathbf{B}j} = R^{(t)}_{\mathbf{B}} \cdot \left(\frac{\mathbf{A}^{(t)}_j}{\sum_v \mathbf{A}^{(t)}_v} \right) \ . \tag{23}$$

The validators' selling constraint for the single token (5) extends naturally into

$$\mathbf{A}_j^{(t+1)} = \mathbf{A}_j^{(t)} + R_{\mathbf{A}j}^{(t)} - s_j{}_{\mathbf{A}}^{(t)} \geq 0 \tag{24}$$

$$\mathbf{B}_j^{(t+1)} = \mathbf{B}_j^{(t)} + R_{\mathbf{B}j}^{(t)} - s_j{}_{\mathbf{B}}^{(t)} \geq 0 \tag{25}$$

Similarly, the users' selling constraint for the single token (6) extends as follows:

$$\mathbf{A}_i^{(t+1)} = \mathbf{A}_i^{(t)} + b_i{}_{\mathbf{A}} \geq 0 \tag{26}$$

$$\mathbf{B}_i^{(t+1)} = \mathbf{B}_i^{(t)} - u_i^{(t)} + b_i{}_{\mathbf{B}} \geq 0 \tag{27}$$

where the asymmetry is due to the fact that only token $\mathbf{B}$ is used for getting the service, by paying $u_i^{(t)}$ tokens. We next generalize the condition of (symmetric) equilibrium (Definition 1 and Remark 2) by requiring a "minimal" set of selling constraints to be non-strict (see Remark 18 below).

▶ **Definition 17** (symmetric equilibrium two tokens)**.** *Consider a system policy given by* (23)*, and a triple of users' strategies, validators' strategies, and prices,* $\{(u_i^{(t)}, b_i^{(t)}), s_j^{(t)}, \mathbb{PRICE}^{(t)}\}$*, such that the corresponding selling constraints of users* (26)-(27) *and of validators* (24)-(25) *are satisfied. We say that such a triple is an* equilibrium *if, for every user* i *and every validator* j

1. $(u_i^{(t)}, b_i^{(t)})$ *maximizes the discounted utility* (21) *of user* i*, given all other strategies*
2. $s_j^{(t)}$ *maximizes the discounted utility* (22) *of validator* j*, given all other strategies*
3. *the selling constraint of token* $\mathbf{B}$ *are non-strict for user* i *(resp., token* $\mathbf{A}$ *for validator* j*), and thus the corresponding token holdings are strictly positive, that is,* $\mathbf{B}_i^{(t)} > 0$ *and* $\mathbf{A}_j^{(t)} > 0$

Moreover, we say that such an equilibrium is symmetric *if the token holdings of all users are the same, and similarly, if all token holdings of all validators are the same. In particular, we have*

$$\mathbf{B}_i^{(t)} = \mathbf{B}_U^{(t)} > 0 \qquad\qquad \mathbf{A}_j^{(t)} = \mathbf{A}_V^{(t)} > 0 \tag{28}$$

for all users i *and for all validators* j *and for all* t*.*

▶ Remark 18 (minimal strict selling constraints). Note that in the definition above, each type of player – validator or user – has non-strict selling constraint only in its own "main purpose" token (see Equation 28). This corresponds to the natural requirement of continuous participation in staking and in accessing and paying for the service. Furthermore, the price analysis below implies that equilibria with certain desired prices are impossible if some of the other selling constraints is also strict.

The next lemma is a generalization of [11, Lemma 1] to two tokens (see also [15] for details), and it provides conditions in the prices based on the validators' strategies.

▶ **Lemma 19** (validators part)**.** *In the two-token model, the corresponding prices in any symmetric equilibrium must satisfy the following conditions. If the selling constraints of* $\mathbf{B}$ *are non-strict, then* $\mathbb{PRICE}_{\mathbf{B}}^{(t-1)} = \delta \cdot \mathbb{PRICE}_{\mathbf{B}}^{(t)}$ *. If the selling constraints of* $\mathbf{A}$ *are non-strict, then*

$$\mathbb{PRICE}_{\mathbf{A}}^{(t-1)} = \delta \cdot \mathbb{PRICE}_{\mathbf{A}}^{(t)} \cdot (1 + \mathcal{I}_{\mathbf{A}}^{(t)}) + \delta \cdot \mathbb{PRICE}_{\mathbf{B}}^{(t)} \cdot \mathcal{I}_{\mathbf{B}}^{(t)} \ , \tag{29}$$

where

$$\mathcal{I}_{\mathbf{A}}^{(t)} := \frac{R_{\mathbf{A}}^{(t)}}{n\mathbf{A}_V^{(t)}} \frac{n-1}{n} \ , \qquad\qquad \mathcal{I}_{\mathbf{B}}^{(t)} := \frac{R_{\mathbf{B}}^{(t)}}{n\mathbf{A}_V^{(t)}} \frac{n-1}{n} \ . \tag{30}$$

and $\mathbf{A}_V^{(t)}$ *is the token holding (staking) of any validator at this equilibrium.*

Note that these two expressions in (30) are *not* symmetric in the two tokens, as the staking token **A** appears in both denominators.

We next consider how the users' buying strategies relate to the prices.

▶ **Lemma 20** (users part). *In the two-token model, the corresponding prices at any symmetric equilibrium must satisfy the following conditions. If the buying constraints of* **B** *are non-strict, then*

$$\mathbb{PRICE}_{\mathbf{B}}^{(t-1)} = \delta \cdot \begin{cases} \mathcal{S}^{(t)} & \text{if } u_i^{(t)} = \mathbf{B}_U^{(t)} \\ \mathbb{PRICE}_{\mathbf{B}}^{(t)} & \text{if } u_i^{(t)} < \mathbf{B}_U^{(t)} \end{cases} \qquad \mathcal{S}^{(t)} = \frac{S^{(t)}}{m\mathbf{B}_U^{(t)}} \cdot g(n) \cdot \frac{m-1}{m} \ .$$

Moreover, if the selling constraints of **A** *are non-strict, then* $\mathbb{PRICE}_{\mathbf{A}}^{(t-1)} = \delta \cdot \mathbb{PRICE}_{\mathbf{A}}^{(t)}$.

4.1 Generic symmetric equilibria

The following class of generic equilibria captures equilibria in the two-token model where prices of token **B** are γ-*stable*, that is, they satisfy (16), thus in particular the case in which we aim at stable prices for token **B** used by the user to get the service.

▶ **Definition 21** (generic symmetric equilibrium for two tokens). *A symmetric equilibrium for the two token model is generic if the following conditions hold for all* $t \geq 1$*:*

1. *The service to fees ratio is constant,*

$$Ser2Fees_{\mathbf{B}}^{(t)} := \frac{S^{(t)}}{m\mathbf{B}_U^{(t)} \cdot \mathbb{PRICE}_{\mathbf{B}}^{(t)}} = Ser2Fees_{\mathbf{B}} \ .$$

2. *The prices of* **B** *satisfy*

$$\mathbb{PRICE}_{\mathbf{B}}^{(t-1)} = \delta \cdot \mathbb{PRICE}_{\mathbf{B}}^{(t)} \cdot Ser2Fees_{\mathbf{B}} \cdot \kappa_S$$

   where constant κ_S *depends only on the number of users* m*, the number of validators* n*, and on the service fee scheme.*

3. *Each user* i *starts round* t *with the same token holding* $\mathbf{B}_U^{(t)}$*, uses all its current holding tokens for the service* ($u_i^{(t)} = \mathbf{B}_U^{(t)}$)*, and buys new tokens needed for the next round accordingly* ($b_i^{(t)} = \mathbf{B}_U^{(t+1)}$)*.*

4. *For the following rewards to stake ratios,* $Rew_{\mathbf{B}}2Stake^{(t)} := \frac{R_{\mathbf{B}}^{(t)}}{n\mathbf{A}_V^{(t)}}$*, the prices of* **A** *satisfy*

$$\mathbb{PRICE}_{\mathbf{A}}^{(t-1)} = \delta \cdot \mathbb{PRICE}_{\mathbf{A}}^{(t)} \cdot (1 + Rew_{\mathbf{A}}2Stake^{(t)}\kappa_R) + \delta \cdot \mathbb{PRICE}_{\mathbf{B}}^{(t)} \cdot Rew_{\mathbf{B}}2Stake^{(t)}\kappa_R \tag{31}$$

   where constant κ_R *depends only on the number of users* n*, and on the rewards sharing scheme.*

▶ **Theorem 22.** *Any symmetric equilibrium for the two-token model with the reward and service fee schemes in* (23) *and whose prices of token* **B** *satisfy* (16) *is a generic symmetric equilibrium. This in particular holds true for the case of stable prices for token* **B***.*

Proof. The first three items in the definition of generic symmetric equilibrium (Definition 17) follow from Lemma 20, from the definition of symmetric equilibrium requiring that users selling constraints of token **B** are non-strict (Definition 17), and from the assumption about the prices (16).

The last condition in Definition 17 is a rewriting of the prices in Lemma 19 and of definition of symmetric equilibrium requiring that validators selling constraints of token **A** are non-strict (Definition 17). ◀

4.2 No buyback in the two-token model

In this section, we study the implications of no buyback in the two-token model. As for the single token setting, we aim at equilibria where the system does not need to buyback tokens of either type.

▶ **Definition 23** (no buyback two tokens). *We say that a symmetric equilibrium in the two-token model satisfies the no buyback condition if no additional tokens (of either type) are bought at any round by the system, that is,* $mb_{i\,\mathbf{B}}^{(t)} \geq ns_{j\,\mathbf{B}}^{(t)}$ *and* $mb_{i\,\mathbf{A}}^{(t)} \geq ns_{j\,\mathbf{A}}^{(t)}$.

The following theorem provides a bound on the rewards that depends only on the next round's service value. The main advantage of the two-token model is that rewards can be adjusted "immediately" to the service value fluctuations, and they need to be low only when the latter is low. The mechanism in Section 4.3 implements this idea.

▶ **Theorem 24.** *In any generic symmetric equilibrium, there is a maximum monetary reward for validators in terms of tokens* $\mathbf{B}$ *that can be awarded without violating the no buyback condition. In particular, it must hold*

$$R_{\mathbf{B}}^{(t)} \cdot \mathbb{PRICE}_{\mathbf{B}}^{(t)} \leq \delta \cdot S^{(t+1)} \cdot \frac{\kappa_S}{n \cdot r(1/n)} \ .$$

This is because validators always sell all the newly rewarded $\mathbf{B}$ *tokens and users buy (only) tokens of type* $\mathbf{B}$*, thus implying that validators keep all their tokens* $\mathbf{A}$ *and possibly buy new ones,* $\mathbf{A}_V^{(t)} \geq \mathbf{A}_V^{(t-1)}$.

Proof. We consider the two inequalities of the no buyback condition (Definition 23) separately.

1. For tokens of type $\mathbf{B}$, we observe that, for $\mathbb{PRICE}_{\mathbf{B}}^{(t-1)} \neq \delta \cdot \mathbb{PRICE}_{\mathbf{B}}^{(t)}$, the users buying strategies satisfy $mb_{i\,\mathbf{B}}^{(t)} = m\mathbf{B}_U^{(t+1)} = \frac{S^{(t+1)}}{Ser2Fees_{\mathbf{B}}^{(t+1)} \cdot \mathbb{PRICE}_{\mathbf{B}}^{(t+1)}} = S^{(t+1)} \cdot \frac{\delta}{\mathbb{PRICE}_{\mathbf{B}}^{(t)}} \cdot \kappa_S$. Moreover, the validators' selling strategies satisfy $s_{j\,\mathbf{B}}^{(t)} = R_{\mathbf{B}}^{(t)} \cdot r(1/n)$, since the validators selling constraint for tokens of type $\mathbf{B}$ must be strict, and thus $\mathbf{B}_i^{(t)} = 0$ for all t. Therefore the no buyback condition for tokens of type $\mathbf{B}$ is equivalent to $n \cdot R_{\mathbf{B}}^{(t)} \cdot r(1/n) \leq S^{(t+1)} \cdot \frac{\delta}{\mathbb{PRICE}_{\mathbf{B}}^{(t)}} \cdot \kappa_S$.
2. We also require no buyback for the tokens of type $\mathbf{A}$, which means that $s_{j\,\mathbf{A}}^{(t)} = 0$ because, in any generic symmetric equilibrium users never hold (and thus never buy) tokens of type $\mathbf{A}$: If users hold tokens $\mathbf{A}$, the second part Lemma 20 implies $\mathbb{PRICE}_{\mathbf{A}}^{(t-1)} = \delta \cdot \mathbb{PRICE}_{\mathbf{A}}^{(t)}$, thus contradicting (31) in Definition 21. Hence, $\mathbf{A}_V^{(t+1)} = \mathbf{A}_V^{(t)} + R_{\mathbf{A}}^{(t)} \cdot r(1/n) + b_{j\,\mathbf{A}}^{(t)}$ with $b_{j\,\mathbf{A}}^{(t)} \geq 0$.

This completes the proof. ◀

4.3 Stable price mechanism in two-token model

We next describe a simple mechanism which implements stable prices for token $\mathbf{B}$, while the equilibrium price of token $\mathbf{A}$ tends to grow but is affected by the shocks. The system provides rewards with tokens $\mathbf{B}$ only. At equilibrium, validators keep their tokens $\mathbf{A}$ as they guarantees access to rewards, and sell only tokens $\mathbf{B}$. More in detail:

1. In order to have stable prices for $\mathbf{B}$ and no buyback, we set $R_{\mathbf{B}}^{(t)}$ such that $R_{\mathbf{B}}^{(t)} \cdot \mathbb{PRICE}_{\mathbf{B}}^{(t)} = S^{(t+1)} \cdot L$, where $L = \delta \cdot \frac{\kappa_S}{n \cdot r(1/n)}$ is the constant given by Theorem 24 that makes the no buyback condition hold tightly. Furthermore, we set $R_{\mathbf{A}}^{(t)} = 0$.

2. The (equilibrium) buying strategies for token $\mathbf{A}$ are "no buy and no sell", that is, $\mathbf{A}_V^{(t)} = \mathbf{A}_V^{(0)}$, for a suitable $\mathbf{A}_V^{(0)}$ specified below. From the equation of the prices (29), we obtain (see full version [15] for details)

$$\mathbb{PRICE}_{\mathbf{A}}^{(t)} = \frac{\mathbb{PRICE}_{\mathbf{A}}^{(t-1)}}{\delta} - S^{(t+1)} \cdot \frac{L}{\mathbf{A}_V^{(0)}} \frac{n-1}{n^2} \ , \tag{32}$$

which implies $\mathbb{PRICE}_{\mathbf{A}}^{(t)} < \frac{\mathbb{PRICE}_{\mathbf{A}}^{(0)}}{\delta^t}$.

3. From the previous equation, the discounted utility of validators, if deviating by selling all tokens $\mathbf{A}$ at some round τ, is at most $\delta^\tau \cdot \mathbf{A}_V^{(0)} \cdot \mathbb{PRICE}_{\mathbf{A}}^{(\tau)} < \mathbf{A}_V^{(0)} \cdot \mathbb{PRICE}_{\mathbf{A}}^{(0)}$, while the discounted utility for the suggested strategies (equilibrium) equals

$$\sum_{t=0}^{\infty} \delta^t \cdot (R_{\mathbf{B}}^{(t)} \cdot \mathbb{PRICE}_{\mathbf{B}}^{(t)} - v) = \sum_{t=0}^{\infty} \delta^t \cdot (S^{(t+1)} \cdot L) - \frac{v}{1-\delta} \ .$$

4. In order to have feasible (nonnegative) prices, we need to set the initial payments and token holdings sufficiently high.

▶ **Theorem 25.** *For any service value satisfying Assumption 1 and any reward and service sharing schemes satisfying Assumption 2, the strategies above are a symmetric equilibrium with stable prices for token* $\mathbf{B}$ *used by users for getting the service.*

Proof. We follow a similar argument as in [11, Remark 2 (existence)] based on the following three conditions. First, the discounted utilities of all players (users and validators) assume finite values (since their instantaneous utilities are proportional to $S^{(t)}$). Second, these strategies at equilibrium are in some bounded interval $[\underline{\theta}, \overline{\theta}]$. For users, this is evident since they buy tokens proportionally to $S^{(t+1)}$ and from Assumption 1. For the validators, they sell all $R_{\mathbf{B}}^{(t)} = O(S^{(t)})$ tokens $\mathbf{B}$, they never sell or buy tokens $\mathbf{A}$. Third, the utilities of the players are concave due to Assumption 2. Finally, Theorem 4.5 in [26] yields the desired result (concavity implies that the necessary equilibrium conditions maximize a round-t decomposition of the utility (see full version [15]), while Theorem 4.5 in [26] ensures the maximum for the decomposition indeed maximizes the corresponding player utilities, thus an equilibrium). ◀

4.3.1 Stable price mechanism implementation

At each round t, the system only needs to estimate the next round service value $S^{(t+1)}$, and thus a simpler "one step lookahead" oracle suffices, as opposed to the single-token case. Then, we have the following simple mechanism:

1. Users use (spend) $\mathbf{B}$ tokens proportionally to $S^{(t)}$ and buy new $\mathbf{B}$ tokens proportionally to the next round service value $S^{(t+1)}$.
2. Validators' rewards consist of only tokens $\mathbf{B}$ proportionally to the next round service value $S^{(t+1)}$.
3. Validators sell all these $\mathbf{B}$ tokens that they get as rewards, and keep all $\mathbf{A}$ tokens they had from the beginning.

It is worth pointing out that in our two-token model, users *can* also buy tokens $\mathbf{A}$, and validators *can* sell them as well. However, in the equilibria induced by the proposed mechanisms, it is not convenient for either party to do so.

▶ Remark 26. We note that the equilibria described above are closely related to certain proposed implementations of IOTA ([23]). As in our proposed reward scheme and equilibria, rewards in the IOTA system are uniquely provided in the form of Mana tokens (= token $\mathbf{B}$),

while staking is conducted via IOTA tokens (= token **A**). Furthermore, only Mana tokens can be traded to "[...] individuals that in some sense are external to the system" [23]. In this context, our equilibria demonstrate that such designs, which forbid trading governance tokens, may still be sustainable. Specifically, Theorem 25 says that there exist equilibria where, even if trading governance tokens indirectly via some off-chain/secondary market became possible, players would not find it advantageous to do so.

5 Conclusions and research directions

We studied the problem of token monetary policy in blockchain systems viz-à-viz a set of desiderata that are crucial for the security and sustainability of blockchain systems. Our analysis lead to our proposal of a new general tool for tokenomics policy, quantitative rewarding, which judiciously adjusts the rewards given to validators with the objective of achieving, at equilibrium, desirable paths for the token price, e.g., stable prices. In the light of this, we investigated how the long-term equilibria between users, validators and token flows are different for blockchains with a single token (used for transaction fees and staking) and two separate tokens. An important finding of this analysis, is that the two-token model affords additional flexibility that can handle a broader variation of service values at equilibrium. Specifically, on the issue of implementation of monetary policies satisfying all our desiderata, the results for single token tokenomics in Section 3 highlight the need for some "global" information about future fluctuations in the service value, making such policies difficult to realize (namely, in order to avoid buybacks and price fluctuations, oracles that predict future shocks are needed to avoid reward explosion). In two-token systems, instead, the policy only needs to know the near future service value to control prices and enforce all our desiderata, a more tractable objective. In summary, in order to facilitate all our desiderata for a monetary policy, a suitable proxy for the service value $S^{(t)}$ is essential, and for two-token systems the presence of such proxy can be put to use much more effectively via our proposed QR mechanism. Future work could delve into assessing various possible proxies and their on-chain implementation as well as the incentive compatibility for users and validators that now will have the additional leverage of observing any of the oracle computations and data dependencies that may be taking place publicly on-chain and could be influenced by their actions.

References

1 Yannis Bakos and Hanna Halaburda. The role of cryptographic tokens and ICOs in fostering platform adoption. *CESifo Working Paper*, 2019. URL: `https://ideas.repec.org/p/ces/ceswps/_7752.html`.

2 Yannis Bakos and Hanna Halaburda. Overcoming the coordination problem in new marketplaces via cryptographic tokens. *Information Systems Research*, 33(4):1368–1385, 2022. `doi:10.1287/ISRE.2022.1157`.

3 Jeff Burdges, Alfonso Cevallos, Peter Czaban, Rob Habermeier, Syed Hosseini, Fabio Lama, Handan Kilinc Alper, Ximin Luo, Fatemeh Shirazi, Alistair Stewart, and Gavin Wood. Overview of polkadot and its design considerations, 2020. `arXiv:2005.13456`.

4 Tarun Chitra. Competitive Equilibria Between Staking and On-chain Lending. *Cryptoeconomic Systems*, 0(1), 2021. URL: `https://cryptoeconomicsystems.pubpub.org/pub/chitra-staking-lending-equilibria`.

5 Lin William Cong, Zhiheng He, and Ke Tang. The tokenomics of staking. *Preprint, submitted April*, 6, 2022.

6 Lin William Cong, Ye Li, and Neng Wang. Tokenomics: Dynamic adoption and valuation. *The Review of Financial Studies*, 34(3):1105–1155, 2021.

7 Lin William Cong, Ye Li, and Neng Wang. Token-based platform finance. *Journal of Financial Economics*, 144(3):972–991, 2022.

8 Cosmos SDK. Gas and Fees. `https://docs.cosmos.network/main/learn/beginner/gas-fees`.

9 Nicola Dimitri. The economic value of dual-token blockchains. *Mathematics*, 11(17):3757, 2023. `doi:10.3390/math11173757`.

10 Ester Félez-Viñas, Sean Foley, Jonathan R Karlsen, and Jiri Svec. Better than bitcoin? can cryptocurrencies beat inflation? *Available at SSNR*, 2021. URL: `https://dx.doi.org/10.2139/ssrn.3970810`.

11 Samuel Häfner. Optimal decentralization and service provision on a blockchain platform with market frictions, September 2023. URL: `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3954773`.

12 Axie Infinity. Axie infinity whitepaper, 2021. NEO & GAS. URL: `https://whitepaper.axieinfinity.com/axs`.

13 Urban J Jermann. A macro finance model for proof-of-stake ethereum. *Available at SSRN 4335835*, 2023.

14 Kose John, Thomas J Rivera, and Fahad Saleh. Equilibrium staking levels in a proof-of-stake blockchain. *Available at SSRN*, 3965599, 2021.

15 Aggelos Kiayias, Philip Lazos, and Paolo Penna. Single-token vs two-token blockchain tokenomics. *arXiv preprint arXiv:2403.15429*, 2024. `doi:10.48550/arXiv.2403.15429`.

16 Aggelos Kiayias, Philip Lazos, and Jan Christoph Schlegel. Would Friedman Burn your Tokens? *Financial Cryptography and Data Security (FC 2024)*, 2024. Also available as arXiv preprint arXiv:2306.17025. `doi:10.48550/arXiv.2306.17025`.

17 Ricardo Lagos and Randall Wright. A unified framework for monetary theory and policy analysis. *Journal of political Economy*, 113(3):463–484, 2005.

18 Jiasun Li and William Mann. Digital tokens and platform building. *The Review of Financial Studies*, 38(7):1921–1954, 2025.

19 neo.org. NEO & GAS. URL: `https://neo.org/neogas#tokens`.

20 Christina Ovezik, Dimitris Karakostas, Mary Milad, Aggelos Kiayias, and Daniel W. Woods. Sok: Measuring blockchain decentralization, 2025. `doi:10.48550/arXiv.2501.18279`.

21 Emiliano S Pagnotta. Decentralizing money: Bitcoin prices and blockchain security. *The Review of Financial Studies*, 35(2):866–907, 2022.

22 Julien Prat, Vincent Danos, and Stefania Marcassa. Fundamental Pricing of Utility Tokens. Working Papers hal-03096267, HAL, 2021. URL: `https://ideas.repec.org/p/hal/wpaper/hal-03096267.html`.

23 Olivia Saa, Andrew Cullen, and Luigi Vigneri. Iota 2.0 incentives and tokenomics whitepaper, 2023. URL: `https://www.iota.org/foundation/research-papers`.

24 Fahad Saleh. Blockchain without waste: Proof-of-stake. *The Review of financial studies*, 34(3):1156–1190, 2021.

25 Linda Schilling and Harald Uhlig. Some simple bitcoin economics. *Journal of Monetary Economics*, 106:16–26, 2019.

26 N Stokey. *Recursive Methods in Economic Dynamics*. Harvard University Press, 1989.

27 The DFINITY Team. The internet computer for geeks. Cryptology ePrint Archive, Paper 2022/087, 2022. URL: `https://eprint.iacr.org/2022/087`.

Transaction Fee Market Design for Parallel Execution

Bahar Acilan[1]
ETH Zürich, Switzerland

Andrei Constantinescu
ETH Zürich, Switzerland

Lioba Heimbach
ETH Zürich, Switzerland

Roger Wattenhofer
ETH Zürich, Switzerland

Abstract

Given the low throughput of blockchains like Bitcoin and Ethereum, scalability – the ability to process an increasing number of transactions – has become a central focus of blockchain research. One promising approach is the parallelization of transaction execution across multiple threads. However, achieving efficient parallelization requires a redesign of the incentive structure within the fee market. Currently, the fee market does not differentiate between transactions that access multiple high-demand storage keys (i.e., unique identifiers for individual data entries) versus a single low-demand one, as long as they require the same computational effort. Addressing this discrepancy is crucial for enabling more effective parallel execution.

In this work, we aim to bridge the gap between the current fee market and the need for parallel execution by exploring alternative fee market designs. To this end, we propose a framework consisting of two key components: a *Gas Computation Mechanism (GCM)*, which quantifies the load a transaction places on the network in terms of parallelization and computation, measured in *units of gas*, and a *Transaction Fee Mechanism (TFM)*, which assigns a price to each unit of gas. We additionally introduce a set of desirable properties for a GCM, propose several candidate mechanisms, and evaluate them against these criteria. Our analysis highlights two strong candidates: the *weighted area* GCM, which integrates smoothly with existing TFMs such as EIP-1559 and satisfies a broad subset of the outlined properties, and the *time-proportional makespan* GCM, which assigns gas costs based on the context of the entire block's schedule and, through this dependence on the overall execution outcome, captures the dynamics of parallel execution more accurately.

2012 ACM Subject Classification Applied computing → Economics

Keywords and phrases blockchain, transaction fee mechanism, parallel execution

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.23

Related Version *Full Version*: https://arxiv.org/abs/2502.11964 [1]

Funding We thank the Robust Incentives Group at the Ethereum Foundation for supporting this work through a grant.

Acknowledgements We thank Boyan Zhou for his work on a student project, supervised by the last three authors, which helped shape this work.

[1] The authors of this work are listed alphabetically.

© Bahar Acilan, Andrei Constantinescu, Lioba Heimbach, and Roger Wattenhofer;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Scalability, the ability to process more transactions efficiently, has become a central focus in blockchain research, especially given the low throughput of many existing networks. Ethereum, for example, is constrained by its single-threaded execution model, limiting transaction throughput. One promising way to enhance scalability is by parallelizing transaction execution across multiple threads, taking advantage of the multi-core processors common in modern hardware. However, achieving the full efficiency gains of parallel execution requires rethinking the fee market design to better account for *storage key* contention and scheduling constraints. A storage key is a unique identifier (like an address label) for a specific data item in storage.

A *transaction fee mechanism (TFM)* is a core component of any blockchain protocol, determining which pending transactions should be processed and what users must pay for the privilege of having their transactions executed. Traditional fee mechanisms, like Bitcoin's first-price auction, involve users submitting bids with their transactions, and the transactions with the highest bids per computation are included in the next block. Ethereum initially used a similar purely first-price auction-based model but switched to the more sophisticated EIP-1559 mechanism in 2021, which introduces a fluctuating base fee based on network demand, aiming to improve incentive compatibility and reduce price volatility [12]. Most of the existing literature on transaction fee mechanisms focuses on settings where transactions are executed sequentially and therefore does not account for storage key contention, which is crucial in parallel execution environments.

Thus, viewed in isolation, these traditional TFMs are ill-suited to the complexities introduced by parallel transaction execution. They price transactions solely based on their computational cost, without distinguishing between those that access a single storage key and those that interact with multiple, potentially contested storage keys. This pricing model works in a single-threaded environment but fails to capture the nuances of parallel execution, where transactions may impose vastly different constraints on storage key scheduling. Transactions that touch multiple high-contention storage keys can introduce bottlenecks, while those interacting with isolated storage keys are far easier to schedule efficiently.

Ethereum has yet to adopt parallel execution, but several blockchains, such as Solana, Aptos, and Sui, already employ parallel transaction processing [5, 43, 57, 60, 46]. However, many of these networks have yet to implement fee models that fully account for the challenges of parallel execution. Sui and Solana have introduced fee markets tailored to parallelization, but these mechanisms require users to engage in first-price auctions for congested storage keys [40, 44]. As a result, these fee markets demand a high level of sophistication from users to effectively optimize their fee settings and are also not incentive-compatible. The requirements for an effective fee market that is suitable for parallel execution and the design of such a market have so far remained unresolved.

Our Contributions

In this work, we aim to bridge this gap by outlining the requirements for such a fee market and evaluating possible candidates. We outline our main contributions below:

- We introduce a framework with two main components: a *Gas Computation Mechanism (GCM)*, which measures the load a transaction imposes on the network in terms of both parallel execution and computation, expressed in *units of gas*, and a Transaction Fee Mechanism (TFM), which determines the cost associated with each unit of gas.
- We introduce a list of desirable properties for a GCM and evaluate against them a set of mechanisms that we propose.

- Our analysis identifies two promising candidates. The weighted area GCM achieves a large subset of the outlined properties and, importantly, can be seamlessly integrated with existing TFMs, inheriting their properties. Complementing it, the time-proportional makespan GCM is designed to price execution costs in proportion to the total computational load of a block, allowing more accurate resource pricing at the block level.

2 Model

We consider a universe consisting of several stateful *storage keys* (e.g., user accounts, storage addresses of smart contracts). Each storage key can be thought of as a system global variable. We write $\mathcal{K}$ for the set of storage keys. For analysis purposes, we assume that $\mathcal{K}$ is infinite. A *transaction* is a sequence of elementary instructions performing computation and interacting with the storage keys. Some of these operations *access* a given target storage key (e.g., read its value, write to it). For simplicity, we assume that the following are known in advance and supplied with the transaction:

- The non-empty set of storage keys $K \subseteq \mathcal{K}$ the transaction accesses, or an overestimate.[2]
- The total time $t > 0$ it takes to execute the transaction on a single thread.[3]

Transactions execute concurrently, but *atomically*, meaning that the overall effect of executing a batch of transactions should also be achievable by a sequential, single-core execution. For simplicity, we restrict to *concurrent schedules* following a *simple lock-based execution policy*: each storage key has a lock associated with it; whenever a thread wants to execute a transaction, it first locks all required storage keys, then executes the transaction, and then releases the locks. We assume that acquiring (and similarly releasing) the required locks happens simultaneously and takes no additional time. These simplifications have a desirable side-effect: t and K for each transaction now uniquely determine the set of admissible concurrent schedules, allowing us to ignore other details about the transactions:

▶ **Definition 1** (Transaction). *A transaction tx is specified through a tuple (t, K), where $t > 0$ denotes the time required to execute the transaction and $K \neq \varnothing$ represents the set of storage keys the transaction demands, which are locked for the duration of the transaction. We will sometimes write $t(tx)$ and $K(tx)$ for t and K respectively.*

To give the tuple associated with a transaction, we will write $tx \simeq (t, R)$. Note that different transactions might have the same associated tuple. We will also write $tx_1 \simeq tx_2$ to mean that the two transactions have the same associated tuple.
In Figure 1, we illustrate a sample transaction $tx \simeq (3, \{k_2, k_3\})$. Transaction tx thus takes 3 units of time to execute and utilizes storage keys k_2 and k_3. We illustrate this by a rectangle of corresponding length (i.e., time) and width (i.e., storage keys). Throughout, we will illustrate transactions in this manner to aid in visualizing concepts and results. Note that we will always represent transactions as rectangles (i.e., they use consecutive storage keys). In reality, this is, of course, not the case. Importantly, all our results also hold in the more general setting where a transaction can use any subset of storage keys.

[2] In Ethereum, transaction accesses are generally not known in advance. Instead, transactions can execute arbitrary logic (constrained by a maximum amount of computational effort) and their execution depends on the blockchain's state at the time of execution. Ethereum currently supports optional *access lists* that allow transactions to specify their accesses [14]. This optional list could be made mandatory to provide an overestimate of accesses. Additionally, it is worth noting that in other blockchains (e.g., Solana [36] and Sui [35]), an overestimate of accesses is typically known in advance.

[3] In reality, the time to execute a transaction depends on the hardware it is executed on. Thus, time is estimated in Ethereum by assigning each operation a computational effort. Then the sum of the computational effort of a transaction's operation can be seen as a proxy for time.

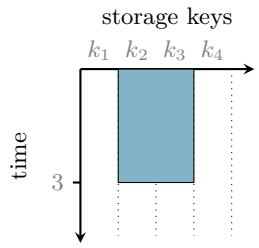


Figure 1 Illustration of a sample transaction $tx \simeq (3, \{k_2, k_3\})$.

A *blockchain* is a sequence of *blocks* comprising bundles of transactions: $B_1, \ldots, B_m$. The system starts in some predetermined initial state. The blocks are executed in order, starting from the oldest (the *genesis* block B_1), successively changing the system's state. For simplicity, we assume no cross-block parallelism, so the execution of a block only starts after the previous block's execution has been completed. However, the execution of transactions inside a block happens concurrently. The system's state after executing B_m is the *current* system state. Users wanting to change the system's state compete for inclusion in the next block B_{m+1} and have to pay a *fee* if successfully included. Desirably, the fee should be higher for more complex transactions and higher during periods of high demand due to limited block space. These requirements are typically decoupled and ensured through different means:

1. A transaction's complexity is quantified in units of *gas*:[4] the more complex a transaction is, the more gas it consumes. Gas encompasses multiple components such as execution, storage space, and data bandwidth. For our purposes, we will only be concerned with *execution gas*.[5] Currently, the execution gas acts as a proxy for the execution time t,[6] but this need not be the case, as we will demonstrate in our paper.
2. The fair competition for block space is ensured through a *transaction fee mechanism (TFM)*: users submit transactions they would like to be included in the next block together with bids of what they would be willing to pay per unit of gas. Importantly, block space is limited, i.e., there is limited space for transactions. The mechanism then determines the set[7] of transactions to be included in the block, together with a price per unit of gas to be paid by each included transaction (potentially not the same for all transactions).[8]

As such, an included transaction consuming g gas units at a price of p per unit of gas will have to pay a fee of $g \cdot p$ (generally in the blockchain's native currency).

To keep the separation of concerns in (1) and (2), we will keep the formula for the fee $g \cdot p$ and instead vary the *gas computation mechanism* used to determine g:

[4] For Ethereum, the unit is called wei.
[5] From this point forward, gas will refer specifically to execution gas.
[6] Throughout we will assume this approximation to the exact.
[7] In practice, blocks are ordered, and transactions often compete for earlier positions (particularly in MEV settings). Following a common simplification in the TFM literature, we model blocks as *sets* of transactions. Accounting for intra-block ordering would require extending our model and constraining it to schedulers that honor the specified ordering. Our framework appears reasonably extensible to such settings, and we leave a detailed treatment to future work.
[8] There are several other components of a TFM, but for the level of detail we need here, this suffices.

▶ **Definition 2** (Gas Computation Mechanism). *A gas computation mechanism (GCM) takes as input a set of transactions T to be included in a block and determines in a deterministic*[9] *manner the amount of gas consumed by each transaction $tx \in T$, written $\mathit{gas}_T(tx)$.*

Currently deployed GCMs associate a fixed, predetermined gas consumption with each transaction, independent of the specific storage keys accessed by the transaction, making them unsuitable for a parallel execution environment. In particular, this is true for Ethereum's *current* GCM:

▶ **Definition 3** (Current GCM). *Given a set of transactions T and a transaction $tx \in T$ with $tx \simeq (t, R)$, the* current *GCM computes the amount of gas used by tx as follows:*

$$\mathit{gas}^C_T(tx) := t.$$

Since this does not depend on T, we often drop the subscript.

Our goal is to ensure that fees accurately reflect the parallelizability of transactions. Therefore, the gas consumption calculated for a transaction should depend on the set of storage keys it accesses and may also be influenced by factors external to the transaction itself (like interactions between transactions).

3 GCM Properties

Next, we outline several desirable properties that a GCM should possess to provide the right incentives for parallelization. These properties should be viewed as a wishlist – as will become clear later on, no single mechanism can satisfy all of them simultaneously.

We begin with two natural monotonicity properties, one for storage keys and one for time. First, a transaction that requires a subset of the storage keys used by another transaction, while taking the same amount of time, should consume no more gas (Property 1 and Figure 2). Similarly, a transaction that requires no more execution time than another, assuming both involve the same set of storage keys, should consume no more gas (Property 2 and Figure 3).

▶ **Property 1** (Storage Key Monotonicity). *Given a set of transactions T and two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$, both not in T, such that $t_1 = t_2$ and $K_1 \subseteq K_2$:*

$$\mathit{gas}_{T\cup\{tx_1\}}(tx_1) \leq \mathit{gas}_{T\cup\{tx_2\}}(tx_2).$$

▶ **Property 2** (Time Monotonicity). *Given a set of transactions T and two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$, both not in T, such that $t_1 \leq t_2$ and $K_1 = K_2$:*

$$\mathit{gas}_{T\cup\{tx_1\}}(tx_1) \leq \mathit{gas}_{T\cup\{tx_2\}}(tx_2).$$

The previous two properties fix one dimension while varying the other. One can also define a seemingly stronger property that allows both to vary, as follows:

▶ **Property 3** (Storage Key-Time Monotonicity). *Given a set of transactions T and two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$, both not in T, such that $t_1 \leq t_2$ and $K_1 \subseteq K_2$:*

$$\mathit{gas}_{T\cup\{tx_1\}}(tx_1) \leq \mathit{gas}_{T\cup\{tx_2\}}(tx_2).$$

[9] We focus solely on deterministic mechanisms. This is common in blockchain fee markets, given the inherent difficulty of accessing true randomness on-chain (though notable approaches exist, such as VRFs). While randomized mechanisms can statistically improve over their deterministic counterparts, their ex-post behavior is often hard to predict, leading to higher fee uncertainty, an outcome generally undesirable in blockchain settings.

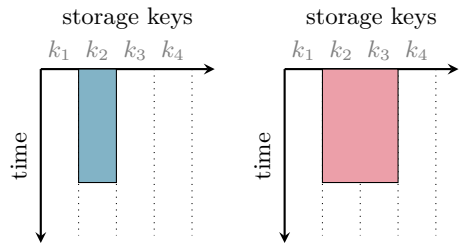


Figure 2 Illustration of Property 1, where $T = \emptyset$, ▭ represents a sample tx_1 and ▭ represents a sample tx_2.

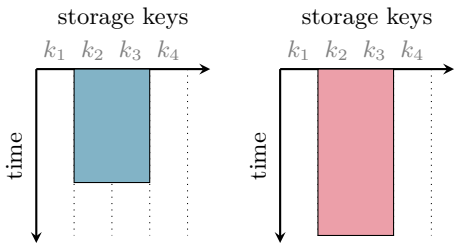


Figure 3 Illustration of Property 2, where $T = \emptyset$, ▭ represents a sample tx_1 and ▭ represents a sample tx_2.

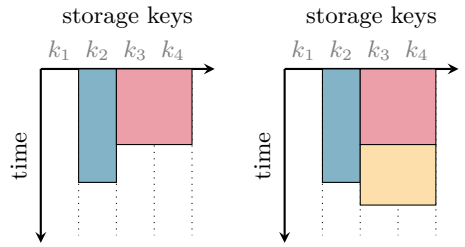


Figure 4 Illustration of Property 4, where $T = \emptyset$, $T_1 = \{tx_1, tx_2\}$, and $T_2 = T_1 \cup \{tx_3\}$. Here, ▭ represents a sample tx_1, ▭ represents a sample tx_2, and ▭ represents a sample tx_3.

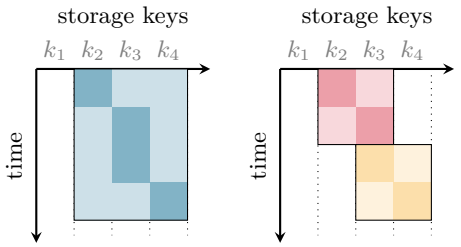


Figure 5 Illustration of Property 5, where $T = \emptyset$, ▭ represents a sample tx_3, while ▭ represents a sample tx_1 and ▭ represents a sample tx_2. The darker shaded areas indicate when a transaction operates on a storage key.

Intuitively, if $tx_1 \lesssim tx_2$, by which we mean $t(tx_1) \leq t(tx_2)$ and $K(tx_1) \subseteq K(tx_2)$, then tx_1 should cost no less than tx_2. However, an attentive reader will observe that the former two are collectively equivalent to the latter (the proof and all subsequent omitted proofs can be found in the full version of this paper [1]):

▶ **Lemma 4.** *Property 3 holds if and only if Properties 1 and 2 hold.*

Let us now take a moment to briefly evaluate why Properties 1–3 are not merely intuitive, but their violation can lead to harmful consequences and misaligned incentives: suppose $tx_1 \lesssim tx_2$ and T is a set of transactions containing neither of the two. If $\mathsf{gas}_{T \cup \{tx_1\}}(tx_1) > \mathsf{gas}_{T \cup \{tx_2\}}(tx_2)$, a user intending to submit tx_1 might instead pad tx_1 with unnecessary instructions and declare a larger access list to decrease the gas consumption. This would paradoxically reduce the gas usage and, assuming a reasonable TFM is used to compute transaction fees, also lower the transaction fee.

For any of the three properties above, we say that the property is *strictly* satisfied if for $tx_1 \not\simeq tx_2$ the conclusion inequality holds strictly. Note that properties holding strictly are even more desirable with respect to the reasoning above: replacing a transaction with a "larger" one is then not only no better but actively worse. Unsurprisingly, Lemma 4 also holds for the strict versions of the properties:

▶ **Lemma 5.** *Property 3 holds strictly if and only if Properties 1 and 2 hold strictly.*

Next, we introduce another desirable monotonicity property, this time with a different emphasis: if two sets of transactions satisfy $T_1 \subseteq T_2$, the transactions in T_1 should collectively consume no more gas than those in T_2 (Property 4 and Figure 4). To formalize this, given a set of transactions T and a subset $T' \subseteq T$, write $\mathsf{gas}_T(T') := \sum_{tx' \in T'} \mathsf{gas}_T(tx')$ for the total gas consumed by the transactions in T' when included in the set T that constitutes a block.

▶ **Property 4** (Set Inclusion). *Given a set of transactions T and two sets of transactions $T_1 \subseteq T_2$, disjoint from T:*

$$\mathit{gas}_{T \cup T_1}(T_1) \leq \mathit{gas}_{T \cup T_2}(T_2).$$

To understand why this property is desirable, consider a GCM for which it does not hold. Then, there must be a scenario where a set of transactions can reduce their total gas consumption by adding additional transactions. This situation could be exploited through collusion by the users originating these transactions. Importantly, such a possibility is undesirable, as it would primarily benefit sophisticated users capable of orchestrating such arrangements.

Similarly to before, we say that Property 4 holds *strictly* if for $T_1 \neq T_2$ the inequality in the conclusion holds strictly, which is desirable for reasons similar to those discussed above.

We now move on to more complex properties. Given two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$, a transaction tx_3 is their *sequential composition* (or, more simply, *concatenation*), if it executes the steps of tx_1 followed by the steps of tx_2. Note that, in this case $tx_3 \simeq (t_1 + t_2, K_1 \cup K_2)$. Our next property states that the concatenation of two transactions should consume no less gas than submitting them individually (Property 5 and Figure 5). The two individual transactions perform the same actions as their concatenation, but not atomically, with no guarantee of their relative ordering or control over what happens between them. Naturally, enforcing atomicity and ordering limits the set of admissible concurrent schedules and requires storage keys to remain locked over longer continuous timespans. In particular, tx_3 requires the storage keys in $K_1 \cup K_2$ to be locked over a continuous span of $t_1 + t_2$ units, while tx_1 and tx_2 submitted individually only require exclusive access to storage keys in K_1 for t_1 units and to storage keys in K_2 for t_2 units. Hence, the "bigger" transaction is at least as hard to schedule as its two constituent "parts" and should hence consume no less gas.

▶ **Property 5** (Transaction Bundling). *Consider a set of transactions T and three transactions $tx_1, tx_2, tx_3 \notin T$ such that tx_3 is the concatenation of tx_1 and tx_2, then:*

$$\mathit{gas}_{T \cup \{tx_1, tx_2\}}(tx_1) + \mathit{gas}_{T \cup \{tx_1, tx_2\}}(tx_2) \leq \mathit{gas}_{T \cup \{tx_3\}}(tx_3).$$

Let us again consider the risks of having a GCM that does not satisfy the previous property. In such a scenario, a group of users could collude to collectively consume less gas by combining their transactions into a single transaction rather than processing them individually. This outcome would be undesirable, particularly because it disproportionately benefits sophisticated users, as we outlined before. Note that this could even be the case for a single user wanting to submit multiple transactions.

We say that Property 5 holds *strictly* if the inequality in the conclusion holds strictly, which is again more desirable than the basic version of the property.

Next, we would like to formalize the intuitive idea that a transaction's gas consumption fairly reflects its impact on the execution time. To do so, we first need to formalize scheduling more precisely. Let $n \geq 2$ be the number of available threads.

▶ **Definition 6.** *A* scheduler *(for n threads) takes as input a set of transactions T, and outputs a* concurrent schedule *using at most n threads to execute all transactions in T. This schedule specifies the operations each thread should perform and the order in which they should be performed.*

The following conditions should hold for any generated schedule:

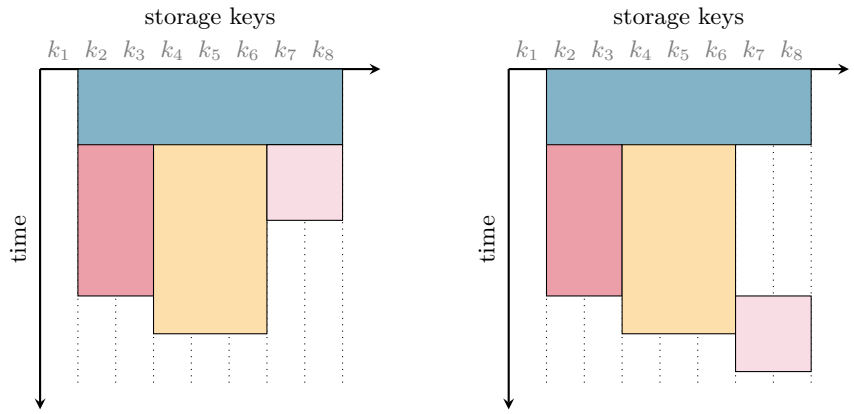


Figure 6 Illustration of optimal schedules for a set of four transactions: $tx_1 \simeq (2, \{k_2, \ldots, k_8\})$ (shown in), $tx_2 \simeq (4, \{k_2, k_3\})$ (shown in), $tx_3 \simeq (5, \{k_4, k_5, k_6\})$ (shown in), and $tx_4 \simeq (2, \{k_7, k_8\})$ (shown in). In the left plot, we show the optimal schedule for $n = 3$, and in the right plot for $n = 2$. Notice how for $n = 2$, we cannot schedule the three transactions tx_2, tx_3, and tx_4 to be executed in parallel even though they access pairwise-disjoint sets of storage keys.

- Each transaction is assigned to a single thread;
- A thread can work on only one transaction at a time;
- There is no preemption: once a thread starts executing a transaction, it completes the transaction without context switching;
- Transactions with overlapping storage key access sets cannot be executed in parallel: one must finish before the other can begin.

Note that some of these conditions are natural for scheduling in general, while others arise from us assuming the *simple lock-based execution policy*.[10] For our purposes, we are not concerned with which thread executes which transaction, but only that no more than n transactions ever execute simultaneously (so we can draw concurrent schedules as in Figure 6, which illustrates the concept). To determine gas consumption, our notation will need to capture even less – given a scheduler s (for n threads), we write $v^s(T)$ for the *makespan* of the schedule produced by s for the set of transactions T (i.e., the time required to execute the schedule in parallel using n threads). It is instructive to read the paper having in mind as a prime example s being the *optimal scheduler* (for n threads), which returns a schedule for n threads that minimizes the makespan. However, implementing such a scheduler is computationally prohibitive in practice, so greedy heuristics are typically used instead. To make our results apply even to non-optimal schedulers, we assume a set of minimal, reasonable properties for the scheduler:

(S1) *Monotonicity in T:* for any $T \subseteq T'$, we have $v^s(T) \leq v^s(T')$. Intuitively, scheduling no fewer transactions takes no less time.

(S2) *Monotonicity under bundling:* consider any set of transactions T and three transactions $tx_1, tx_2, tx_3 \notin T$, such that tx_3 is the concatenation of tx_1 and tx_2, then we have $v^s(T \cup \{tx_1, tx_2\}) \leq v^s(T \cup \{tx_3\})$. Intuitively, replacing two transactions by their concatenation makes scheduling no easier.

[10] Using locks is one way to enforce this policy, but in our case – where the contents of a block are known before execution commences – it can also be achieved without locks, as long as the execution environment ensures that threads strictly follow a pre-determined schedule. We chose this name because it is largely suggestive of the intended semantics.

(S3) *Monotonicity in t and K:* consider any set of transactions T and two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$, both not in T, such that $t_1 \leq t_2$ and $K_1 \subseteq K_2$, i.e., $tx_1 \lesssim tx_2$, then we have $v(T \cup \{tx_1\}) \leq v(T \cup \{tx_2\})$. Intuitively, "larger" transactions are no easier to schedule.

(S4) *Empty set:* $v(\varnothing) = 0$.

The optimal scheduler can be easily seen to satisfy these properties.[11] *Our positive results will apply to any scheduler satisfying the properties, while our negative results will be for the optimal scheduler itself.* Henceforth, we drop the s superscript for brevity and follow this convention for who s is.

An attentive reader may notice that (S1)–(S3) partially resemble Properties 1–5 for GCMs. This resemblance is not coincidental, but it is important to emphasize that they address different aspects: the former are properties of the scheduler, while the latter are properties of the GCM. A GCM can, in fact, satisfy Properties 1–5 without depending on the scheduler at all (e.g., the current mechanism). Conversely, for mechanisms defined in terms of the scheduler, (S1)–(S3) play a crucial role in establishing their properties, including Properties 1–5.

Armed as such, we now return to our latest goal: formalizing the idea that a transaction's gas consumption should fairly reflect its impact on execution time. We do this as follows:

▶ **Property 6** (Scheduling Monotonicity). *Given a set of transactions T and two transactions tx_1 and tx_2, both not in T, such that $v(T \cup \{tx_1\}) < v(T \cup \{tx_2\})$:*[12]

$$gas_{T \cup \{tx_1\}}(tx_1) \leq gas_{T \cup \{tx_2\}}(tx_2).$$

Intuitively, transactions with higher marginal contributions to the execution time should consume no less gas.[13] As usual, we define a *strict* version of this property, where the latter inequality also becomes strict (i.e., higher marginal contributions in execution time imply higher gas consumption). One might be tempted to believe that Scheduling Monotonicity implies Storage Key-Time Monotonicity. However, proving this requires a non-strict inequality in the premise $v(T \cup \{tx_1\}) < v(T \cup \{tx_2\})$.

There is a second way in which gas consumption should adequately indicate the effort required to execute transactions (Property 7). Specifically, the gas consumption of all transactions in a block should *collectively* account for the total time needed to execute the block. This can be seen as properly tracking the time consumed by the execution environment to execute the block.

▶ **Property 7** (Efficiency). *Consider a set of transactions T and recall the definition $gas_T(T) = \sum_{tx \in T} gas_T(tx)$, then:*

$$gas_T(T) = v(T).$$

[11] All but the second straightforward, while for the second, any admissible schedule for $T \cup \{tx_3\}$ can be turned into a same-makespan admissible schedule for $T \cup \{tx_1, tx_2\}$ by replacing tx_3 by tx_1 followed immediately by tx_2.

[12] Interestingly, unlike our earlier properties, writing $v(T \cup \{tx_1\}) \leq v(T \cup \{tx_2\})$ here may be too strong, as applying the property twice would then imply that if $v(T \cup \{tx_1\}) = v(T \cup \{tx_2\})$, then $\mathsf{gas}_{T \cup \{tx_1\}}(tx_1) = \mathsf{gas}_{T \cup \{tx_2\}}(tx_2)$, which is not necessarily desirable. Writing $v(T \cup \{tx_1\}) \leq v(T \cup \{tx_2\})$ would also unintentionally make the strict and non-strict versions of the property incomparable.

[13] Technically, what we wrote above in Property 6 are not marginal contributions, but can be made to be by subtracting $v(T)$ from both sides of the inequality in the antecedent.

Next, we introduce two practical properties. The first requires that transaction submitters be able to estimate a transaction's gas consumption in advance. We formalize this by requiring that $\mathsf{gas}_T(tx)$ does not depend on T (Property 8).

▶ **Property 8** (Easy Gas Estimation). *Given two sets of transactions T_1 and T_2 and a transaction tx belonging to neither T_1 nor T_2:*

$$\mathit{gas}_{T_1 \cup \{tx\}}(tx) = \mathit{gas}_{T_2 \cup \{tx\}}(tx).$$

This property ensures a good user experience by making it easy for users to estimate gas usage. If the gas consumption of a transaction depended on the remaining transactions in the block, this estimation would become significantly more complex. In general, we aim to keep gas estimation straightforward to avoid giving an advantage to more sophisticated users. Additionally, as we will see later, a GCM satisfying this property can be seamlessly composed with existing TFMs, retaining their desirable properties (see Section 6). Sadly, as one might have already guessed, Easy Gas Estimation is incompatible with Scheduling Monotonicity (except for *constant* mechanisms, i.e., GCMs that return a constant independent of T and tx), and also incompatible with Efficiency:

▶ **Theorem 7.** *Easy Gas Estimation (Property 8) is:*

1. *Incompatible with Scheduling Monotonicity (Property 6) unless using a constant GCM.*[14]
2. *Incompatible with Efficiency (Property 7).*

Proof. Consider a mechanism M with Easy Gas Estimation; i.e., $\mathsf{gas}^M(tx)$ is meaningful without a subscript for the set of transactions T in the block. Then:

1. Assume that M has Scheduling Monotonicity; we will show that M is a constant mechanism.
   Call two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$ *incomparable* if neither $K_1 \subseteq K_2$ nor $K_2 \subseteq K_1$. As a first step, we will show that if tx_1 and tx_2 are incomparable, then $\mathsf{gas}^M(tx_1) = \mathsf{gas}^M(tx_2)$. To show this, assume that $K_1 \not\subseteq K_2$. We will show that then $\mathsf{gas}^M(tx_1) \geq \mathsf{gas}^M(tx_2)$. Exchanging the roles of tx_1 and tx_2 will then give the conclusion. Let $k \in K_1 \setminus K_2$ and tx_3 be another transaction such that $tx_3 \simeq (t_3, \{k\})$ for some $t_3 > t_2$, and define $T = \{tx_3\}$. Then, for any number of threads $n \geq 2$ and any scheduler that is optimal for two transactions, we have $v(T \cup \{tx_1\}) = t_1 + t_3 > t_3 = v(T \cup \{tx_2\})$. By Scheduling Monotonicity, this implies that $\mathsf{gas}^M(tx_1) \geq \mathsf{gas}^M(tx_2)$.
   We now know that M associates the same gas consumption to any pair of incomparable transactions. Armed as such, consider two arbitrary transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$. Let k be an arbitrary storage key *not* in $K_1 \cup K_2$ and tx_3 be a transaction such that $tx_3 \simeq (1, \{r\})$. Since K_1 and K_2 are non-empty, one can easily see that tx_3 is incomparable with both tx_1 and tx_2, from which $\mathsf{gas}^M(tx_1) = \mathsf{gas}^M(tx_3) = \mathsf{gas}^M(tx_2)$.
2. Assume for a contradiction that M has Efficiency, then for any transaction $tx \simeq (t, K)$ we have $\mathsf{gas}^M(tx) = \mathsf{gas}^M_\varnothing(\{tx\}) = v(\{tx\}) = t$. Hence, by definition, M is the current mechanism, which can be easily seen to not satisfy Efficiency: assume $n \geq 2$ threads and consider the set of transactions $T = \{tx_1, tx_2\}$ where $tx_1 \simeq (1, \{k_1\})$ and $tx_2 \simeq (1, \{k_2\})$. In this case, $\mathsf{gas}^M(T) = \mathsf{gas}^M(tx_1) + \mathsf{gas}^M(tx_1) = 1 + 1 = 2 \neq 1 = v(T)$. ◀

Our second practical property requires that gas consumption be efficiently computable, i.e., in polynomial time (Property 9). A GCM that does not satisfy this property would be unsuitable for the blockchain context, where gas computation is intended to be a straightforward component.

[14] Constant GCMs satisfy Scheduling Monotonicity, but not Strict Scheduling Monotonicity.

▶ **Property 9** (Poly-time Computable). *There exists a polynomial-time algorithm that takes as input a transaction set T and a transaction tx and outputs $\mathsf{gas}_{T\cup\{tx\}}(tx)$.*

4 GCM Proposals

We now propose multiple GCM designs. Motivated by the incompatibilities in Theorem 7, these fall into two categories:

(C1) Mechanisms with Easy Gas Estimation, in which each transaction's gas consumption is computed in isolation. Such mechanisms tend to be both straightforward and attractive but can achieve neither Scheduling Monotonicity[15] nor Efficiency.

(C2) Mechanisms *without* Easy Gas Estimation, which, given a set of transactions T, rely on $v(T)$ or, more generally, $(v(T'))_{T'\subseteq T'}$ to holistically calculate gas consumptions for the block T, requiring knowledge of the entire block. Instead, these mechanisms aim to achieve Scheduling Monotonicity and/or Efficiency.

4.1 Mechanisms with Easy Gas Estimation

We already introduced the *current* GCM (Definition 3). For the next mechanism, we assume a globally available vector of *positive* weights for the storage keys $(w_k)_{k\in\mathcal{K}}$. For instance, these weights could all be 1. Alternatively, higher-weight storage keys could correspond to storage keys under higher (historical) demand. For our purposes, we only need to assume that the weights are known and do not depend on the block being built. In practice, one could update the weights between blocks to accurately reflect storage key demand (the exact details are not relevant here; see Section 6 for further discussion). Given the weights, we define:

▶ **Definition 8** (Weighted Area GCM). *Given a set of transactions T and a transaction $tx \in T$ with $tx \simeq (t, K)$, the* Weighted Area *GCM computes the amount of gas used by tx as follows:*

$$\mathsf{gas}_T^{WA}(tx) := t \cdot \left(1 + \sum_{k\in K} w_k\right) \tag{1}$$

Since this does not depend on T, we often drop the subscript.

To gain intuition, it is instructive to consider the former case, where all weights are 1, i.e., the *unweighted area* mechanism, in which case Equation (1) becomes $\mathsf{gas}_T^{WA}(tx) = t\cdot(1+|K|)$. This mechanism can also be viewed as the addition of two terms: the *current term*, i.e., the gas consumption of tx under the *current* GCM, namely t, and the *area term*, i.e., the area of a $t \times |K|$ rectangle (which is also how we draw transactions in our diagrams). The *current term* is helpful to ensure no transaction ever costs too little when the weights of all storage keys accessed by a transaction are too close to zero.[16]

The *area term*, on the other hand, which is the main component, intuitively corresponds to "negated throughput." That is, executing the transaction requires holding $|K|$ locks for t time units, during which no other transactions requiring any of those storage keys can

[15] Except for constant GCMs.

[16] This term has a practical motivation. In reality, transactions will be scheduled on a small finite number of threads. Thus, even if they do not access any high-demand storage key, they occupy space in the schedule proportional to their execution time. Note, moreover, that this term could technically be omitted, and our results would still hold.

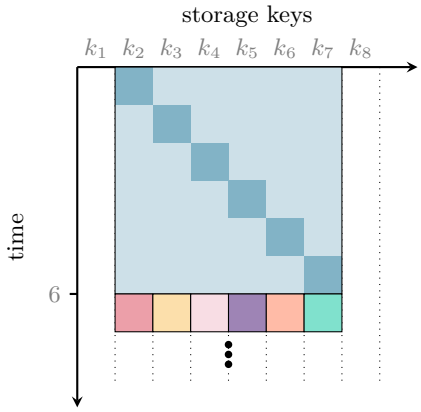


Figure 7 Illustration of a worst-case transaction in terms of parallelizability. Transaction $tx \simeq (6, \{k_2, \ldots, k_7\})$ (shown in ▭) takes 6 units of time to execute and accesses 6 storage keys. However, the transaction only operates on each storage key for one unit of time each (shown in ▭).

execute. In Figure 7, we illustrate this idea. Transaction tx prevents the execution of other transactions across its entire storage key set but only utilizes each storage key for a short period. While the current GCM charges a transaction based solely on its total computation time (i.e., the height of the rectangle), the weighted area GCM also accounts for the storage keys for which it "negated throughput", i.e., the area occupied by the transaction. This occupied area prevents other transactions using the same storage keys from being scheduled in parallel.

Along the same line of reasoning, to further reinforce why this mechanism is a reasonable choice, consider a transaction set T. The sum $\sum_{tx \in T} t(tx) \cdot |K(tx)|$ represents the "total area" of transactions in T. Dividing this term by the number of storage keys used in T provides a lower bound on $v(T)$, serving as a rough proxy that does not require knowledge of T when computing individual gas consumptions. Finally, we note that, even in the case of non-unit weights, the term *weighted area* remains meaningful, as the area term still corresponds to the area of the respective rectangle in our diagrams if we give the column of each storage key $k \in \mathcal{K}$ a width of w_k.

4.2 Mechanisms *without* Easy Gas Estimation

We now switch gears towards mechanisms without Easy Gas Estimation that instead aim to achieve Scheduling Monotonicity and/or Efficiency. Intuitively, for a set of transactions T, such mechanisms should start from $v(T)$ and take each transaction's marginal contribution towards $v(T)$ as its gas consumption. This, however, has to be done carefully, as, e.g., simply looking for each transaction $tx \in T$ at $v(T) - v(T \setminus \{tx\})$ does not help. Note that one can easily construct cases where removing any one transaction from T does not change $v(T)$, so all reported numbers will be 0. However, all these numbers being zero does not mean that no transaction contributes to $v(T)$; it just means that we also have to consider removing multiple transactions to see the differences. This gives us the idea to look at the marginal contribution of each transaction $tx \in T$ when added to each possible subset $S \subseteq T \setminus \{tx\}$, and not just to $S = T \setminus \{tx\}$ as before. Formally, we would like to compute tx's gas consumption as an aggregate of $(v(S \cup \{tx\}) - v(S))_{S \subseteq T \setminus \{tx\}}$. We next present two mechanisms based on this idea: the *Shapley* (Definition 9) and *Banzhaf* (Definition 11) GCMs, inspired by corresponding concepts in cooperative game theory.

▶ **Definition 9** (Shapley GCM). *Given a set of transactions T consisting of $|T| = n$ transactions and a transaction $tx \in T$, the* Shapley *GCM computes the amount of gas used by tx as follows:*

$$\mathit{gas}_T^S(tx) := \frac{1}{n!} \sum_{\sigma} \left[v(P_{tx}^{\sigma} \cup \{tx\}) - v(P_{tx}^{\sigma}) \right] \tag{2}$$

$$= \sum_{S \subseteq T \setminus \{tx\}} \frac{|S|! \cdot (n - |S| - 1)!}{n!} \left[v(S \cup \{tx\}) - v(S) \right]. \tag{3}$$

Here, σ ranges over the $n!$ possible ways to order the transactions in T and P_{tx}^{σ} denotes the set of transactions that precede tx in the order σ. The equality between Equations (2) and (3) follows by counting the number of orders σ such that $P_{tx}^{\sigma} = S$, which there are $|S|! \cdot (n - |S| - 1)!$ of.

Another way to understand the Shapley GCM is through the following probabilistic experiment:

1. Select an ordering σ of the transactions in T uniformly at random.
2. Start with an empty set of transactions and add transactions one by one in the order given by σ.
3. Whenever a transaction tx is added, let $S = P_{tx}^{\sigma}$ be the set of transactions just before adding it and compute tx's *marginal contribution* to the execution time as $v(S \cup \{tx\}) - v(S)$; i.e., by how much did the execution time increase by adding tx to the current set of transactions.
4. The gas consumption of $tx \in T$ is the expectation of its marginal contribution across the experiment.

The reader familiar with cooperative game theory will have already recognized the immediate connection with Shapley values: if we see each transaction $tx \in T$ as a player in a game with valuation function $v : 2^T \to \mathbb{R}$, then $\mathsf{gas}_T^S(tx)$ is precisely the celebrated *Shapley value* of player tx, more traditionally written $\phi_{tx}(v)$. One classical property of Shapley values is that, given $v(\varnothing) = 0$, as is the case for us by (S4), their sum equals the valuation of the *grand coalition* T: $\sum_{tx \in T} \phi_{tx}(v) = v(T)$. The proof is straightforward: for any fixed ordering σ of T, the sum of the marginal contributions of the transactions is a telescoping sum, i.e., except first and last terms, all others appear once positively and once negatively, canceling as a result and leaving us with $v(T) - v(\varnothing) = v(T)$. Since the sum does not depend on σ, the same holds when taking the expectation with respect to σ. This already gives us our first property of the Shapley GCM, namely Efficiency:

▶ **Lemma 10.** *The Shapley GCM satisfies Efficiency (Property 7).*

We postpone studying the Shapley GCM further until introducing our other mechanisms in the section.

A second GCM based on the idea that a transaction's gas consumption should be its marginal contribution to the execution time is the *Banzhaf* mechanism:

▶ **Definition 11** (Banzhaf GCM). *Given a set of transactions T consisting of $|T| = n$ transactions and a transaction $tx \in T$, the* Banzhaf *GCM computes the amount of gas used by tx as follows:*

$$\mathit{gas}_T^B(tx) := \frac{1}{2^{n-1}} \sum_{S \subseteq T \setminus \{tx\}} \left[v(S \cup \{tx\}) - v(S) \right].$$

As for the Shapley mechanism, the Banzhaf mechanism can be understood probabilistically, but this time with a separate experiment for each transaction tx. Sample a subset S of transactions other than tx uniformly at random and compute tx's marginal contribution to the execution time when added to S, namely $v(S \cup \{tx\}) - v(S)$. The gas consumption of tx is then its expected marginal contribution to the execution time. The familiar reader will recognize this as the definition of the *Banzhaf power index* $\beta_{tx}(v)$. For consistency with *Shapley values*, we will instead call these *Banzhaf values*. While more straightforward than the corresponding experiment used in defining Shapley values, the fact that we now have n separate experiments makes the sum of the values no longer well-behaved, losing Efficiency for the Banzhaf mechanism:

▶ **Lemma 12.** *The Banzhaf GCM does* not *satisfy Efficiency (Property 7).*

We conclude this section by introducing two additional reasonable mechanisms *without* Easy Gas Estimation. These mechanisms are notably more straightforward than the Shapley and Banzhaf GCMs, as they avoid computing marginal contributions. However, they have other drawbacks that will become more apparent when we begin studying their normative properties alongside the other mechanisms.

▶ **Definition 13** (Time-Proportional Makespan GCM)**.** *Given a set of transactions T and a transaction $tx \in T$, the* Time-Proportional Makespan (TPM) *GCM computes the amount of gas used by tx as follows:*

$$\mathit{gas}_T^{TPM}(tx) := \frac{t(tx)}{\sum_{tx' \in T} t(tx')} \cdot v(T).$$

▶ **Definition 14** (Equally-Split Makespan GCM)**.** *Given a set of transactions T and a transaction $tx \in T$, the* Equally-Split Makespan (ESM) *GCM computes the amount of gas used by tx as follows:*

$$\mathit{gas}_T^{ESM}(tx) := \frac{v(T)}{|T|}.$$

We also consider the following rather pathological mechanism because of the combination of properties it turns out to satisfy (that none of our other mechanisms do):

▶ **Definition 15** (Exponentially-Split Makespan GCM)**.** *Given a set of transactions T and a transaction $tx \in T$, the* Exponentially-Split Makespan (XSM) *GCM computes the amount of gas used by tx as follows:*

$$\mathit{gas}_T^{XSM}(tx) := \frac{v(T)}{3^{|T|}}.$$

5 Analysis of Our GCMs

In this section, we provide a detailed analysis of the normative properties of our GCMs. Our results are summarized in Table 1.

5.1 Mechanisms with Easy Gas Estimation

In this section, we analyze the current and Weighted Area GCMs. By definition, both satisfy Easy Gas Estimation (Property 8) and are Poly-time Computable (Property 9). Since they satisfy Easy Gas Estimation but are not constant GCMs, Theorem 7 implies that *neither*

Table 1 Comparison of GCMs based on their adherence to the defined properties. $<$ indicates that the mechanism *strictly* satisfies the property, $=$ indicates *trivial* satisfaction (by equality), and $\leq$ indicates satisfaction (not necessarily strict). A "✓" means the property is satisfied; "×" means it is not satisfied. For computational complexity, "v" means the mechanism is as hard to compute as $v(\cdot)$ itself, "$S(v)$" means it is as hard as computing Shapley values for v, and "$B(v)$" means it is as hard as computing Banzhaf values for v.

Property	Current	W. Area	Shapley	Banzhaf	TPM	ESM	XSM
Storage Key Monotonicity (Property 1)	$=$	$<$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$
Time Monotonicity (Property 2)	$<$	$<$	$<$	$<$	$<$	$\leq$	$\leq$
Storage Key-Time Monotonicity (Property 3)	$\leq$	$<$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$
Set Inclusion (Property 4)	$<$	$<$	×	×	$\leq$	$\leq$	×
Transaction Bundling (Property 5)	$=$	$\leq$	×	$\leq$	$\leq$	×	$<$
Scheduling Monotonicity (Property 6)	×	×	×	×	×	$<$	$<$
Efficiency (Property 7)	×	×	✓	×	✓	✓	×
Easy Gas Estimation (Property 8)	✓	✓	×	×	×	×	×
Poly-time Computable (Property 9)	✓	✓	$S(v)$	$B(v)$	v	v	v

satisfies Scheduling Monotonicity (Property 6) nor Efficiency (Property 7). Furthermore, both mechanisms satisfy strict Time Monotonicity (Property 2): increasing the time t of a transaction strictly increases its gas consumption. Similarly, both satisfy strict Set Inclusion (Property 4): since transactions' gas consumptions are strictly positive, a strict superset of a given set of transactions consumes strictly more gas. For the remaining three properties, the two mechanisms behave slightly differently:

Storage Key Monotonicity (Property 1). The current mechanism ignores the set of storage keys K that a transaction accesses, so replacing K with a strict superset of it does not change the transaction's gas consumption. Therefore, the current mechanism satisfies Storage Key Monotonicity *with equality*. On the other hand, the Weighted Area mechanism adds an extra term of $t \cdot w_k > 0$ to the gas consumption for each additional storage key k added to K, so it satisfies strict Storage Key Monotonicity.

Storage Key-Time Monotonicity (Property 3). From the above results on whether our two mechanisms satisfy Storage Key Monotonicity and Time Monotonicity, Lemmas 4 and 5 allow us to conclude that the current mechanism satisfies Storage Key-Time Monotonicity, while the Weighted Area mechanism satisfies it strictly.

Transaction Bundling (Property 5). Concatenating two transactions with times t_1 and t_2 results in a transaction with time t_1+t_2. Since the current mechanism equates gas consumption with time, the bundled transaction has the same gas consumption as the two individual transactions combined. This implies that the current mechanism satisfies Transaction Bundling *with equality*. Finally, the Weighted Area mechanism satisfies Transaction Bundling, as shown below. If we restrict ourselves to bundling transactions with different storage key sets, the property is strictly satisfied.

▶ **Lemma 16.** *The Weighted Area GCM satisfies Transaction Bundling (Property 5). If we only consider bundling transactions with different storage key sets, the property is strictly satisfied.*

Proof. Consider two transactions $tx_1 \simeq (t_1, K_1)$ and $tx_2 \simeq (t_2, K_2)$. Let $tx_3 \simeq (t_1 + t_2, K_1 \cup K_2)$ be a transaction consisting of the concatenation of tx_1 and tx_2. We want to show that $\mathsf{gas}^{WA}(tx_1) + \mathsf{gas}^{WA}(tx_2) \leq \mathsf{gas}^{WA}(tx_3)$. By definition, this amounts to:

$$t_1 \cdot \left(1 + \sum_{k \in K_1} w_k\right) + t_2 \cdot \left(1 + \sum_{k \in K_2} w_k\right) \leq (t_1 + t_2) \cdot \left(1 + \sum_{k \in K_1 \cup K_2} w_k\right) \tag{4}$$

Which is true because $\sum_{k \in K_i} w_k \leq \sum_{k \in K_1 \cup K_2} w_k$ for $i \in \{1, 2\}$. Because the weights are strictly positive, equality occurs if and only if $K_1 = K_1 \cup K_2$ and $K_2 = K_1 \cup K_2$; i.e., $K_1 = K_2$. Hence, the property is satisfied strictly if we restrict bundling transactions to cases where $K_1 \neq K_2$. ◀

5.2 Mechanisms *without* Easy Gas Estimation

In this section, we analyze the Shapley, Banzhaf, TPM, ESM, and XSM GCMs. By definition, all of them require knowledge of T to compute $\mathsf{gas}_T(tx)$, so they do not satisfy Easy Gas Estimation (Property 8). Next, we examine each of the remaining properties individually and analyze whether they hold for our five mechanisms.

Poly-time Computable (Property 9). The TPM, ESM, and XSM GCMs are all Poly-time Computable whenever determining the makespan of a given set of transactions under the chosen scheduler is feasible in polynomial time, i.e., when $v(T)$ can be computed in polynomial time.[17] In fact, the problems are all equally difficult to computing such makespans. In contrast, computing gas consumptions under the Shapley and Banzhaf GCMs hits another hurdle: it is no longer enough to be able to efficiently compute $v(T)$, but instead, one needs to be able to compute the Shapley or Banzhaf values of the transactions, involving aggregating over $(v(T'))_{T' \subseteq T}$. Hence, computing Shapley and Banzhaf values is, in general, NP-hard and #P-complete [19, 24, 41, 42, 52]. Moreover, no deterministic polynomial-time algorithm can approximate the Shapley values within a constant factor unless P = NP. Using randomization could, in principle, circumvent this: to approximate an average consisting of exponentially many terms, sample polynomially many uniformly at random, and take their average. One could imagine implementing this in a blockchain via a VRF or similar mechanism to ensure unbiased randomness. Still, this comes with a notable downside: the accuracy of the computed gas consumption may be hard to predict in advance and might vary wildly. Moreover, even with randomization, one could still run into issues computing the sampled terms, which is as hard as evaluating v a constant number of times. See [15] for an ampler discussion of computing Shapley and Banzhaf values. Note that the previous results mostly pertain to functions v of a different shape than ours (i.e., not related to scheduling). We have not attempted to show that hardness is retained in our context, but expect this to be true. Note still that because the Shapley GCM is Efficient, it can be used to compute $v(T)$ by adding up the gas consumptions of all transactions in T, meaning that the Shapley GCM is at least as difficult to compute as $v(T)$. However, this simple reduction no longer works for the Banzhaf GCM.

Efficiency (Property 7). We have already seen that the Shapley GCM is Efficient (Lemma 10) while the Banzhaf GCM is not (Lemma 12). Moreover, by adding up the gas consumption, one can immediately see from the definitions that TPM and ESM are

[17] Notably, this is essentially never true for non-trivial makespan minimization problems. In particular, if we restrict our attention to the case of unit-length transactions (i.e., with $t = 1$) and infinitely many threads $n = \infty$, then checking for the existence of a schedule with makespan c corresponds to checking whether the intersection graph of the transactions (i.e., where edges correspond to transactions with intersecting storage key sets) is c-colorable. For $c = 3$, this is well-known to be NP-complete, but this result is for general graphs. However, there is a straightforward way to model any graph $G = (V, E)$ as an intersection graph of transactions: vertices $v \in V$ are transactions and for every edge $(u, v) \in E$, create a new storage key k and add it to the storage key sets of transactions u and v.

Efficient, while XSM is not. We take this occasion to also note that any GCM can be made Efficient by appropriately scaling its output. Most notably, applying this to the Banzhaf GCM yields a mechanism based on the well-known *normalized Banzhaf values*, though the resulting mechanism otherwise seems rather poorly behaved.

Scheduling Monotonicity (Property 6). The ESM and XSM GCMs can be easily seen to strictly satisfy Scheduling Monotonicity. This is because, under both mechanisms, given a set of transactions T, the gas consumption of any transaction in T is computed as $f(|T|) \cdot v(T)$, where f is either $\frac{1}{x}$ or $\frac{1}{3^x}$. Notably, the first factor depends only on $|T|$, so if a transaction in T were modified in a way that increases the makespan, this increase would also be reflected proportionally in its gas consumption. In contrast, the Shapley, Banzhaf, and TPM GCMs do *not* satisfy Scheduling Monotonicity (proven in Lemma 17 below). This may be particularly surprising for the Shapley and Banzhaf GCMs, as they were specifically designed to account for marginal increases in makespan. The catch is that Scheduling Monotonicity considers replacing a transaction tx with another one leading to an increase in makespan. However, some elements in $(v(S \cup \{tx\}) - v(S))_{S \subseteq T \setminus \{tx\}}$ may still decrease as a result (except for $S = T \setminus \{tx\}$, for which we assumed an increase). Because both the Shapley and Banzhaf values of tx take a weighted average over these values, the average might still decrease, which is what happens in the example in the lemma below.

▶ **Lemma 17.** *The Shapley, Banzhaf and TPM GCMs do* not *satisfy Scheduling Monotonicity (Property 6).*

Proof. Consider the set of transactions $T = \{tx_1, tx_2\}$ and two additional transactions tx_3 and tx_4 such that $tx_1 \simeq (1, \{k_1\})$, $tx_2 \simeq (3, \{k_2\})$, $tx_3 \simeq (2, \{k_1\})$, and $tx_4 \simeq (1, \{k_2\})$. Then, for any number of threads $n \geq 2$ and the optimal scheduler we have $v(T \cup \{tx_3\}) = 3 < 4 = v(T \cup \{tx_4\})$. However:

$$\mathsf{gas}^{S}_{T\cup\{tx_3\}}(tx_3) = \frac{2+2+2+0+0+0}{6} = 1 > \frac{5}{6} = \frac{1+1+1+1+1+0}{6} = \mathsf{gas}^{S}_{T\cup\{tx_4\}}(tx_4)$$

$$\mathsf{gas}^{B}_{T\cup\{tx_3\}}(tx_3) = \frac{2+2+0+0}{4} = 1 > \frac{3}{4} = \frac{1+0+1+1}{4} = \mathsf{gas}^{B}_{T\cup\{tx_4\}}(tx_4)$$

$$\mathsf{gas}^{TPM}_{T\cup\{tx_3\}}(tx_3) = \frac{2}{1+3+2} \cdot 3 = 1 > \frac{4}{5} = \frac{1}{1+3+1} \cdot 4 = \mathsf{gas}^{TPM}_{T\cup\{tx_4\}}(tx_4).$$

So, the Shapley, Banzhaf, and TPM GCMs all violate Scheduling Monotonicity in this case. ◀

Transaction Bundling (Property 5). We find that the Banzhaf, TPM and XSM GCMs satisfy Transaction Bundling, only XSM satisfying it strictly, while the Shapley and ESM GCMs do not satisfy the property. We prove these facts in the following 5 lemmas.

▶ **Lemma 18.** *The Shapley GCM does* not *satisfy Transaction Bundling (Property 5).*

▶ **Lemma 19.** *The Banzhaf GCM satisfies Transaction Bundling (Property 5).*

▶ **Lemma 20.** *The TPM GCM satisfies Transaction Bundling (Property 5).*

▶ **Lemma 21.** *The ESM GCM does* not *satisfy Transaction Bundling (Property 5).*

▶ **Lemma 22.** *The XSM GCM strictly satisfies Transaction Bundling (Property 5).*

Set Inclusion (Property 4). We find that the TPM and ESM GCMs satisfy Set Inclusion, while the Shapley, Banzhaf and XSM GCMs do not satisfy the property. We prove these facts in the following 5 lemmas.

▶ **Lemma 23.** *The Shapley GCM does* not *satisfy Set Inclusion (Property 4).*

▶ **Lemma 24.** *The Banzhaf GCM does* not *satisfy Set Inclusion (Property 4).*

▶ **Lemma 25.** *The TPM GCM satisfies Set Inclusion (Property 4).*

▶ **Lemma 26.** *The ESM GCM satisfies Set Inclusion (Property 4).*

▶ **Lemma 27.** *The XSM GCM does* not *satisfy Set Inclusion (Property 4).*

Storage Key-Time Monotonicity (Property 3). All five mechanisms satisfy this property. For the ESM and XSM GCMs, this is an immediate consequence of property (S3). For the Shapley and Banzhaf GCMs, one can see this by recalling that they compute the gas consumption of a transaction $tx \in T$ as a weighted average over $(v(S \cup \{tx\}) - v(S))_{S \subseteq T \setminus \{tx\}}$. Hence, by property (S3), when tx is replaced with some $tx' \gtrsim tx$, no term in the previous decreases, so their weighted average also does not decrease. Finally, for the TPM GCM, we show this in the lemma after the next paragraph*.

Storage Key Monotonicity (Property 1) and Time Monotonicity (Property 2). Because all five mechanisms satisfy Storage Key-Time Monotonicity, by Lemma 4, they also all satisfy Storage Key Monotonicity and Time Monotonicity. Out of the five mechanisms, the Shapley, Banzhaf, and TPM GCMs satisfy the property strictly. For the first two, this is because when $t(tx)$ increases, at least one term in $(v(S \cup \{tx\}) - v(S))_{S \subseteq T \setminus \{tx\}}$ strictly increases, namely the term for $S = \varnothing$, while no terms decrease by property (S3). Last, for the TPM GCM, we show this in the lemma below.

▶ **Lemma 28.** *The TPM GCM satisfies Storage Key-Time Monotonicity (Property 3) and strict Time Monotonicity (Property 2).*

6 Towards a Fee Market for Parallel Execution

Armed with an understanding of the trade-offs between desirable properties in a GCM, in particular the impossibility of satisfying all properties within a single mechanism, and informed by our analysis of various candidate mechanisms and their properties, we propose two mechanisms for practical implementations of a fee market supporting parallel execution. Each represents one side of the design spectrum: one drawn from the class of mechanisms with Easy Gas Estimation, and the other from the class without it.

The advantage of adopting a mechanism with Easy Gas Estimation is that satisfying this property ensures that each transaction consumes a fixed amount of gas, regardless of other transactions in the same block. From the perspective of currently deployed TFMs (e.g., EIP-1559), which process transactions with fixed sizes, nothing changes. Thus, the existing properties of these mechanisms remain intact. On a high level, the key properties we strive for in a TFM are incentive compatibility for both block producers and users, welfare optimality, and collusion resistance. However, no TFM can achieve all these properties simultaneously [18, 17, 27]. Importantly, when composing a GCM that satisfies Easy Gas Estimation with a TFM of choice, the level of sophistication required from users in their bidding strategy does not increase, unlike in currently deployed fee markets for parallel execution [40, 44].

On the other hand, foregoing Easy Gas Estimation makes it possible to price transactions according to the load they impose on the network relative to the other transactions in the block, rather than evaluating each one in isolation. Mechanisms in this class can align the total gas charged in a block with the actual execution cost of that block, enabling resource pricing at the block level rather than only at the transaction level. Recent discussions in Ethereum research have explored similar designs under the umbrella of block-level fee markets [11]. If the protocol is already moving toward block-level metering for other resources, such as data availability or storage contention, extending this approach to execution costs could enable direct efficiency gains from parallel execution that are not achievable with transaction-level pricing alone.

For the class of mechanisms with Easy Gas Estimation, the natural choice is the Weighted Area GCM. Given that it satisfies Easy Gas Estimation, it achieves the most additional properties one could hope for in a (non-constant) GCM (see Theorem 7).

For the class of mechanisms without Easy Gas Estimation, we propose the Time-Proportional Makespan GCM as a concrete candidate. By allocating gas in proportion to each transaction's execution time relative to the block's total makespan, it provides a simple way to align gas consumption with how transactions constrain parallel execution within a block.

The decision between these approaches ultimately depends on protocol-level priorities: maintaining user-side simplicity and compatibility with existing TFMs through Easy Gas Estimation, which also makes implementation simpler by reusing current transaction-level pricing infrastructure, or pursuing block-level efficiency and more accurate resource pricing at the cost of giving up the previous benefits.

7 Related Work

7.1 Transaction Fee Mechanisms

There is extensive research on blockchain fee markets, with a particular focus on Ethereum and Bitcoin. Early studies primarily examined Bitcoin, exploring monopolistic pricing mechanisms [37, 64]. More recent contributions to this field include [48, 26, 50]. Unlike these works, our study concerns measuring storage key usage on a blockchain with client-side parallel execution, rather than focusing on pricing.

The TFM design framework was introduced by Roughgarden [54, 55]. Roughgarden's analysis of the EIP-1559 mechanism [12] initiated an active line of research on TFMs. Chung and Shi [18] demonstrated that no TFM can be ideal – meaning it cannot simultaneously be incentive-compatible for users and block producers while also being resistant to collusion between the two. This conclusion holds even for weaker definitions of collusion resilience, as shown by Chung et al. [17] and Gafni and Yaish [27]. Finally, attempts to address these limitations using cryptographic techniques [59, 62] have made progress in overcoming certain impossibilities, while other attempts relax the desiderata [28]. However, designing an ideal TFM still remains out of reach. While these studies examine the limitations of TFMs, our focus is on GCMs for parallel execution and how to integrate them with a TFM.

A related body of work examines the dynamics of TFMs over multiple blocks, particularly focusing on the base fee in EIP-1559. Leonardos et al. [38, 39] demonstrate that the stability of the base fee depends on the adjustment parameter, with short-term volatility but long-term block size stability. Reijsbergen et al. [53] suggest using an adaptive adjustment parameter to mitigate block size fluctuations, while Ferreira et al. [25] highlight user experience issues caused by bounded base fee oscillations. Additionally, Hougaard and Pourpouneh [32] and Azouvi et al. [6] reveal that the base fee can be manipulated by non-myopic miners.

Given the discussion surrounding multi-dimensional fees in Ethereum [9, 10] and the deployment of EIP-4844 [13] (a first step towards a multi-dimensional fee market on Ethereum), a recent line of work explores multi-dimensional fee markets, focusing on efficient pricing mechanisms and their optimality. This work is further refined by Diamandis et al. [21], who design and analyze multi-dimensional blockchain fee markets to align incentives and improve network performance. Building on this, Angeris et al. [3] prove that such fee markets are nearly optimal, with efficiency improving over time even under adversarial conditions. Multidimensional fee markets are closely related to fee markets designed for parallel execution. In particular, in the weighted area GCM, the weights can be interpreted as fees within a multidimensional fee market. Unlike previous literature on multidimensional fee markets, we focus on parallelization, introduce desirable properties, and evaluate how various mechanisms perform.

Further extensions of TFMs have emerged. Bahrani et al. [8] consider TFMs in the presence of maximal extractable value (MEV), i.e., value extractable by the block producer. Further, Wang et al. [61] design a fee mechanism for proof networks, whereas Bahrani et al. [7] introduce a transaction fee mechanism for heterogeneous computation. Our work most closely relates to the latter, but, in contrast, our chosen approach is closer to multidimensional fee markets, trading complexity for the block producer for stronger incentive compatibility for the user.

Local fee markets have recently been a topic of discussion in the blockchain space [23, 20, 40, 34]. The core idea is that transactions interacting with highly contested states incur higher fees, while those involving non-contested states pay lower fees. However, discussions on local fee markets have largely remained high-level, without a precise characterization of the desired properties beyond this general goal. Moreover, currently implemented local fee markets [40] require significant user sophistication to set fees appropriately. In this work, we formalize the desiderata for fee markets in the context of parallel execution and identify the weighted area GCM as a promising candidate. One key advantage is its compatibility with a TFM, enabling simple fee estimation for users.

7.2 Parallel Execution

Blockchain concurrency has been a focal point in an active line of research. In particular, numerous efforts have aimed to enable parallel transaction processing through speculative execution [58, 65, 2, 22, 4, 29, 16, 56]. Note that speculative execution is already deployed by multiple blockchains [5, 57, 43]. Static analysis has also been employed to identify parallelizable transactions, though it cannot completely eliminate inherent dependencies [51, 45]. Similarly, Neiheiser et al. [47] demonstrate how parallel execution can assist struggling nodes in catching up. While these works are orthogonal to ours, they highlight the overhead of parallel execution when there is no advance knowledge about a transaction's state accesses.

Further, Saraph and Herlihy [56] and Heimbach et al. [30] have evaluated the parallelization potential of the Ethereum workload. The latter demonstrates that a speedup of approximately fivefold is achievable, assuming state accesses are known in advance. Additionally, Solana [60] and Sui [46] already perform parallel execution with advance knowledge of state accesses. However, in practice, state accesses are not known beforehand on many blockchains such as Ethereum. There, less than 2% of transactions disclose them proactively, as shown by Heimbach et al. [31], due to a lack of incentives. In this work, we aim to take a step toward unlocking the parallelization potential by designing a TFM that supports parallel execution. This mechanism relies on the disclosure of state accesses as done in Solana [60] and Sui [46].

7.3 Cloud Computing Pricing and Parallel Resource Allocation

While our primary focus is on transaction fee mechanisms for blockchain networks, parallels can be drawn to resource allocation and pricing in cloud computing. Several papers study pricing schemes for cloud computing resources in a monopolistic setting, where a single provider sells access to multiple resources and sets prices to optimize revenue [63]. In these models, the provider posts prices for resource usage, users select resource bundles to maximize their individual surplus, and the provider earns revenue from the resulting allocation. Mechanisms such as CloudPack [33] extend this approach by incorporating workload flexibility, using concepts like Shapley values for fair cost distribution among colocated jobs. Similarly, works such as [49] address resource contention through dynamic pricing strategies that adjust costs based on runtime slowdowns.

While structurally similar, the blockchain context introduces an additional constraint. In addition to objectives like maximizing provider revenue, it emphasizes maximizing throughput by ensuring transactions are processed efficiently, minimizing their collective execution time within a block. Moreover, blockchain transactions explicitly declare access to distinct storage keys, introducing contention through overlapping access patterns. This adds complexity compared to cloud models, which typically abstract resource demands as scalar quantities and address contention at a more aggregated level.

Capturing the effects of storage key-level conflicts on parallel execution in blockchains requires fee computation mechanisms that are sensitive to these fine-grained interactions. Thus, while conceptually addressing similar challenges, the blockchain setting imposes unique constraints that motivate the need for specialized GCMs as explored in this work.

8 Conclusion

In this work, we took a step towards creating a fee market that meets the demands of parallel execution environments while also upholding the properties we want from a TFM.

Recently, the idea of local fee markets has been proposed for blockchains that support parallel execution. However, to the best of our knowledge, before this work, the demands on these fee markets have only been outlined at a very high level, and the markets that have been implemented are not ideal yet, e.g., they require high levels of sophistication from users when bidding.

In this work, we addressed this gap by introducing a framework with two key components: a GCM, which measures the execution-related load a transaction imposes on the network in units of gas, and a TFM, which determines the cost associated with each unit of gas. We then formalized the desired properties for the GCM in such a fee market. After outlining these desiderata, we evaluated various mechanisms against them and identified two strong candidates through this analysis: the *weighted area* GCM for the class of mechanisms with Easy Gas Estimation, and the *time-proportional makespan* GCM for the class without it.

Setting the right incentives in fee markets for parallel execution is crucial to unlocking the full potential of execution layer parallelization, and we hope that our work contributes to the development of fee markets capable of meeting the demands of such environments.

References

1 Bahar Acilan, Andrei Constantinescu, Lioba Heimbach, and Roger Wattenhofer. Transaction Fee Market Design for Parallel Execution, 2025. doi:10.48550/arXiv.2502.11964.

2 Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. Parblockchain: Leveraging Transaction Parallelism in Permissioned Blockchain Systems. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1337–1347. IEEE, 2019. doi:10.1109/ICDCS.2019.00134.

3 Guillermo Angeris, Theo Diamandis, and Ciamac Moallemi. Multidimensional Blockchain Fees are (Essentially) Optimal. *arXiv preprint arXiv:2402.08661*, 2024. doi:10.48550/arXiv.2402.08661.

4 Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. Optsmart: A space efficient optimistic concurrent execution of smart contracts. *Distributed and Parallel Databases*, pages 1–53, 2022.

5 Aptos Labs. Aptos, 2023. Accessed: 2025-01-23. URL: https://www.aptoslabs.com.

6 Sarah Azouvi, Guy Goren, Lioba Heimbach, and Alexander Hicks. Base Fee Manipulation in Ethereum's EIP-1559 Transaction Fee Mechanism. In *37th International Symposium on Distributed Computing*, 2023.

7 Maryam Bahrani and Naveen Durvasula. Resonance: Transaction Fees for Heterogeneous Computation. *arXiv preprint arXiv:2411.11789*, 2024. doi:10.48550/arXiv.2411.11789.

8 Maryam Bahrani, Pranav Garimidi, and Tim Roughgarden. Transaction Fee Mechanism Design with Active Block Producers. In *International Conference on Financial Cryptography and Data Security*, pages 85–90. Springer, 2024. doi:10.1007/978-3-031-69231-4_6.

9 Vitalik Buterin. Multidimensional EIP-1559. https://ethresear.ch/t/multidimensional-eip-1559/11651, 2022. Accessed: 2025-01-23.

10 Vitalik Buterin. Multidimensional Pricing for EIP-1559. https://vitalik.eth.limo/general/2024/05/09/multidim.html, May 2024. Accessed: 2025-01-23.

11 Vitalik Buterin. Block-level fee markets: Four easy pieces. https://ethresear.ch/t/block-level-fee-markets-four-easy-pieces/21448, 2025. Accessed: 2025-08-04.

12 Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. Eip-1559: Fee market change for eth 1.0 chain. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md, 2024. Accessed: 2025-01-23.

13 Vitalik Buterin, Dankrad Feist, Diederik Loerakker, George Kadianakis, Matt Garnett, Mofi Taiwo, and Ansgar Dietrichs. EIP-4844: Shard Blob Transactions. https://eips.ethereum.org/EIPS/eip-4844, 2022. Accessed: 2025-01-23.

14 Vitalik Buterin and Martin Swende. Eip-2930: Optional access lists, October 2020. Accessed: 2025-01-22. URL: https://eips.ethereum.org/EIPS/eip-2930.

15 Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. *Weighted Voting Games*, pages 49–70. Springer International Publishing, Cham, 2012. doi:10.1007/978-3-031-01558-8_5.

16 Yang Chen, Zhongxin Guo, Runhuai Li, Shuo Chen, Lidong Zhou, Yajin Zhou, and Xian Zhang. Forerunner: Constraint-Based Speculative Transaction Execution for Ethereum. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 570–587, 2021. doi:10.1145/3477132.3483564.

17 Hao Chung, Tim Roughgarden, and Elaine Shi. Collusion-Resilience in Transaction Fee Mechanism Design. In *Proceedings of the 25th ACM Conference on Economics and Computation*, pages 1045–1073, 2024. doi:10.1145/3670865.3673550.

18 Hao Chung and Elaine Shi. Foundations of Transaction Fee Mechanism Design. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3856–3899. SIAM, 2023. doi:10.1137/1.9781611977554.CH150.

19 Xiaotie Deng and Christos H. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994. doi:10.1287/moor.19.2.257.

20 Theo Diamandis, Tarun Chitra, and 0xShitTrader. Toward multidimensional solana fees. *Umbra Research*, 2024. URL: https://www.umbraresearch.xyz/writings/toward-multidimensional-solana-fees.

21 Theo Diamandis, Alex Evans, Tarun Chitra, and Guillermo Angeris. Designing Multidimensional Blockchain Fee Markets. In *5th Conference on Advances in Financial Technologies (AFT 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.AFT.2023.4.

22 Thomas Dickerson, Paul Gazzillo, Maurice Herlihy, and Eric Koskinen. Adding Concurrency to Smart Contracts. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 303–312, 2017. doi:10.1145/3087801.3087835.

23 Eclipse Labs. Local fee markets are necessary to scale ethereum. *Eclipse*, 2024. URL: https://www.eclipse.xyz/articles/local-fee-markets-are-necessary-to-scale-ethereum.

24 Piotr Faliszewski and Lane Hemaspaandra. The complexity of power-index comparison. *Theoretical Computer Science*, 410(1):101–107, 2009. doi:10.1016/j.tcs.2008.09.034.

25 Matheus VX Ferreira, Daniel J Moroz, David C Parkes, and Mitchell Stern. Dynamic Posted-Price mMchanisms for the Blockchain Transaction-Fee Market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 86–99, 2021.

26 Yotam Gafni and Aviv Yaish. Greedy Transaction Fee Mechanisms for (Non-) Myopic Miners. *arXiv preprint arXiv:2210.07793*, 5, 2022. doi:10.48550/arXiv.2210.07793.

27 Yotam Gafni and Aviv Yaish. Barriers to Collusion-Resistant Transaction Fee Mechanisms. In *Proceedings of the 25th ACM Conference on Economics and Computation*, pages 1074–1096, 2024. doi:10.1145/3670865.3673469.

28 Yotam Gafni and Aviv Yaish. Discrete and Bayesian Transaction Fee Mechanisms. In *The International Conference on Mathematical Research for Blockchain Economy*, pages 145–171. Springer, 2024. doi:10.1007/978-3-031-68974-1_8.

29 Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Yu Xia, Runtian Zhou, and Dahlia Malkhi. Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing. *arXiv preprint arXiv:2203.06871*, 2022. doi:10.48550/arXiv.2203.06871.

30 Lioba Heimbach, Quentin Kniep, Yann Vonlanthen, and Roger Wattenhofer. Defi and nfts hinder blockchain scalability. In *International Conference on Financial Cryptography and Data Security*, pages 291–309. Springer, 2023. doi:10.1007/978-3-031-47751-5_17.

31 Lioba Heimbach, Quentin Kniep, Yann Vonlanthen, Roger Wattenhofer, and Patrick Züst. Dissecting the EIP-2930 Optional Access Lists. *arXiv preprint arXiv:2312.06574*, 2023. doi:10.48550/arXiv.2312.06574.

32 Jens Leth Hougaard and Mohsen Pourpouneh. Farsighted Miners Under Transaction Fee Mechanism EIP-1559. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2023. doi:10.1109/ICBC56567.2023.10174974.

33 Vatche Isahagian, Raymond Sweha, Azer Bestavros, and Jonathan Appavoo. Cloudpack: Exploiting workload flexibility through rational pricing. In *Proceedings of the 13th ACM/IFIP/USENIX Middleware Conference*, December 2012.

34 keyneom. Local fee markets in ethereum. *Ethereum Research*, 2024. URL: https://ethresear.ch/t/local-fee-markets-in-ethereum/20754.

35 Mysten Labs. Sui object model documentation, 2025. Accessed: 2025-01-22. URL: https://docs.sui.io/concepts/object-model.

36 Solana Labs. Solana transactions documentation, 2025. Accessed: 2025-01-22. URL: https://solana.com/de/docs/core/transactions.

37 Ron Lavi, Or Sattath, and Aviv Zohar. Redesigning Bitcoin's Fee Market. *ACM Transactions on Economics and Computation*, 10(1):1–31, 2022. doi:10.1145/3530799.

38 Stefanos Leonardos, Barnabé Monnot, Daniël Reijsbergen, Efstratios Skoulakis, and Georgios Piliouras. Dynamical Analysis of the EIP-1559 Ethereum Fee Market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 114–126, 2021. doi:10.1145/3479722.3480993.

39 Stefanos Leonardos, Daniël Reijsbergen, Barnabé Monnot, and Georgios Piliouras. Optimality Despite Chaos in Fee Markets. In *International Conference on Financial Cryptography and Data Security*, pages 346–362. Springer, 2023. doi:10.1007/978-3-031-47751-5_20.

40 Lostin. The truth about solana local fee markets. *Helius*, 2025. URL: https://www.helius.dev/blog/solana-local-fee-markets.

41 Tomomi Matsui and Yasuko Matsui. A survey of algorithms for calculating power indices of weighted majority games. *Journal of the Operations Research Society of Japan*, 43:71–86, November 2000. doi:10.15807/jorsj.43.71.

42 Yasuko Matsui and Tomomi Matsui. Np-completeness for calculating power indices of weighted majority games. *Theoretical Computer Science*, 263(1):305–310, 2001. Combinatorics and Computer Science. doi:10.1016/S0304-3975(00)00251-6.

43 Monad Labs. Monad, 2023. Accessed: 2025-01-23. URL: https://www.monad.xyz.

44 Sebastian Mueller. X post by Sebastian Mueller. https://x.com/NaitsabesMue/status/1862519048069959893, 2025. Accessed: 2025-02-08.

45 Maurizio Murgia, Letterio Galletta, and Massimo Bartoletti. A Theory of Transaction Parallelism in Blockchains. *Logical Methods in Computer Science*, 17, 2021. doi:10.46298/LMCS-17(4:10)2021.

46 Mysten Labs. Sui, 2023. Accessed: 2025-01-23. URL: https://sui.io.

47 Ray Neiheiser, Arman Babaei, Ioannis Alexopoulos, Marios Kogias, and Eleftherios Kokoris Kogias. Pythia: Supercharging Parallel Smart Contract Execution to Guide Stragglers and Full Nodes to Safety. https://aftsib.com/papers/SIB24_paper_4.pdf, 2024. Accessed: 2025-01-23.

48 Noam Nisan. Serial Monopoly on Blockchains. *arXiv preprint arXiv:2311.12731*, 2023. doi:10.48550/arXiv.2311.12731.

49 Qi Pei, Yipeng Wang, and Seunghee Shin. Litmus: Fair pricing for serverless computing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, ASPLOS '24, pages 155–169. ACM, April 2024. doi:10.1145/3622781.3674181.

50 Paolo Penna and Manvir Schneider. Serial Monopoly on Blockchains with Quasi-patient Users. *arXiv preprint arXiv:2405.17334*, 2024. doi:10.48550/arXiv.2405.17334.

51 George Pîrlea, Amrit Kumar, and Ilya Sergey. Practical Smart Contract Sharding with Ownership and Commutativity Analysis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 1327–1341, 2021. doi:10.1145/3453483.3454112.

52 K. Prasad and J. S. Kelly. Np-completeness of some problems concerning voting games. *International Journal of Game Theory*, 19(1):1–9, March 1990. doi:10.1007/BF01753703.

53 Daniël Reijsbergen, Shyam Sridhar, Barnabé Monnot, Stefanos Leonardos, Stratis Skoulakis, and Georgios Piliouras. Transaction Fees on a Honeymoon: Ethereum's EIP-1559 One Month Later. In *2021 IEEE International Conference on Blockchain (Blockchain)*, pages 196–204. IEEE, 2021. doi:10.1109/BLOCKCHAIN53845.2021.00034.

54 Tim Roughgarden. Transaction Fee Mechanism Design for the Ethereum Blockchain: An Economic Analysis of EIP-1559. *arXiv preprint arXiv:2012.00854*, 2020. arXiv:2012.00854.

55 Tim Roughgarden. Transaction Fee Mechanism Design. *Journal of the ACM*, 71(4):1–25, 2024. doi:10.1145/3674143.

56 Vikram Saraph and Maurice Herlihy. An Empirical Study of Speculative Concurrency in Ethereum Smart Contracts. *arXiv preprint arXiv:1901.01376*, 2019. arXiv:1901.01376.

57 Sei Labs. Sei protocol, 2023. Accessed: 2025-01-23. URL: https://www.sei.io.

58 Ilya Sergey and Aquinas Hobor. A Concurrent Perspective on Smart Contracts. In *International Conference on Financial Cryptography and Data Security*, pages 478–493. Springer, 2017. doi:10.1007/978-3-319-70278-0_30.

59 Elaine Shi, Hao Chung, and Ke Wu. What can Cryptography do for Decentralized Mechanism Design. *arXiv preprint*, 2022. arXiv:2209.14462.

60 Solana Labs. Solana, 2023. Accessed: 2025-01-23. URL: https://solana.com.

61 Wenhao Wang, Lulu Zhou, Aviv Yaish, Fan Zhang, Ben Fisch, and Benjamin Livshits. Mechanism Design for ZK-Rollup Prover Markets. *arXiv preprint arXiv:2404.06495*, 2024. doi:10.48550/arXiv.2404.06495.

62 Ke Wu, Elaine Shi, and Hao Chung. Maximizing Miner Revenue in Transaction Fee Mechanism Design. *arXiv preprint*, 2023. arXiv:2302.12895.

63 Hong Xu and Baochun Li. A study of pricing for cloud resources. *SIGMETRICS Perform. Eval. Rev.*, 40(4):3–12, April 2013. doi:10.1145/2479942.2479944.

64 Andrew Chi-Chih Yao. An Incentive Analysis of some Bitcoin Fee Designs. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2020. doi:10.4230/LIPIcs.ICALP.2020.1.

65 An Zhang and Kunlong Zhang. Enabling Concurrency on Smart Contracts Using Multiversion Ordering. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data*, pages 425–439. Springer, 2018. doi:10.1007/978-3-319-96893-3_32.

Multidimensional Blockchain Fees Are (Essentially) Optimal

Guillermo Angeris
Bain Capital Crypto, Boston, MA, USA

Theo Diamandis
Bain Capital Crypto, Boston, MA, USA

Ciamac Moallemi
Graduate School of Business, Columbia University, New York, NY, USA

Abstract

In this paper we show that, using only mild assumptions, dynamic multidimensional blockchain fee markets have strong performance guarantees, even against worst-case adversaries. In particular, we show that the average welfare gap between the following two scenarios is at most $O(1/\sqrt{T})$, where T is the length of the time horizon considered. In the first scenario, the designer knows all future actions by users and is allowed to fix the optimal prices of resources ahead of time, based on the designer's oracular knowledge of those actions. In the second, the prices are updated by a very simple algorithm that does not have this oracular knowledge, special cases of which are EIP-4844 and EIP-1559, both fee mechanisms used by the Ethereum blockchain. Roughly speaking, this means that, on average, over a reasonable timescale, there is no difference in welfare between "correctly" fixing the prices, with oracular knowledge of the future, when compared to the proposed algorithm. We show a matching lower bound of $\Omega(1/\sqrt{T})$ for any implementable algorithm and also separately consider the case where the adversary is known to be stochastic.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Convex optimization; Information systems → Digital cash; Theory of computation → Algorithmic mechanism design

Keywords and phrases Blockchains, transaction fees, online optimization, convex optimization

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.24

Related Version *Full Version*: https://arxiv.org/abs/2402.08661

1 Introduction

Public blockchains allow any user to submit a transaction that modifies the shared state of the network. These transactions are independently verified and executed by a decentralized network of full nodes, who must each re-execute the transaction to verify the correct updated state of the blockchain. Each transaction requires some amount of *resources*, such as compute or storage, to be executed. Because full nodes have finite resources, blockchains must limit the total computational resources that can be consumed per unit of time, usually measured per-block. As user demand may fluctuate, most blockchains implement a transaction fee mechanism in order to allocate finite resources among competing transactions.

Transaction fees. Transaction fee mechanisms in practice often use a combination of a *base fee*, which is the minimum cost of entry for a particular block, and a *priority fee*, which is an additional fee that a user may pay so that a particular contentious transaction is included. In this paper, we concentrate on the base fee for transaction inclusion. These fees are computed algorithmically as part of the blockchain's consensus mechanism; any transaction not paying the computed base fee is invalid. Thus, the algorithm for computing these base fees is a

© Guillermo Angeris, Theo Diamandis, and Ciamac Moallemi;
licensed under Creative Commons License CC-BY 4.0

crucial part of the blockchain's design, as it ultimately determines which transactions can be included in the blockchain and the blockchain's robustness to adversarial behavior, such as spam or denial-of-service attacks.

Dynamic base fees. Because user demand fluctuates, many blockchains have implemented or proposed a dynamic base fee mechanism. These mechanisms update the base fee based on current demand for resources, measured by the resource consumption of the transactions in previous blocks. Most famously, EIP-1559 [14] for the Ethereum blockchain introduced a base fee that adjusts to track a target block usage. Many of these mechanisms can be interpreted as applying gradient descent on the dual of a particular optimization problem [19].

Multidimensional fees. Calculating transaction fees through a single, joint unit of account fixes the relative prices of all resources. These fixed relative prices inhibit granular price discovery and, ultimately, reduce the throughput of the blockchain. To mitigate this behavior, some blockchains have proposed *multidimensional fees*, where different resources are priced separately. EIP-4844 [15], to be implemented in the upcoming Dencun update, allows users to submit special pieces of data called blobs [1, 2], which have an independent fee mechanism. In essence, EIP-4844 creates a two-dimensional fee market. SIMD-110 [38] in the Solana ecosystem proposes a local fee controller for each piece of state on the Solana virtual machine. During periods of high user demand, highly contested pieces of state would become more expensive without affecting the price of other state. Similar multidimensional mechanisms have been proposed for the Ethereum ecosystem [3, 13] and other blockchains, such as Avalanche [29] and Penumbra [30], have also implemented multidimensional blockchain fees. Like dynamic base fees, these multidimensional mechanisms can be designed and interpreted by considering the optimization problem that they attempt to solve.

Optimality. While the design and incentives of particular existing transaction fee mechanisms have been extensively studied, the overall design space has not been well-characterized. In particular, it is unclear what properties we can hope for from a "good" fee mechanism, the tradeoffs present within the space of "good" mechanisms, and what "good" even means in this context, where adversarial behavior may preclude the existence of "true" market-clearing prices for resources. These open questions are important for the many blockchains currently working on updated fee mechanisms, especially because many traditional results do not apply in the adversarial blockchain environment.

This paper. In this paper, we seek to characterize the optimality of the algorithm for setting multidimensional blockchain fees proposed in [19] when extended to the online setting. To judge the optimality of this algorithm, we rely on the notion of *regret* from the online convex optimization literature [35, 23]. This quantity measures the difference between two scenarios: in the first, we dynamically update the resource prices as in [19, §3.4]; in the second, we assume that the algorithm knows all future transactions and chooses the best fixed vector of prices given this knowledge. (The second scenario is similar to the current implementation of gas in Ethereum, but, of course, the fixed gas prices of individual resources are not known to be optimal.) We show that the average welfare gap between the two scenarios is at most $O(1/\sqrt{T})$, where T is the length of the time horizon considered, and we construct a corresponding lower bound. Taken together our results show that the proposed algorithm is essentially optimal according to this metric, even under arbitrary adversarial behavior that eludes traditional game theoretic analysis. We also show that, if the adversary is stochastic,

the prices do converge to a market clearing price and give an explicit rate of convergence. We point to several additional directions for future work, where we suspect additional tools from online optimization may be used to improve upon the results presented here.

1.1 Related work

There are a number of papers proposing new mechanisms for pricing resources or analyzing mechanisms currently implemented in practice, which we describe next. The most similar in spirit to ours is [6], which cites a preprint of this article for its set up. They show that, when users submitting transactions are patient (*i.e.*, their utility does not depend on when their transaction ultimately gets included), a one-dimensional exponential update rule, akin to EIP-1559, is essentially optimal compared to any baseline. This mechanism is a special case of the mechanism studied in this paper (see (7) and derivation in 3.1). Our mechanism allows for both alternative update rules and more than one resource. We review some of the other relevant literature below.

Blockchain resource pricing. In the specific case of blockchain resource pricing, these papers can be roughly broken down into three buckets, which we describe next. The first set analyzes past or current fee mechanisms or variations thereof. These papers include some of the original economic analyses of EIP-1559 [33], empirical studies [12, 27], simulation studies [31], and clean theoretical models [25, 26]. All of these papers focus on the single-dimensional resource case, usually under some simplifying assumptions such as, *e.g.*, certain stochastic limits in the case of [26]. The second set of papers proposes simple extensions or characterizations of mechanisms for pricing single resources in a variety of settings [20, 8, 9], including some impossibility results [16]. The final set proposes multidimensional fee mechanisms for a variety of settings [13, 19, 18]. The multidimensional fees work closest in spirit to ours is [18], which proposes a stochastic demand model of resources with stochastic observations. They apply linear-quadratic control theory to set prices that minimize a composite loss function that seeks to balance matching the target block size, while smoothly changing prices.

This work, on the other hand, shows that, even in an adversarial setting, a very simple update rule suffices and has low regret relative to optimally setting the prices. We also show that, in the "easier" stochastic setting, we recover a market-clearing price and provide a rate of convergence. One important case where we differ from many previous analyses is that we do not consider only incentive compatibility, but, indeed, consider worst-case adversarial behavior against the pricing mechanism and provide optimality bounds even in this particularly difficult setting.

Tatonnement processes. The design of transaction fee mechanisms via gradient descent on an appropriately-chosen dual problem is also related to the literature on tatonnement processes [36, 17]. Many works (*e.g.*, [11] and [32]) use gradient descent-like algorithms to develop dynamic pricing strategies for various types of markets. These algorithms use a dynamic step size to ensure convergence to optimal prices. More similar to our work, [5] uses a fixed step size, and their algorithm, as a result, does not necessarily converge to any single price.

2 Problem set up

In this section, we briefly introduce the blockchain resource allocation problem of [19]. We review how the dual problem leads to a notion of resource prices, which, when set correctly, ensure that block builders include the "optimal" transactions, *i.e.*, those that maximize the utility of the transaction producers, minus the loss incurred by the blockchain for including those transactions.

2.1 The block building problem

We will fix some notation that we use for the remainder of the paper in the paragraphs that follow and describe the problem we are looking to solve.

Transactions and resources. First, let $t = 1, \dots, T$ denote the current blockheight. At blockheight $t-1$, we assume that there are n_t transactions from which we may pick a subset to be included in block t. (This list of transactions that we are allowed to choose from is often called the *mempool*.) Each of these $j = 1, \dots, n_t$ transactions has some associated utility $(q_t)_j \in \mathbf{R}_+$ which is gained if it is included in the block. Additionally, transaction j, if included in the block, will consume some amount of the m available resources, which we denote as an m-vector $a_j \in \mathbf{R}^m$. (The ith entry of this vector, $(a_j)_i$, denotes how much of resource i is consumed by transaction j.) Finally, the vector $x_t \in \{0,1\}^{n_t}$ will denote the transactions that were included in block t: the jth entry $(x_t)_j = 1$ if transaction j is included in block t and is 0, otherwise.

Total consumption. This set up lets us write the total resource consumption of all transactions included in block t in a convenient way. Define the m-vector

$$y_t = \sum_{j=1}^{n} a_j (x_t)_j = A_t x_t,$$

where the matrix $A_t \in \mathbf{R}_+^{m \times n_t}$ has jth column a_j denoting the resources consumed by transaction j in the mempool. We then have that $(y_t)_i$ denotes the total amount of resource i is consumed by the transactions included in block t.

Allowable transactions. We denote the set of allowable transactions in a block by $S_t \subseteq \{0,1\}^{n_t}$, which may include resource limits (*e.g.*, if $x \in S_t$, then $Ax \leq b$), and interactions among transactions. Examples may include complementarity constraints such as, if $x \in S_t$, then $x_j = 1$ implies that $x_{j'} = 0$ for some $j' \neq j$, among many other, potentially very complicated, constraints.

Loss function. Finally, we denote the network's "unhappiness" with the resource usage at block t as the loss function $\ell_t : \mathbf{R}_+^m \to \mathbf{R} \cup \{\infty\}$. This function maps the resource usage at block t, which is always a nonnegative vector, to some number $\ell_t(y_t)$. Higher values denote less desirable resource usage. We assume this function, which is chosen by the blockchain designer, is convex and lower semicontinuous for the remainder of the paper.

Block building problem. We now define the *block building problem* as the problem of optimizing the utility of the included transactions, minus the loss incurred by the network, subject to the transactions being in the allowable transaction set:

$$\begin{array}{ll} \text{maximize} & q_t^T x_t - \ell_t(y_t) \\ \text{subject to} & y_t = A_t x_t \\ & x_t \in \mathbf{conv}(S_t). \end{array} \tag{1}$$

Here, the variables are the included transactions $x_t \in \mathbf{R}^{n_t}$ and the total resource consumption of the included transactions, $y_t \in \mathbf{R}^m$. Additionally, $\mathbf{conv}(S_t)$ denotes the convex hull of the set S_t. Replacing S_t with its convex hull allows "fractional" transactions, such as $x_j = 1/3$. These values of $x_j \in (0, 1)$ can be interpreted as "spreading out" transaction j over the next $1/x_j$ blocks. We denote the optimal value of problem (1) by $s^\star$.

Bounds. In general, it is a good idea to have maximum per-block resource consumption bounds, which we write as $b \in \mathbf{R}^m_+$, so we will assume that

$$S_t \subseteq \{x \in \{0, 1\}^{n_t} \mid A_t x \leq b\},$$

so any allowable choice of transactions $x_t \in S_t$ will satisfy

$$0 \leq A_t x_t \leq b,$$

as both A_t and x_t are nonnegative. In other words, we assume that the maximum amount of resource i that any set of acceptable transactions may consume is no more than b_i. Note that this also implies

$$0 \leq y_t \leq b,$$

for any y_t feasible for (1) since $y_t = A_t x_t$. (For why such limits are necessary, see the discussion in [19, §1].) Since $0 \leq y_t \leq b$ for any feasible points, we assume, without loss of generality, that $\ell_t(y) = \infty$ whenever $y \not\leq b$ or $y \not\geq 0$ for convenience.

Discussion. We may view this block building problem as the following "idealized" scenario: if we, as the network, knew the welfare generated by each transaction, along with all possible constraints for all transactions then solving problem (1) would give us a way to include transactions as to maximize welfare minus the loss incurred by the network. Of course, in general, almost all of these things are not known (or, potentially even knowable) to the network, so we will show how to approximately solve this problem by, instead, correctly setting the price of the consumed resources for each block.

Resources. We also note that the number of resources n here is left to the blockchain designer. A valid question might be: what is a reasonable number of resources and just how granular should these resources be in order to ensure that the system behaves as intended? While it may be tempting to have as many different resources as possible in order to have the most granular possible price discovery, we will see later that this may result in a very weak bound, which we show can be matched by an adversary. On the other hand, if many resources are correlated in their usage, it may be good to run some dimensionality reduction technique (such as PCA) on the historical resource utilization and only explicitly price a few "meta" resources.

2.2 The dual problem

In this subsection, we construct a particular dual problem, whose variables are the prices of the m possible resources; correctly setting these prices would solve our original problem (1). We then show that its gradients can be efficiently evaluated, without knowledge of things like the allowable transactions S_t nor the welfare q_t for each block t. This suggests a heuristic in which we use the gradient at each step to update the prices of the resources, which we prove in a later section is (essentially) optimal.

The dual function. One possible relaxation of (1), for each block $t = 1, \ldots, T$, can be constructed by relaxing the equality constraint in (1) to a penalty, parametrized by a vector $p_t \in \mathbf{R}^m$, which we will call the *prices*:

$$\begin{array}{ll} \text{maximize} & q_t^T x - \ell_t(y_t) + p_t^T(y_t - A_t x_t) \\ \text{subject to} & x_t \in \mathbf{conv}(S_t), \end{array}$$

where the variables are the same as those of problem (1). If we denote the optimal value of this problem, which is parametrized by the vector p_t, as $f_t(p_t)$, then, since the objective is separable over x_t and y_t, we can write the optimal value of this problem as

$$f_t(p_t) = \sup_y \left(p_t^T y - \ell_t(y)\right) + \sup_{x \in \mathbf{conv}(S_t)} (q_t - A_t^T p_t)^T x, \tag{2}$$

We note that this function f_t, which we will call the *dual function*, is convex since it is the pointwise supremum of a family of functions linear in p_t.

Fenchel conjugate. The first term of (2) can be recognized as the Fenchel conjugate of ℓ [10, §3.3] evaluated at p_t, which we write as $\ell_t^*(p_t)$. This function can be interpreted as choosing the resource allocation y that maximizes the network's profit minus the loss as a result of resource consumption y. We note that the Fenchel conjugate has a closed-form expression for many commonly used loss functions.

Block building problem. The second term in (2) is the optimal value of the following problem:

$$\begin{array}{ll} \text{maximize} & (q_t - A_t^T p_t)^T x \\ \text{subject to} & x \in \mathbf{conv}(S_t), \end{array}$$

with variable $x \in \mathbf{R}^n$. We can interpret this problem as the transaction producers' problem of creating and choosing transactions to be included in a block in order to maximize the utility of the included transactions ($q_t^T x$), after netting the fees paid to the network ($(A_t^T p_t)^T x$). (This problem is sometimes called the *block building problem* or *packing problem* that is solved by block producers. These "block producers" include both users and validators.) Since the objective of this problem is linear, it has the same optimal value as the nonconvex problem

$$\begin{array}{ll} \text{maximize} & (q_t - A_t^T p_t)^T x \\ \text{subject to} & x \in S_t, \end{array} \tag{3}$$

with variable $x \in \mathbf{R}^n$. Note that the packing problem problem depends on the problem data associated with block t: the utilities q_t, the resources required by the submitted transactions, A_t, and the associated constraints S_t for a particular block at time t. We will denote the optimal value of the packing problem as $h_t(p)$, where the index t indicates that the problem data is associated with block t.

Gradient. With this notation, the dual function can be written as

$$f_t(p) = \ell_t^\star(p) + h_t(p).$$

We note, for future use, that, when f_t is differentiable at p,

$$\nabla f_t(p) = y_t^\star(p) - A_t x_t^\star(p), \tag{4}$$

where $y_t^\star(p)$ denotes the maximizer for the Fenchel conjugate of ℓ_t, at price p, while $x_t^\star(p)$ denotes the optimal transactions to be included for the block building problem (3) at price p. If f_t is not differentiable at p, then any solution to each subproblem suffices. Note that both of these quantities are easy to compute if price p is set at blockheight $t-1$ and block t is submitted with these resource prices p: the former term, $y^\star(p)$, is the solution to a basic problem that often has a closed form, while the latter term, $Ax^\star(p)$, is exactly the total resource consumption of the transactions included in block t.

Bounds on the gradient. From the discussion in §2.1, we have that the allowable transactions in S_t can only consume up to some resource limit b, which means that:

$$0 \leq A_t x_t^\star(p) \leq b,$$

from problem (3). Additionally, since $\ell_t(y) = \infty$ unless $0 \leq y \leq b$, as noted in §2.1, then, it is not hard to show that

$$0 \leq y_t^\star(p) \leq b,$$

so we receive the simple bound on the gradient of f_t at p:

$$-b \leq y_t^\star(p) - A_t x_t^\star(p) \leq b.$$

This implies the gradient bound

$$\|\nabla f_t(p)\|_s \leq \|b\|_s$$

for any s-norm. Note that, in the case where $s < \infty$, the norm of b is nondecreasing in the number of elements in b; in other words, the more resources with nonzero limits that we wish to price, the worse this resulting bound is.

2.3 The aggregated dual problem

If the dual function f_t is differentiable, then, since it is convex, finding a minimizer of the dual function corresponds exactly to finding a point $p^\star$ satisfying

$$\nabla f_t(p^\star) = 0.$$

Using (4), this is the same as finding a set of prices $p^\star$ such that

$$A_t x_t^\star(p^\star) = y_t^\star(p^\star).$$

We can interpret the left term as the demand for resources at price $p^\star$ (from users and nodes) and the right term as the supply of resources at price $p^\star$ (from the network). Setting the prices correctly balances the demand and supply of the available resources. One can also show that, when f_t is differentiable and this condition is satisfied, the transactions $x_t^\star(p^\star)$ and $y_t^\star(p^\star)$ exactly solve the original problem (1).

The (aggregated) dual problem. As blockchain designers, we would like to set prices that, at least on average, render the supply and demand of resources (approximately) equal. That is, we would like to find a set of prices that solves the problem:

$$\text{minimize} \quad \sum_{t=1}^{T} f_t(p), \tag{5}$$

over $p \in \mathbf{R}^m$. Finding a price $p^\star$ that minimizes this objective means that, using (1), over the T blocks, we have

$$\sum_{t=1}^{T} A_t x_t^\star(p^\star) = \sum_{t=1}^{T} y_t^\star(p^\star),$$

i.e., that the demand and supply for resources over the T blocks are both equal. Or, in other words, that the market, over all T blocks, clears on aggregate when the prices p are correctly set. (The case where the objective is not differentiable over $p^\star$, but is subdifferentiable, means that there is at least one solution to each subproblem, at optimal prices $p^\star$, for which supply and demand are both equal.)

A basic heuristic. Since the prices p for the blockchain must be set before the blocks are observed, it would be hard to solve (5) without oracular knowledge of the functions f_t. A simple heuristic is then the following: start with some prices p_t, set at blockheight $t-1$, observe what happens at block t with these prices, and update the prices slightly (by, say $\eta > 0$) according to the gradient information:

$$p_{t+1} = p_t - \eta \nabla f_t(p_t). \tag{6}$$

(If f_t is not differentiable at p_t, then replace the gradient with a subgradient.) From before, we know that the gradient at block t is observable at the prices set in the previous block, p_t. This heuristic has roughly the "right behavior": if our prices are too low for a certain resource, say i, then the demand at these prices exceeds the expected supply, so

$$(y_t(p_t))_i > (A_t x_t^\star(p_t))_i.$$

From the update step, this would increase the price $(p_t)_i$ of resource i proportional to this amount. (The opposite is true if the prices at time $t-1$ were set too high.) See [19] for additional discussion.

A multiplicative update. Of course, we can also imagine any number of other update rules. For example, we may use a multiplicative update:

$$p_{t+1} = p_t \circ \exp(-\eta \nabla f_t(p_t)), \tag{7}$$

where $\circ$ denotes the elementwise (Hadamard) product. This update ensures that the prices are always nonnegative; though the previous update (6) can be modified slightly to accommodate this as well. We note that this update rule, when there is one resource, corresponds exactly to the EIP-4844 update found in Ethereum today.

Discussion. Of course, the next natural question is: just how well do these heuristics perform? We will show in the remainder of the paper that the average gap between the objective values provided by this heuristic and those given by setting the optimal prices $p^\star$ (which clear the market on average; *i.e.*, those that solve problem (5)) tends to zero as the number of blocks in question, T, grows:

$$\frac{1}{T}\left(\sum_{t=1}^{T} f_t(p_t) - \sum_{t=1}^{T} f_t(p^\star)\right) \leq \frac{C}{\sqrt{T}}, \tag{8}$$

for any (reasonable) choice of p. Perhaps the most surprising part is that we will show that this is true *even when the allowable transactions S_t, resource utilizations A_t, and the welfare q_t are controlled by an adversary.* We will also show that these algorithms are essentially optimal: no better rate, in terms of T, is possible for any algorithm with such an adversary. The fact that this is possible, along with the original bounds for such problems, come from a long line of work in online convex optimization [35], which we present (and adapt to our specific scenario) in the next section.

3 Bounding the gap

From §2, the main quantity we wish to bound is the difference between the algorithm's performance (*i.e.*, the aggregate performance of the p_t) against the performance of the aggregate market clearing price (*i.e.*, the price $p^\star$ that minimizes (5)):

$$R = \sum_{i=1}^{T} f_t(p_t) - \sum_{i=1}^{T} f_t(p^\star). \tag{9}$$

This quantity R is called the *regret* of the algorithm. Bounding this regret for a variety of algorithms for choosing p_t is the main goal of the field of online algorithms, for which there are many great resources, *cf.*, [35, 23]. A bound on this quantity R would then also give us a bound of the form of (8) given by R/T, which is called the *average regret*. We will give a very short and self-contained exposition of online convex optimization, tailored to our special case, in the what remains of this section and in the next section.

Discussion. Note that this comparison is interesting for a few reasons. First, the price $p^\star$ can only be computed if all of the dual functions f_t are known in advance, which would require knowledge that the network does not have. (On the other hand, updating the prices p_t via a rule such as (6) is very easy to implement in practice.) The second observation is that it need not be true that p_t approaches $p^\star$, except in very special cases, even when the regret is low. (One case where this is true is, for example, when the $f_t = f$, for some fixed function f, which is a common simplification in some analyses of dynamic fees, or when the f_t are i.i.d. drawn from some fixed distribution and strongly convex.)

Linearization. Since the functions f_t are convex, then, by definition [10, §3.1.3] we have that

$$f_t(p^\star) \geq f_t(p_t) + \nabla f_t(p_t)^T (p^\star - p_t).$$

Rearranging, this gives an upper bound on the difference in objective value at time t,

$$f_t(p_t) - f_t(p^\star) \leq \nabla f_t(p_t)^T (p_t - p^\star).$$

For the remainder of the paper, we will write $g_t = \nabla f_t(p_t)$ for notational convenience. (If f_t is not differentiable at p_t, then any subgradient will suffice.) Summing both sides over t then gives

$$R \leq \sum_{i=1}^{T} g_t^T (p_t - p^\star). \tag{10}$$

In other words, if we can find a reasonable upper bound for the linear term on the right hand side, we can give a bound on the regret R, or the average regret R/T.

Results. The main point of this section will be to use bound (10), along with a simple framework to show that the linear update rule (6), with appropriately chosen step size $\eta > 0$ has average regret bounded from above by

$$\frac{R}{T} = \frac{1}{T}\left(\sum_{i=1}^{T} f_t(p_t) - \sum_{i=1}^{T} f_t(p^\star)\right) \leq \frac{BM}{\sqrt{T}},$$

where $\|b\|_2 \leq B$ is an upper bound on the norm of the resource limits, while $\|p^\star\|_2 \leq M$ is a bound on the optimal price. In the case that the adversary is stochastic (*i.e.*, if the f_t are random functions, uniformly and identically drawn under certain conditions we state below) there is a much stronger result. Here, the time-averaged price $\bar{p} = (1/T)\sum_{t=1}^{T} p_t$ converges to the optimal value in expectation in that

$$\mathbf{E}\left[f(\bar{p}) - f(p^\star)\right] \leq \frac{B^2 M}{\sqrt{T}},$$

using again rule (6), where $f(p) = \mathbf{E}[f_1(p)]$, for any fixed p. (We can, of course, replace f_1 with any other f_t.) This also implies that, if the f_t are almost surely μ-strongly convex, for some $\mu > 0$, then $\bar{p}$ converges to the optimal price $p^\star$ in expectation. Finally, if the update used is the multiplicative rule of (7), then the average regret can be bounded from above by

$$\frac{R}{T} \leq BM\sqrt{\frac{m\log(M/\varepsilon)}{2T}}.$$

Here, we define $\|p^\star\|_\infty \leq M$, while $B = \|b\|_2$, as before, while $\varepsilon \leq 1$ is a lower bound on the price used in the first iteration of the rule, $p_0 \geq \varepsilon\mathbf{1}$, where $\mathbf{1}$ is the all-ones vector and the inequality is elementwise.

3.1 Choice functions and updates

We will focus on the two heuristics presented in §2.3 to update the prices and show that, for each, the regret is low. To do this, we rewrite the algorithms as a special case of a general update rule, sometimes known as FTRL or Follow The Regularized Leader [35, 23], which we will then use to find a bound on the regret. The presentation here is a slight variation of the lovely theorem and proof of [24, §2].

Choice function. Given some *choice function* $F : \mathbf{R}^m \to \mathbf{R} \cup \{\infty\}$, we choose the new prices p_{t+1} given the previous gradients $g_1, \ldots, g_t$ by setting

$$p_{t+1} = \nabla F(-\eta(g_1 + \cdots + g_t)), \tag{11}$$

where $\eta > 0$ will be some specific step size to be chosen later. (The gradient can be replaced with any subgradient whenever the function is not differentiable, but is convex.) The one requirement we will have of this function F is that it is "smooth"; in particular that the Hessian of F (if twice-differentiable) is bounded from above by σI, where $\sigma > 0$, in some suitable domain $D \subseteq \mathbf{R}^n$.

Gradient descent. One choice of functions we will use is the norm-squared,

$$F(z) = \frac{1}{2}\|p_0 + z\|_2^2.$$

Since $\nabla F(z) = p_0 + z$, then

$$p_{t+1} = p_0 - \eta(g_1 + \cdots + g_t) = p_t - \eta g_t,$$

which corresponds exactly to the gradient descent rule proposed in (6). This particular choice function is smooth with $\sigma = 1$ for any domain D.

Multiplicative update. The second choice function is

$$F(z) = \sum_{i=1}^{m} (p_0)_i \exp(z_i),$$

which leads to the multiplicative update rule found in (7),

$$p_{t+1} = p_0 \circ \exp(-\eta(g_1 + \cdots + g_t)) = p_t \circ \exp(-\eta g_t).$$

Here $\exp(\cdot)$ is taken elementwise, while $\circ$ denotes the elementwise (Hadamard) product. For the domain $D = \{z \mid \|z\|_\infty \leq \log(M)\}$, this choice function is smooth with $\sigma = \|p_0\|_\infty M$. (We use $\log(M)$ since a bound on the prices of order M would imply a bound on the domain size of order around $\log(M)$ – this is a somewhat technical condition that can be avoided by a slightly more careful choice of F; *cf.*, appendix A.) We provide an alternate interpretation of this rule as both follow-the-regularized-leader and online mirror descent in appendix C. More specifically, these multiplicative update rules emerge from regularization by the nonnegative entropy function. This fact might explain some empirical observations of EIP-1559's behavior.

3.2 Constructing general regret bounds

We will combine two simple observations to give a bound on the linear term of (10). The first is that, if we define the Fenchel conjugate of F over the domain D by

$$F^*(\tilde{p}) = \sup_{z \in D} (\tilde{p}^T z - F(z)),$$

then we have the definitional inequality

$$z^T \tilde{p} - F^*(\tilde{p}) \leq F(z), \tag{12}$$

for any choice of $\tilde{p} \in \mathbf{R}^m$ and $z \in D$. The second is that, since F is smooth with constant σ, we have, by a simple exercise in integration:

$$F(z + z') \leq F(z) + \nabla F(z)^T z' + \frac{\sigma}{2}\|z'\|_2^2.$$

So, given a sequence of $g_1, \ldots, g_T$, by applying this inequality repeatedly and using the definition of the prices p_t given in (11), we have

$$F(-\eta(g_1 + \cdots + g_T)) \leq F(0) - \eta \sum_{t=1}^{T} g_t^T p_t + \frac{\sigma\eta^2}{2} \sum_{t=1}^{T} \|g_t\|_2^2.$$

Finally, setting $z = -\eta(g_1 + \cdots + g_T)$ in (12) and using the above inequality gives

$$\sum_{t=1}^{T} g_t^T (p_t - \tilde{p}) \leq \frac{F(0) + F^*(\tilde{p})}{\eta} + \frac{\sigma\eta}{2} \sum_{t=1}^{T} \|g_t\|_2^2.$$

Since the gradients g_t are bounded from above by $\|g_t\|_2 \leq \|b\|_2 = B$, this leads to the following bound on the linearized quantity:

$$\sum_{t=1}^{T} g_t^T (p_t - \tilde{p}) \leq B\sqrt{\sigma T(F(0) + F^*(\tilde{p}))}, \tag{13}$$

by choosing the step size

$$\eta = \sqrt{\frac{F(0) + F^*(\tilde{p})}{\sigma T B^2/2}},$$

for any choice of $\tilde{p}$. Since this is an online algorithm which may run indefinitely, we may choose T to trade off between responsiveness of the system and regret. In particular, as T becomes large, our regret bound gets tighter (roughly going as $\sqrt{T}$) but the system's responsiveness to events becomes slower and, depending on the system's implementation, the "optimal fixed baseline" of $p^\star$ becomes less realistic.

Regret bound. Using (13), we can now give a bound on the regret R using the inequality of (10). More specifically, if we know that the optimal aggregate clearing price $p^\star$ lies in some set P, then, using (13), we have

$$R \leq B\sqrt{\sigma T(F(0) + F^*(p^\star))} \leq B\sqrt{\sigma T(F(0) + \sup_{p\in P} F^*(p))},$$

or, alternatively, that the average regret over that time horizon, R/T, is

$$\frac{R}{T} \leq B\sqrt{\frac{\sigma(F(0) + \sup_{p\in P} F^*(p))}{T}}. \tag{14}$$

3.3 Regret bounds for updates

Given the set up above, we are now ready to give bounds on the regret for both the gradient-descent-like update rule of (6) and the multiplicative update of (7).

Norm-squared choice function. For the gradient-descent-like rule (6), we have the choice function $F(z) = \frac{1}{2}\|p_0 + z\|_2^2$. Applying the bound (13) gives, since $F(0) = \|p_0\|_2^2$ and $\sigma = 1$ for this choice function (from §3.1):

$$\frac{1}{T}\sum_{t=1}^{T} g_t^T (p_t - \tilde{p}) \leq B\sqrt{\frac{\|p_0\|_2^2 + F^*(\tilde{p})}{2T}}.$$

It is not hard to show that $F^*(\tilde{p}) = \frac{1}{2}\|\tilde{p}\|_2^2$, and, assuming that we have a bound $\|p^\star\|_2 \leq M$ (and similarly for p_0), then the average regret is bounded by (14),

$$\frac{R}{T} \leq \frac{BM}{\sqrt{T}}.$$

Exponential choice function. In the multiplicative update rule (7), the exponential choice function is given by $F(z) = \sum_{i=1}^m (p_0)_i \exp(z_i)$ has $F(0) = \|p_0\|_1$ and $\sigma = M$ such that, again, using bound (13),

$$\frac{1}{T}\sum_{t=1}^T g_t^T(p_t - \tilde{p}) \leq B\sqrt{\frac{M(\|p_0\|_1 + F^*(\tilde{p}))}{2T}}. \tag{15}$$

It is also not hard to show that the conjugate of F is

$$F^*(\tilde{p}) = \sum_{i=1}^m \tilde{p}_i \log \frac{\tilde{p}_i}{(p_0)_i} - \tilde{p}_i,$$

and, if we have a bound that $0 \leq \tilde{p} \leq M\mathbf{1}$ and $\varepsilon\mathbf{1} \leq p_0 \leq M\mathbf{1}$ for $\varepsilon \leq 1$, then

$$\sup_{\|\tilde{p}\|_\infty \leq M} F^*(\tilde{p}) \leq m \max\{M\log(M/\varepsilon) - M, 0\},$$

which follows from the fact that F^* is separable and convex so is maximized at the boundary of the domain. For simplicity, we will assume that $M \geq e$, in which case the first term in the max is nonnegative. Putting it all together, we know that

$$\|p_0\|_1 \leq mM \quad \text{and} \quad F^*(p^\star) \leq m(M\log(M/\varepsilon) - M).$$

Plugging these values into (15) gives

$$\frac{R}{T} \leq BM\sqrt{\frac{m\log(M/\varepsilon)}{2T}},$$

where, from before, M is the maximum price allowed for any one resource, m is the number of resources, and ε is the minimum starting price. The latter condition cannot be dropped since, if $p_0 = 0$, no update would ever be able to increase the price, while on the order of $\sim \log(1/\varepsilon)$ updates are needed to scale the price p_0 to any constant. A sufficient condition to ensure that $p_0 \geq \varepsilon\mathbf{1}$ is that there is always some minimal demand for blockspace at a small enough price, as in, *e.g.*, [19, §3.2].

3.4 A stochastic model

In this subsection, we consider a stochastic setting instead of the adversarial one above, which will give us stronger results on the convergence of the prices p_t. In particular, given a probability space with sample space Ω and distribution P, we assume that a sample $\omega_t \in \Omega$ determines the parameters of the block building problem at time t, so that,

$$q_t = \tilde{q}(\omega_t), \quad S_t = \tilde{S}(\omega_t), \quad \ell_t(\cdot) = \tilde{\ell}(\cdot, \omega_t), \quad A_t = \tilde{A}(\omega_t),$$

where the welfare $\tilde{q}$, feasible set $\tilde{S}$, losses $\ell(\cdot, \omega)$ (viewed as a function over ω), and consumption matrices $\tilde{A}$ are random variables over Ω. Given a sample $\omega \in \Omega$ and a price vector p, the utility in scenario ω for price p is given by the concave function

$$\tilde{f}(p, \omega) = \sup_y \left(p^T y - \tilde{\ell}(y, \omega)\right) + \sup_{x \in \mathbf{conv}(\tilde{S}(\omega))} (\tilde{q}(\omega) - \tilde{A}(\omega)^T p)^T x.$$

(In other words, we replace the problem data in the definition of the dual function (2) with the corresponding sampled versions.) Here, we will show that, in a certain sense, the time-averaged price of the algorithm minimizes the expected utility,

$$f(p) = \mathbf{E}[\tilde{f}(p, \omega)]. \tag{16}$$

For later, we will define $\tilde{p}^\star \in \operatorname{argmin} f$. Unlike in the previous set up for adversarial functions, in this problem, the observed functions are random variables $f_t(p) = \tilde{f}(p, \omega_t)$, where the ω_t are drawn i.i.d. according to P. So, in this setting, the gradient update rule of (6),

$$p_{t+1} = p_t - \eta \nabla f_t(p_t),$$

corresponds to an instance of stochastic gradient descent with stepsize given by $\eta > 0$.

Result. A standard result in stochastic gradient descent from, say [21, Thm. 9.5], is that,

$$\mathbf{E}[f(\bar{p})] - f(\tilde{p}^\star) \leq \frac{2M^2}{\eta T} + \frac{\eta B^4}{2}.$$

Here, $B = \|b\|_2$ and $\|\tilde{p}^\star\|_2 \leq M$ is a bound on the optimal price $\tilde{p}^\star$ that minimizes the expected utility f given in (16). (We also assume that the initial price $\|p_0\|_2 \leq M$.) Finally, the time-averaged price $\bar{p}$ is defined

$$\bar{p} = \frac{1}{T} \sum_{t=1}^{T} p_t.$$

Setting $\eta = 2M/B^2\sqrt{T}$ then gives the bound:

$$\mathbf{E}[f(\bar{p})] - f(\tilde{p}^\star) \leq \frac{B^2 M}{\sqrt{T}}.$$

Extensions. There is an immediate extension to this result. The first is to note that, by assumption, this result relies on i.i.d. draws of the sample ω_t, but this can be generalized considerably. If we define ϵ_t to be the noise in the gradient estimate at block t, *i.e.*,

$$\epsilon_t = \nabla f(p_t) - \nabla_p \tilde{f}(p_t, \omega_t),$$

then the above result requires that $\{\epsilon_t\}$ be zero mean and i.i.d. Under suitable conditions, this can be relaxed to the $\{\epsilon_t\}$ being a martingale difference sequence [22, §5.2].

3.5 Discussion

With only the basic assumptions that (a) there is some bound on the maximum price $p^\star$ that an adversary may choose and (b) some hard constraint on the block size, which is enforced by the validators, we achieve average regret that scales as $1/\sqrt{T}$ for either update rule, even when the transactions and welfare, which are included in the functions f_t, are chosen adversarially. An interesting point is that this shows why a hard constraint on the resources consumed per-block is a good idea: it bounds any adversary's ability to construct arbitrarily-sized blocks which could make the regret of most reasonable rules quite large. The restriction given in (a), that the adversary has bounded prices, seems strong, but is implied by a number of simple statements: for example, it suffices that the adversary has finite budget and must pay $p_t^T A_t x_t^\star(p_t)$ for each transaction, which is, in fact, a much stronger requirement. Two interesting questions for future research are: first, is there a more natural model for an adversary in the blockchain setting than the essentially all-powerful one provided here? And, second, given such an adversary, can we achieve better results?

Time-independent step sizes. In this model, we assumed that the step size η is fixed and can depend on the total length of observed time period, T. Indeed, the step size trades off the ability of the prices p to "react" to observed gradients g_t with the average regret. (In particular, as T becomes large, the average regret becomes small, but η must be similarly small and must remain so for the time period in observation.) It is not hard to show that the regret bound in both the adversarial (§3.3) and stochastic case (§3.4) above for the gradient descent rule (6) (*i.e.*, the one derived from the norm-square choice function) also carries through in the case where the step sizes η are decreasing with respect to time, to get an update rule of the form

$$x_{t+1} = x_t - \eta_t g_t. \tag{17}$$

Here, $\eta_t = C/\sqrt{t}$ and $C > 0$ is some appropriately chosen constant that, importantly, does not depend on the observation window T, but only on the known bounds for the price $\|p\|_2 \leq M$ and the maximum resource utilizations $\|b\|_2 \leq B$. The regret for this algorithm, under the same assumptions as §3.3 for the norm-squared choice function, is no more than $\sqrt{2}BM\sqrt{T}$ in the adversarial case; see [23, §3.1] for a proof. (The bound for the stochastic case is more refined, but see, *e.g.*, [21, Thm. 5.3] for the statement.) More generally, any sequence $\{\alpha_t\}$ which asymptotically satisfies

$$\sum_{t=1}^{T} \alpha_t = O(\sqrt{T})$$

and $1/\alpha_T = O(\sqrt{T})$ has a similar guarantee, up to a potentially different constant.

Strong convexity. Indeed, if the step sizes of the algorithm are allowed to vary with respect to time, we can get much stronger regret guarantees in the case that the conjugate of the loss $\ell_t^\star$ is strongly convex, using the same update rule (17). More specifically, if $\ell_t^\star$ is μ-strongly convex, then we can achieve *logarithmic* regret that, additionally, does not depend on there being any bound on the optimal price $p^\star$. (One case where $\ell_t^\star$ is strongly convex is, *e.g.*, when $\ell_t(y) = (1/2)\|(y - b^\star)_+\|_2^2$ for some target utilizations $b^\star \in \mathbf{R}_+^m$ and y is bounded by the maximum resource utilizations $y \leq b$.) In this particular case, we have that for the update rule (17), where the step sizes are $\eta_t = \mu/t$, the regret is no larger than $B^2 \log(T)/\mu$, making the average regret quite small: no more than $B^2 \log(T)/(\mu T)$. (For a proof of this, see [23, §3.3.1].)

4 Lower bound

In this section, we will show that, up to a scalar constant, the bounds provided in §3.3 are essentially optimal for any algorithm. In particular, we will show that any rule to choose the prices p_{t+1}, based on the previous t observed gradients $g_1, \ldots, g_t$, achieves, at best, regret on the order of $1/\sqrt{T}$, for an adversary who can control the transactions appearing in a block, subject to the constraints of the set S containing some maximum resource usage limit $b \in \mathbf{R}_+^m$, for a commonly-used loss. As in the previous section, we assume that the optimal price $p^\star \in \mathbf{R}^m$ for the aggregated program is bounded, $\|p^\star\|_\infty \leq M$ for some value $M > 0$.

4.1 Construction

The construction we provide here is very simple: the adversary is purely stochastic and picks the resource utilizations, up to the bound, from a simple distribution, independently of all of the previous gradients, such that any algorithm achieves 0 objective value, while the best possible price p achieves an objective of $-C\sqrt{T}$ for some constant $C > 0$.

Loss function and set. We construct a lower bound for any algorithm which attempts to minimize the dual problem corresponding to the commonly-used loss

$$\ell(y) = \begin{cases} 0 & 0 \leq y \leq b^\star \\ +\infty & \text{otherwise,} \end{cases} \tag{18}$$

where $b^\star \in \mathbf{R}^m_+$ is the resource utilization target. Using this definition, it is not hard to show that the Fenchel conjugate of ℓ is

$$\ell^*(p) = (b^\star)^T (p)_+,$$

where $((p)_+)_i = \max\{p_i, 0\}$ denotes the "positive part" of p, applied elementwise. The set $S_t \subseteq \{0,1\}^{n_t}$ we assume satisfies

$$S_t \subseteq \{x_t \in \mathbf{R}^{n_t}_+ \mid A_t x_t \leq b\},$$

for some matrix $A_t \in \mathbf{R}^{m \times n_t}$ and max resource utilization $b \in \mathbf{R}^m_+$. (We note that this set up reduces exactly to the resource metering of EIP-1559 when the number of resources $m = 1$, where n_t is the number of transactions in the mempool of the block builder and b is the block limit.)

Adversary. Consider a (stochastic) adversary who is able to control the amount of resources used by the network. The adversary picks the welfare to be $q_t = 0$ and picks transactions A_t such that the welfare maximization problem (3) satisfies, at price p_t,

$$A_t x_t^\star = b^\star - \varepsilon_t \circ \tilde{b},$$

where $\varepsilon_t \sim \{\pm 1\}^m$ is uniformly randomly chosen (independently, for each $t = 1, \ldots, T$) and

$$\tilde{b}_i = \min\{b_i^\star, b_i - b_i^\star\},$$

for $i = 1, \ldots, m$. (The adversary can achieve this by constructing, say, exactly one transaction that is always included with zero welfare and utilizations equal to exactly $b^\star - \varepsilon_t \circ \tilde{b}$; we assume this in what follows.) This gives

$$g_t = y_t^\star - A_t x_t^\star = b^\star - (b^\star - \varepsilon_t \circ \tilde{b}) = \varepsilon_t \circ \tilde{b}. \tag{19}$$

Proof. It suffices to show that, in expectation, the regret is $\Omega(\sqrt{T})$, which means that there is at least one choice of resource consumption trajectories (gradients $\{g_t\}$) that leads to large regret, for any possible choices of prices p_t by an algorithm. First note that the linear bound (13),

$$\sum_{t=1}^T (f_t(p_t) - f_t(p^\star)) \leq \sum_{t=1}^T g_t^T (p_t - p^\star),$$

is met at exact equality for any nonnegative p_t and $p^\star$ using our construction. To see this, note that the transactions were chosen to have zero welfare, $q_t = 0$. Using the definition of ℓ (18) means that, for any $p, p_t \geq 0$:

$$f_t(p) = \ell_t^*(p) + h_t(p) = (b^\star)^T p + (q_t - A_t^T x_t^\star(p_t))^T p = (b^\star - A_t x_t^\star(p_t))^T p = g_t^T p,$$

for every $t = 1, \ldots, T$. It will then suffice to show that the linear sum

$$\mathbf{E}\left[\sum_{t=1}^{T} g_t^T (p_t - p^\star)\right] \tag{20}$$

is "large" in expectation, for $p^\star$ chosen with knowledge of g_t.

The first term of the loss (20) is zero since

$$\mathbf{E}[g_t^T p_t] = \mathbf{E}[g_t]^T p_t = 0,$$

as p_t can depend only on the observed gradients, $g_1, \ldots, g_{t-1}$. The expected regret then reduces to

$$\mathbf{E}\left[\sum_{t=1}^{T} g_t^T p_t - g_t^T \bar{p}\right] = \mathbf{E}\left[-\bar{p}^T \sum_{t=1}^{T} g_t\right].$$

Since $p^\star$ is chosen after observing the gradients g_t, and we are free to choose any $p^\star$ with $\|p^\star\|_\infty \leq M$ and $p^\star \geq 0$, we then this term is maximized by choosing $p_i^\star = M$ if

$$\sum_{t=1}^{T} (g_t)_i \leq 0,$$

and 0 otherwise, for $i = 1, \ldots, m$. This means that

$$\mathbf{E}\left[-\bar{p}^T \sum_{t=1}^{T} g_t\right] = M\mathbf{E}\left[\mathbf{1}^T \left(-\sum_{t=1}^{T} g_t\right)_+\right],$$

where $((z)_+)_i = \max\{z_i, 0\}$ is the "positive part" of z_i. Using the definition of g_t above (19), we can bound the latter term

$$M\mathbf{E}\left[\mathbf{1}^T \left(-\sum_{t=1}^{T} g_t\right)_+\right] = M(\mathbf{1}^T \tilde{b})\mathbf{E}\left[\left(\sum_{t=1}^{T} \varepsilon_t'\right)_+\right] \geq M(\mathbf{1}^T \tilde{b})C\sqrt{T},$$

where $\varepsilon_t' \sim \{\pm 1\}$ for $t = 1, \ldots, T$ is uniformly and independently drawn and $C \geq 1/24$. The inequality follows from a mechanical (if involved) counting argument, or any number of lower bounds on the expected value of the positive side of a random walk. We reproduce a simple, noncounting proof of this in appendix B. We note that, compared to either of the bounds of §3.3, this is tight up to a twiddle factor of M and a square root factor of m.

5 Conclusion

In this paper, we have shown that two simple update rules: a gradient update rule and a multiplicative update rule, lead to low average regret for the discovery and assignment of prices for blockchain resources, even under adversarial conditions. This implies that, on average, over a long enough timescale, the difference between correctly setting resource prices with oracular knowledge of the future (to roughly match supply and demand over the period of time in question) and these two simple update rules, is very small. We have also shown that, under the assumptions above, this is essentially the best we can hope to do by also showing a matching lower bound. Since Ethereum's EIP-1559 is the single-dimensional special case of the multiplicative rule presented above, the analysis also applies directly, along with the proposed pricing rule for EIP-4844, which is the two-dimensional special case.

Future work. Future potential avenues for research include weaker, but potentially more realistic adversarial models (as the adversary has a lot of power in this particular formulation) which may yield tighter bounds to the expected performance. These include models like that of patient bidders [6], where the changes to the set of transactions at each time step are limited. Additionally, a framework to set the number of resources m in a principled way (*e.g.*, one which delineates the tradeoff between more resources versus lower worst-case regret) may be of practical interest. We note that these types of adversarial resource allocation problems also show up in zero knowledge proof markets [37, 34] and in general compute markets [7]. We suspect our results can be easily extended to and are useful for these settings.

References

1 John Adler. Eip-2242: Transaction postdata, 2019. URL: `https://eips.ethereum.org/EIPS/eip-2242`.

2 John Adler. Multi-threaded data availability on eth 1. Ethresearch, 2019. URL: `https://ethresear.ch/t/multi-threaded-data-availability-on-eth-1/5899`.

3 John Adler. Always has been (or, wait, it's all resource pricing? part 2). EthCC, 2022. URL: `https://www.youtube.com/watch?v=Zq8uwpX39oI`.

4 Guillermo Angeris. A non-counting lower bound for the expected distance of a simple random walk. `https://angeris.github.io/blog/content/simple-proof-rademacher/`, 2023.

5 Itai Ashlagi, Jacob Leshno, Pengyu Qian, and Amin Saberi. Price discovery in waiting lists: A connection to stochastic gradient descent. *Available at SSRN 4192003*, 2022.

6 Moshe Babaioff and Noam Nisan. On the optimality of eip-1559 for patient bidders (draft–comments welcome), 2024.

7 Maryam Bahrani and Naveen Durvasula. Resonance: Transaction fees for heterogeneous computation. *arXiv preprint arXiv:2411.11789*, 2024. `doi:10.48550/arXiv.2411.11789`.

8 Maryam Bahrani, Pranav Garimidi, and Tim Roughgarden. Transaction fee mechanism design with active block producers. *arXiv preprint arXiv:2307.01686*, 2023. `doi:10.48550/arXiv.2307.01686`.

9 Soumya Basu, David Easley, Maureen O'Hara, and Emin Gün Sirer. Stablefees: A predictable fee market for cryptocurrencies. *Management Science*, 2023.

10 Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK; New York, 2004.

11 Sébastien Bubeck, Nikhil R Devanur, Zhiyi Huang, and Rad Niazadeh. Multi-scale online learning: Theory and applications to online auctions and pricing. *Journal of Machine Learning Research*, 20(62):1–37, 2019. URL: `https://jmlr.org/papers/v20/17-498.html`.

12 Vitalik Buterin. Blockchain resource pricing, 2018.

13 Vitalik Buterin. Multidimensional eip 1559. *ethresear.ch*, 2022.

14 Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. Eip-1559: Fee market change for eth 1.0 chain, 2019. URL: `https://eips.ethereum.org/EIPS/eip-1559`.

15 Vitalik Buterin and Ansgar Dietrichs. Eip-4488: Transaction calldata gas cost reduction with total calldata limit, 2021. URL: `https://eips.ethereum.org/EIPS/eip-4488`.

16 Hao Chung and Elaine Shi. *Foundations of Transaction Fee Mechanism Design*, pages 3856–3899. SIAM, 2023. `doi:10.1137/1.9781611977554.ch150`.

17 Richard Cole and Lisa Fleischer. Fast-converging tatonnement algorithms for one-time and ongoing market problems. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 315–324, 2008. `doi:10.1145/1374376.1374422`.

18 Davide Crapis, Ciamac C Moallemi, and Shouqiao Wang. Optimal dynamic fees for blockchain resources. *arXiv preprint arXiv:2309.12735*, 2023. `doi:10.48550/arXiv.2309.12735`.

19 Theo Diamandis, Alex Evans, Tarun Chitra, and Guillermo Angeris. Designing Multidimensional Blockchain Fee Markets. In Joseph Bonneau and S. Matthew Weinberg, editors, *5th Conference on Advances in Financial Technologies (AFT 2023)*, volume 282 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:23, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.AFT.2023.4.

20 Matheus V. X. Ferreira, Daniel J. Moroz, David C. Parkes, and Mitchell Stern. Dynamic posted-price mechanisms for the blockchain transaction-fee market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 86–99, 2021. doi:10.1145/3479722.3480991.

21 Guillaume Garrigos and Robert M Gower. Handbook of convergence theorems for (stochastic) gradient methods. *arXiv preprint*, 2023. arXiv:2301.11235.

22 J Harold, G Kushner, and George Yin. Stochastic approximation and recursive algorithm and applications. *Application of Mathematics*, 35(10), 1997.

23 Elad Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016. doi:10.1561/2400000013.

24 Sham M. Kakade, Shai Shalev-Shwartz, and Ambuj Tewari. Regularization techniques for learning with matrices. *Journal of Machine Learning Research*, 13:1865–1890, 2009. URL: https://api.semanticscholar.org/CorpusID:8477893.

25 Stefanos Leonardos, Barnabé Monnot, Daniël Reijsbergen, Efstratios Skoulakis, and Georgios Piliouras. Dynamical analysis of the eip-1559 ethereum fee market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 114–126, 2021. doi:10.1145/3479722.3480993.

26 Stefanos Leonardos, Daniël Reijsbergen, Barnabé Monnot, and Georgios Piliouras. Optimality despite chaos in fee markets. In *Financial Cryptography and Data Security: 27th International Conference, FC 2023, Bol, Brač, Croatia, May 1–5, 2023, Revised Selected Papers, Part II*, pages 346–362, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-3-031-47751-5_20.

27 Yulin Liu, Yuxuan Lu, Kartik Nayak, Fan Zhang, Luyao Zhang, and Yinhong Zhao. Empirical analysis of eip-1559: Transaction fees, waiting times, and consensus security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pages 2099–2113, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3548606.3559341.

28 Brendan McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 525–533. JMLR Workshop and Conference Proceedings, 2011. URL: http://proceedings.mlr.press/v15/mcmahan11b/mcmahan11b.pdf.

29 Patrick O'Grady. Multidimensional fees + fifo mempool. https://github.com/ava-labs/hypersdk/pull/352, 2023.

30 Penumbra Team. Groundwork for multidimensional gas fees. https://github.com/penumbra-zone/penumbra/issues/2970, 2023.

31 Daniël Reijsbergen, Shyam Sridhar, Barnabé Monnot, Stefanos Leonardos, Stratis Skoulakis, and Georgios Piliouras. Transaction fees on a honeymoon: Ethereum's eip-1559 one month later. In *2021 IEEE International Conference on Blockchain (Blockchain)*, pages 196–204. IEEE, 2021. doi:10.1109/BLOCKCHAIN53845.2021.00034.

32 Aaron Roth, Aleksandrs Slivkins, Jonathan Ullman, and Zhiwei Steven Wu. Multidimensional dynamic pricing for welfare maximization. *ACM Transactions on Economics and Computation (TEAC)*, 8(1):1–35, 2020. doi:10.1145/3381527.

33 Tim Roughgarden. Transaction fee mechanism design. *SIGecom Exch.*, 19(1):52–55, July 2021. doi:10.1145/3476436.3476445.

34 Uma Roy, John Guibas, Mallesh Pai, Kshitij Kulkarni, and Dan Robinson. Provewith.us. https://www.provewith.us/, 2024.

35 Shai Shalev-Shwartz et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012. doi:10.1561/2200000018.

36 Hirofumi Uzawa. Walras' tatonnement in the theory of exchange. *The Review of Economic Studies*, 27(3):182–194, 1960.

37 Wenhao Wang, Lulu Zhou, Aviv Yaish, Fan Zhang, Ben Fisch, and Benjamin Livshits. Mechanism design for zk-rollup prover markets. *arXiv preprint arXiv:2404.06495*, 2024. doi:10.48550/arXiv.2404.06495.

38 Anatoly Yakovenko and Tao Zhu. Simd-110: Exponential fee for write lock accounts, 2024. URL: https://github.com/solana-foundation/solana-improvement-documents/pull/110.

A Alternative multiplicative choice function

Instead of setting a bound on the gradients such that the multiplicative update choice function of §3.1 has arguments $z \in \mathbf{R}^m$ with $\|z\|_\infty \leq \log(M)$, we can instead choose the function F in the following way: if we know that $\|p\|_\infty \leq M$ (or if we wish to have a rule that satisfies this condition) then we can set the function F to be, instead,

$$F(z) = \sum_{i=1}^{m} \min\{(p_0)_i \exp(z_i), M(z_i - \log(M/(p_0)_i) + 1)\}.$$

We then have that

$$(\nabla F(z))_i = \begin{cases} (p_0)_i \exp(z_i) & z_i \leq \log(M/(p_0)_i) \\ M & \text{otherwise.} \end{cases}$$

It is not hard to show that F is indeed convex by noting that it is both separable and the derivative of any one term is nondecreasing by construction. Additionally, F is smooth with constant mM, which is easy to see by noting that, again, F is separable and the second derivative of each term is bounded from above by M. By construction, we will have that any

$$\tilde{p} \in \nabla F(z),$$

satisfies $\|\tilde{p}\|_\infty \leq M$, as required, while

$$F^*(p) = \sum_{i=1}^{m} \left(p_i \log\left(\frac{p_i}{(p_0)_i} \right) - p_i \right),$$

whenever $0 \leq p \leq M\mathbf{1}$ (where we define $0 \log 0 = 0$ for convenience).

B Noncounting lower bound

In this appendix, we give a simple lower bound for the expected value of a random walk. More specifically, given $\varepsilon'_t \in \{\pm 1\}$ chosen uniformly and independently for $t = 1, \ldots, T$, we will show that there exists a constant $C' \geq 1/12$ such that

$$\mathbf{E}\left[\left| \sum_{t=1}^{T} \varepsilon'_t \right| \right] \geq C' \sqrt{T}.$$

Since ε'_t is symmetric about 0 (*i.e.*, ε'_t and $-\varepsilon'_t$ have the same distribution) then the positive part would satisfy

$$\mathbf{E}\left[\left(\sum_{t=1}^{T} \varepsilon'_t \right)_+ \right] \geq \frac{C'}{2} \sqrt{T} \geq \frac{1}{24} \sqrt{T},$$

given the above. This material is a shorter version of [4].

Proof. For succinctness, we will write the sum as

$$X = \sum_{t=1}^{T} \varepsilon'_t.$$

The odd moments of X vanish, while the second and fourth moments are

$$\mathbf{E}[X^2] = T, \quad \mathbf{E}[X^4] = 3T^2 - 2T \leq 3T^2.$$

which is easy to see from expanding the sum and noting that any term containing an odd power of ε'_t has expectation equal to zero. The bound will come from the simple observation that, for any $a \geq 0$,

$$\mathbf{E}[|X|] \geq a \Pr(|X| \geq a). \tag{21}$$

We bound the second term in the inequality from below by expanding the second moment,

$$\mathbf{E}[X^2] \leq a^2 + \mathbf{E}[X^2 \mathbf{1}_{|X|\geq a}] \leq a^2 + \sqrt{\mathbf{E}[X^4]}\sqrt{\mathbf{E}[\mathbf{1}_{|X|\geq a}]} = a^2 + \sqrt{\mathbf{E}[X^4]}\sqrt{\Pr(|X| \geq a)}, \tag{22}$$

for any fixed constant $a \geq 0$, where $\mathbf{1}_{|X|\geq a}$ is the zero-one indicator function for the event $|X| \geq a$. Here, the first inequality follows from the pointwise inequality

$$X^2 \leq a^2 \mathbf{1}_{|X|<a} + X^2 \mathbf{1}_{|X|\geq a},$$

while the second inequality follows from an application of Cauchy—Schwarz to the latter term and the fact that $\mathbf{1}^2_{|X|\geq a} = \mathbf{1}_{|X|\geq a}$. Rearranging the second moment bound (22) above, we find

$$\Pr(|X| \geq a) \geq \frac{(\mathbf{E}[X^2] - a^2)^2}{\mathbf{E}[X^4]} \geq \frac{(T - a^2)^2}{3T^2},$$

where we have implicitly assumed that $a^2 \leq T$. Plugging this in to the basic bound (21) on the expectation gives, for any $0 \leq a \leq \sqrt{T}$,

$$\mathbf{E}[|X|] \geq a \frac{(T - a^2)^2}{3T^2}.$$

The right hand quantity is maximized at $a = \sqrt{T/5}$ (which is easy to see by using the first-order optimality conditions applied to a) so we get

$$\mathbf{E}[|X|] \geq \frac{16}{75\sqrt{5}}\sqrt{T} \geq \frac{1}{12}\sqrt{T},$$

which means that $C' \geq 1/12$, as required. We note that this constant is loose, as a simple counting argument shows that, in fact

$$C' \sim \sqrt{2/\pi},$$

for T large enough, but this bound is enough to show the lower bound holds up to a constant.

C Follow-the-regularized-leader interpretation

We may interpret the price update rule (11) as follow-the-regularized-leader (FTRL), with the regularizer defined by the choice function. The FTRL update rule is given by

$$p_{t+1} = \operatorname*{argmin}_{p \in D} \sum_{s=1}^{t} g_s^T p + (1/\eta) R(p),$$

where R is some strongly convex regularizer and D is the effective domain of the $\{f_t\}$. From the optimality conditions, we can write

$$p_{t+1} = (\nabla R)^{-1}(-\eta(g_1 + \cdots + g_t)),$$

assuming the range of $(\nabla R)^{-1}$ is contained in the domain D. (Note that $\nabla R : D \to \mathbf{R}^n$ is an invertible function since R is strongly convex.) Comparing this equation with the choice function-based price update,

$$p_{t+1} = \nabla F(-\eta(g_1 + \cdots + g_t)),$$

we immediately see the relationship between the choice function F and the regularizer R, namely that $\nabla R = (\nabla F)^{-1}$ and vice versa. (Equivalently, this is saying that R and F are Fenchel conjugates of one another.) We may alternatively interpret the update rule as online mirror descent with Bregman divergence defined by the regularizer. See [28] for details.

EIP 1559-style price updates. This connection allows us to interpret the multiplicative update rule (7) as FTRL. Consider the loss function $\ell_t(y) = I(y = b^\star)$ for some target gas usage $b^\star \in \mathbf{R}_+$. The matrix A_t is simply a row vector of the gas costs for each transaction. The gradient at block t, which has gas price p_t, is then

$$g_t = \nabla f_t(p_t) = b^\star - A_t x_t^\star(p_t).$$

From section 3.1, we write the EIP-1559-style multiplicative update rule with $p_0 = 1$ as

$$p_{t+1} = p_t \circ \exp(-\eta g_t) = \nabla F(-\eta(g_1 + \cdots + g_t)).$$

As discussed above, we may also interpret this update rule as FTRL with the regularizer

$$R(p) = p\log(p) - p.$$

This regularizer achieves its minimum at $p = p_0 = 1$. Note that it is not symmetric around its minimum; price decreases are more quickly penalized than price increases.

EIP 1559 as online mirror descent. Alternatively, we may interpret the EIP-1559-style multiplicative price update rule as online mirror descent with Bregman divergence defined by the nonnegative entropy $p\log(p)$. This update rule is given by

$$p_{t+1} = \operatorname*{argmin}_{p \in \mathbf{dom}\, f} f(p_t) + g_t^T(p - p_t) + \frac{1}{\eta} D(p,\, p_t),$$

where the first two terms are a linear approximation to the convex function f and the last term is the Bregman divergence regularizer

$$D(p,\, p_t) = p\log(p) - p_t\log(p_t) - (\log(p_t) + 1)(p - p_t).$$

Dropping constant terms, we can rewrite the update rule as

$$\begin{aligned} p_{t+1} &= \operatorname*{argmin}_{p \in \mathbf{dom}\, f} g_t^T p + \frac{1}{\eta}\left(p\log(p/p_t) - p\right) \\ &= p_t \circ \exp(-\eta g_t). \end{aligned}$$

Note that the regularizer $D(p,\, p_t)$ achieves its minimum at $p = p_t$ and penalizes price decreases more than price increases. (See figure 1.) This asymmetric penalty may explain some past observations, *e.g.*, that EIP 1559 has a positive overshoot [26].

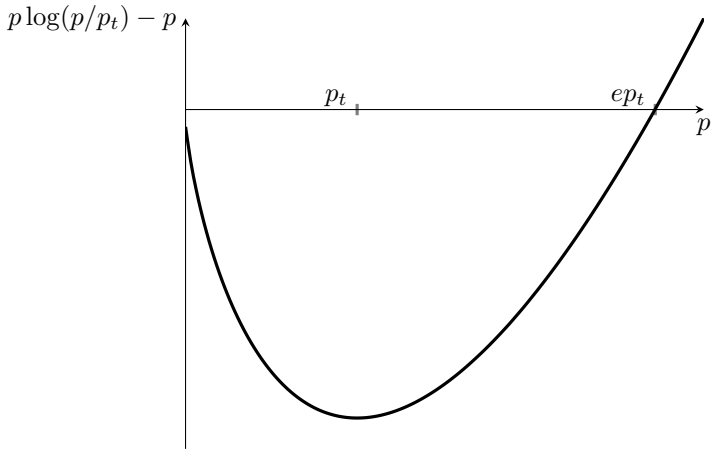


Figure 1 Bregman divergence corresponding to the nonnegative entropy function, which is used in the EIP 1559-style update rule.

The Exchange Problem

Mohit Garg
Indian Institute of Science, Bengaluru, India

Suneel Sarswat
National Institute of Securities Markets, Mumbai, India

Abstract

Auctions are widely used in exchanges to match buy and sell requests. Once the buyers and sellers place their requests, the exchange determines how these requests are to be matched. The two most popular objectives used while determining the matching are maximizing volume *with dynamic pricing* and maximizing volume *at a uniform price*. In this work, we study the algorithmic complexity of the problems arising from these matching tasks.

For **dynamic-price matching**, we establish a lower bound of $\Omega(n \log n)$ on the running time, thereby proving that the currently best-known $O(n \log n)$ algorithm is time-optimal. In contrast, for **uniform-price matching**, we present a **linear-time** algorithm, improving upon previous methods that require $O(n \log n)$ time to match n requests.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Applied computing → Electronic commerce

Keywords and phrases Exchanges, Double Auctions, Matching Algorithms, Element Distinctness, Time Complexity

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.25

Funding *Mohit Garg*: Supported in part by a fellowship from the Walmart Center for Tech Excellence at IISc (CSR Grant WMGT-23-0001).

Acknowledgements The authors express their gratitude to Jaikumar Radhakrishnan for bringing the element distinctness problem to their attention and for engaging in insightful discussions that enriched their understanding of it. They are also thankful to him for helping improve the presentation of this work.

1 Introduction

We study problems of the following nature: The input is a list of trade requests from buyers and sellers of a particular product. Each request consists of a price and a quantity. The buyer's request, known as a *bid*, is represented by a pair (p, q), which indicates that the buyer offers to buy a maximum of q units of the product while paying at most p per unit. Similarly, a seller's request, known as an *ask*, is also a pair (p, q), which indicates that the seller offers to sell a maximum of q units of the product provided they receive at least p per unit. On receiving the input, we are required to generate a *matching*, which is a collection of *transactions*.[1] A transaction between a bid (p_b, q_b) and an ask (p_a, q_a) consists of a transaction price $p \in [p_a, p_b]$ and a transaction quantity $q \leq \min\{q_b, q_a\}$, indicating that q units of the product change hands between the traders at price p per unit. For each trade request (p, q), the sum of the total transaction quantities of the transactions involving the request must be at most q. The volume of the matching is the total quantity that changes hands. The matching is *uniform* if all transactions occur at a common price. We consider the following two tasks.

[1] We use the term *matching*, which has been traditionally used for this object in the auction theory literature [10, 13, 12, 9, 4], while a term like *flow* might be more suitable for a computer science audience.

© Mohit Garg and Suneel Sarswat;
licensed under Creative Commons License CC-BY 4.0

Task 1: Determine a matching with the largest volume.
Task 2: Determine a uniform matching with the largest volume.

The first task can be solved by formulating it as a max-flow problem. However, due to the underlying problem structure, simpler solutions are available for these tasks. In fact, two simple and almost identical algorithms can be used for these tasks. We describe them below.

Algorithms: These algorithms start with sorting the list of bids in decreasing order of their prices. Next, the list of asks is sorted. For the first task, the sell requests are sorted in decreasing order of their prices. Whereas, for the second task, the sell requests are sorted in increasing order of their prices.
After the sorting step, both algorithms work in linear time as follows. The bid on top of its sorted list (p_b, q_b) is matched with the ask on top of its sorted list (p_a, q_a) if they are compatible, i.e., $p_b \geq p_a$. In this case, a transaction between them is established with quantity $q = \min(q_b, q_a)$; a quantity of q is reduced from their existing quantities of q_b and q_a; finally, the 0 quantity request(s) are deleted from the lists. If the requests are incompatible, the topmost ask in the sorted list of asks is deleted. The above steps are then repeatedly applied until one of the lists becomes empty. Finally, for the first task, for each matched pair: $\langle (p_b, q_b), (p_a, q_a) \rangle$ any transaction price in the interval $[p_a, p_b]$ can be put. For the second task, let the last matched pair in the above process be $\langle (p_b, q_b), (p_a, q_a) \rangle$. Then, for all matched pairs any number in the interval $[p_a, p_b]$ can be chosen as the common transaction price.

Clearly, the above algorithms take time $O(n \log n)$ in the comparison model. We ask the following question.

Do these algorithms have the optimal running time for the above tasks?

The problems we consider in this work are slight generalizations of the above tasks where bids and asks have additional parameters, and the objectives incorporate a notion of *fairness*. These problems arise out of *call auctions*, which form the backbone of many modern trading platforms, most notably stock exchanges, commodities markets, and various electronic marketplaces. Numerous works have focused on the mechanism-design and game-theoretic aspects of such auctions (e.g., see [3] and the citations therein), but despite the fundamental importance and widespread usage of these auctions, the precise computational complexity of determining maximum-volume matchings – either in uniform-price or dynamic-price settings – has remained relatively unexplored, with only a few works that include a runtime analysis of their algorithms ([12, 10, 13]. For the above tasks, we prove the following.

▶ **Theorem 1.** *Any comparison-based algorithm for Task 1 requires $\Omega(n \log n)$ time in the worst-case, where n is the number of trade requests.*

▶ **Theorem 2.** *There exists a comparison-based algorithm for Task 2 that runs in $O(n)$ time, where n is the number of trade requests.*

These findings reveal an unexpected dichotomy. While **Task 1** intrinsically requires sorting, **Task 2** admits a more efficient linear-time solution. This result has not appeared previously, despite its importance to many exchanges' opening and closing routines.

To describe the general problems that we consider, previous results, and our contributions formally, we need the following preliminaries.

1.1 Preliminaries

We now formally describe call auctions which are slight generalizations of the tasks introduced above.

Setup, Definitions, and Notation

A call auction is used at an exchange to match the bids and asks of traders for a particular product. Bids and asks are collectively termed orders. Orders are collected by the exchange for a fixed duration of time, at the end of which the collected orders are simultaneously matched to produce transactions. Each transaction between a bid and an ask consists of a transaction price and a transaction quantity. Each order consists of four attributes that are natural numbers: a unique identification number (id), a unique timestamp, a limit price, and a quantity that is at least one.[2] $\mathsf{timestamp}(w)$ represents the time the order w was received. For an order w, the limit price $\mathsf{price}(w)$ is the minimum (if w is an ask) or maximum (if w is a bid) possible transaction price of a transaction involving w. For an order w, the quantity $\mathsf{qty}(w)$ represents the total quantity offered for trade, i.e., the sum of transaction quantities of transactions involving w in the matching must be at most $\mathsf{qty}(w)$. Given a collection of orders with distinct ids, i.e., given a set of bids and asks with distinct ids, a matching is a collection of transactions that satisfy the natural constraints. Next, we define a few important terms and a matching formally.

Given a bid b and an ask a, we say that b and a are **tradable** if $\mathsf{price}(b) \geq \mathsf{price}(a)$. For a set of transactions M and an order w, $\mathsf{Qty}(w, M)$ denotes the sum of transaction quantities of transactions in M that involve w, and $\mathsf{Vol}(M)$ denotes the sum of all transaction quantities of the transactions in M. With slight abuse of notation for a set of asks or a set of bids Ω, we define $\mathsf{Vol}(\Omega)$ to be the sum of the quantities of the orders in Ω.

A set of transactions M is a **matching** over a list of bids B and asks A with distinct ids if (i) For each transaction $m \in M$, the bid and ask of m come from B and A, respectively. (ii) For each transaction $m \in M$, the bid b of m is tradable with the ask a of m. (iii) For each order $w \in B \cup A$, $\mathsf{Qty}(w, M) \leq \mathsf{qty}(w)$.

Fairness and Competitiveness

In the problems we study, the matching produced needs to be fair, which essentially states that more competitive orders (based on price-time priority) have to be given preference in the matching. To formally define a fair matching, we first introduce the notion of competitiveness. For bids b_1 and b_2 we say b_1 is more **competitive** than b_2, denoted by $b_1 \succ b_2$, iff $\mathsf{price}(b_1) > \mathsf{price}(b_2)$ or $(\mathsf{price}(b_1) = \mathsf{price}(b_2)$ and $\mathsf{timestamp}(b_1) < \mathsf{timestamp}(b_2))$. Similarly, for asks a_1 and a_2, we say a_1 is more competitive than a_2, denoted by $a_1 \succ a_2$, iff $\mathsf{price}(a_1) < \mathsf{price}(a_2)$ or $(\mathsf{price}(a_1) = \mathsf{price}(a_2)$ and $\mathsf{timestamp}(a_1) < \mathsf{timestamp}(a_2))$. We say a matching M is fair on bids if a bid b participates in M, then all bids that are more competitive than b must be fully traded in M. Formally, M is fair on bids iff for all pairs of bids b_1 and b_2 such that $b_1 \succ b_2$, $\mathsf{Qty}(b_2, M) \geq 1 \implies \mathsf{Qty}(b_1, M) = \mathsf{qty}(b_1)$. Similarly, a matching M is fair on asks iff for all pairs of asks a_1 and a_2 such that $a_1 \succ a_2$, $\mathsf{Qty}(a_2, M) \geq 1 \implies \mathsf{Qty}(a_1, M) = \mathsf{qty}(a_1)$. A matching is **fair** if it is fair on bids as well as fair on asks.

[2] Later, the orders given as input to the algorithms, are all assumed to have distinct ids and distinct timestamps.

Uniform vs. Dynamic Price

There are two main types of call auctions: uniform-price and dynamic-price. In uniform price call auctions, the exchange is supposed to output a fair matching with maximum volume subject to the constraint that all transaction prices in the matching are identical. Customarily, the transaction price of a uniform price matching is referred to as the *equilibrium price*, and the process of discovering this price is called *price discovery*. In contrast, for dynamic price call auctions, the exchange is supposed to output a fair matching with maximum volume, where the transaction prices need not be the same. It must be noted that dropping the requirement of fairness from a dynamic price matching can only make the problem easier; the following result states that for any matching, there exists a fair matching of the same volume.

▶ **Theorem 3** (Proved in [9, 4]). *Given a set of bids B, a set of asks A, and a matching M over (B, A), one can find a fair matching M' over (B, A) such that*
(a) $\mathsf{Vol}(M) = \mathsf{Vol}(M')$, *and*
(b) *if M is uniform, then M' is uniform.*

Furthermore, [9, 4] provides an $O(n \log n)$-time algorithm to compute the matching M'.

▶ Remark 4. It is clear from the above theorem, that unlike uniformity, ensuring fairness does not change the volume of a matching. More precisely, given sets of bids B and asks A, consider the following matchings over (B, A). Let M_1 be a largest volume matching, M_2 be a fair matching with the largest volume, M_3 be a uniform matching with the largest volume, and finally let M_4 be a uniform and fair matching with the largest volume. Then,

$$\mathsf{Vol}(M_1) = \mathsf{Vol}(M_2) \geq \mathsf{Vol}(M_3) = \mathsf{Vol}(M_4).$$

Nevertheless, computing M_1 can be easier than computing M_2, and similarly computing M_3 can be easier than computing M_4. More precisely, computing M_1 reduces to computing M_2, and computing M_3 reduces to computing M_4.

Two Problems

The two matching problems that we study in this work can be stated as follows.

- **Problem 1: Dynamic-Price Matching.** Given a set of bids B and a set of asks A, find a fair matching over (B, A) of maximum volume.
- **Problem 2: Uniform-Price Matching.** Given a set of bids B and a set of asks A, find a fair and uniform matching over (B, A) of maximum volume.

Two models

We obtain our lower bound results in the following models of computation.

- **The Comparison Model:** We assume that the prices of the orders in the input are not given to us explicitly. Instead, two prices can be compared to each other via an oracle in unit time.
- **The Binary Query Model:** We will also derive a lower bound in a more general model, namely the binary query model, where the oracle can be made to evaluate an arbitrary function $f : \mathbb{N} \times \mathbb{N} \to \{0, 1\}$ on two prices in unit time. Note that the query function can be different for different queries.

▶ Remark 5. Our upper bound results do adhere to the more restricted comparison model.

We are now ready to state the previous and new results.

1.2 Earlier Works

We first briefly summarize the past results that were obtained for Problems 1 and 2. In the following, n represents the number of orders in the input, i.e., $n = |A| + |B|$.

- First, in [12] it was shown that Problem 2 can be solved in $O(n \log n)$ time.
- Next, in [13] it was shown that Problem 1 can be solved in $O(n^2)$ time under the assumption that all orders have unit quantity. The authors mention that for multi-unit orders, one can simply break an order into multiple single-unit orders and use their algorithm. But, this would then result in overall complexity $O(Q^2)$, where Q is the sum of the total demand and total supply, i.e., $Q = \mathsf{Vol}(A) + \mathsf{Vol}(B)$.
- This was improved in [10] where it was shown that Problem 1 can be solved in $O(n \log n)$ time for single-unit orders.
- Finally, inspired from previous works, in [9, 4] algorithms for both Problems 1 and 2 were presented that run in $O(n \log n)$ time each for arbitrary multi-unit orders. However, the authors did not analyze the time complexity of their algorithms, as they were more focused on formalizing these algorithms in a theorem prover. Their algorithms are general versions of the two algorithms we saw on page 2, where instead of sorting the lists of bids and asks by their prices, they are sorted by their competitiveness. After this modification, the first algorithm yields a maximum volume matching (note that Problem 1 requires the output matching to be also fair), whereas the second algorithm yields a uniform and fair matching with maximum volume (precisely solving Problem 2). For solving Problem 1, in addition to the above algorithm, they use an $O(n \log n)$ time algorithm that takes an arbitrary matching and outputs a fair matching of the same volume (this is always possible, see Theorem 3). It is straightforward to see that their algorithms indeed run in $O(n \log n)$ time.

1.3 Our Contribution

Now we describe the results obtained in this work.

Tight Lower Bound for Dynamic-Price Matching. For Problem 1, we show that any comparison-based algorithm for computing a maximum-volume matching (even ignoring fairness) requires $\Omega(n \log n)$ time in the worst case. We also show a stronger $\Omega(n\sqrt{\log n})$ bound in the broader binary query model. This confirms that the widely used $O(n \log n)$ (which also works for determining a maximum volume matching) sorting-based methods ([10, 9, 4]) are asymptotically optimal in the comparison model.

▶ **Theorem 6** (Result 1). *Any algorithm that takes as input a set of bids B and a set of asks A and computes a matching over (B, A) with the maximum volume has a worst-case running time of $\Omega(n \log n)$ in the comparison model and $\Omega(n\sqrt{\log n})$ in the binary query model, where $n = |B| + |A|$.*

The proof of Theorem 6 also applies to Theorem 1. In this proof, we obtain the desired lower bound by reducing the element distinctness problem over a small domain to the maximum volume matching problem.[3] In order to prove the correctness of our reduction, we crucially employ a stronger inequality than the demand-supply inequality of [10]. This

[3] An entropy-based argument similar to the one used for sorting in [11] also yields a lower bound for the maximum matching problem in the comparison model, but it fails to deliver any non-trivial lower bound in the binary query model.

stronger inequality, which we prove, can be of independent interest. Our result then follows from applying known lower bounds on the time complexity of the element distinctness problem [2, 7]. We include a proof of the lower bound in the comparison model making the argument in [7] more precise as it seems to be missing some important details.

Tight Linear-Time Algorithm for Uniform-Price Matching. For Problem 2, contrary to the dynamic case, we design a new algorithm that achieves a uniform-price matching in $O(n)$ time. This is an improvement over the previous algorithms [12, 13, 9, 4] that take $O(n \log n)$ time to match n requests.

▶ **Theorem 7** (Result 2). *There exists an algorithm that takes as input a set of bids B and a set of asks A, and outputs a uniform and fair matching with maximum volume in $O(n)$ time, where $n = |B| + |A|$.*

Theorem 7 immediately implies Theorem 2, as ensuring fairness does not reduce the size of a uniform matching, as observed earlier. Note that obtaining a linear time algorithm in the volume of the input orders is simpler than Result 2. Our improvement for uniform-price matching, roughly speaking, comes from eliminating the sorting step in the algorithm of [9, 4] as described earlier. Instead, we compute the "medians" of the bids and asks in linear time which bisects the sets of bids and asks into two "equal halves" each. Depending on whether the medians are tradable or not, we can either get rid of half of the bids and asks, or match half of the input in linear time, thereby reducing the problem size by half in either case. This results in a linear time algorithm. A number of interesting challenges are encountered while implementing the above idea. In particular, when we discard "half" of the orders, they might still be matchable with the other "half" that we recurse on; prima facie, it is not obvious why such orders can be safely discarded.

1.4 Practical Significance

Call auctions are ubiquitous in financial markets. Major stock exchanges – including the New York Stock Exchange (NYSE), NASDAQ, and various global exchanges – often rely on uniform-price call auctions at market open and market close to establish a single clearing price. The main advantage is its simplicity and impartiality: every matched trader pays/receives the same price. Similar rules govern IPO allocations and some commodity trading platforms. Our work thus not only answers fundamental complexity questions – shedding light on the intrinsic difficulty of maximum-volume matching – but also yields a more efficient uniform-price procedure. In high-frequency or large-volume trading environments, any reduction in algorithmic overhead can be consequential.

Moreover, these results highlight an intellectually appealing paradox: while dynamic-price matching intrinsically requires sorting (or an equivalent $O(n \log n)$ process) under the comparison model, uniform-price matching admits a more efficient linear-time solution. This dichotomy had never been formally established, despite the centrality of call auctions in marketplace design.

This work offers a fundamental yet previously unexamined algorithmic advance for widely used exchange procedures, and we believe these insights will be of significant interest to researchers and practitioners in auction theory, matching markets, operations research, and financial engineering alike.

1.5 Further Related Work

In [5, 6], continuous auctions were studied, that form a class of double auctions that complement the call auctions. The authors presented an algorithm that implements the continuous auction and runs in $O(n \log n)$ time for processing n *instructions* (requests/orders). They also showed that their algorithm is time-optimal by reducing the task of sorting to continuous auctions.

Organization of the Rest of the Paper

Theorems 1 and 6 are proved in Section 2. Next, in Section 3, Theorem 7 is proved. Finally, we conclude the paper in Section 4 with some concluding remarks.

2 Lower Bound for Maximum-Volume Matching

In this section we prove Theorem 6 (note that the proof also works for Theorem 1), which we restate.

▶ **Theorem 6** (Result 1). *Any algorithm that takes as input a set of bids B and a set of asks A and computes a matching over (B, A) with the maximum volume has a worst-case running time of $\Omega(n \log n)$ in the comparison model and $\Omega(n\sqrt{\log n})$ in the binary query model, where $n = |B| + |A|$.*

In our proof, we will show a reduction from the following problem to the maximum-volume matching problem.

Element Distinctness Problem (on small domain). Given an input $(x_1, \cdots, x_n)$, where each $x_i \in [n]$, check whether there are distinct indices i and j such $x_i = x_j$.[4]

Our result then immediately follows from the fact that the element distinctness problem on a small domain requires time $\Omega(n \log n)$ in the comparison model [7] (for which we include a proof in the next subsection) and time $\Omega(n\sqrt{\log n})$ in the binary query model [2].

Furthermore, our reduction will rely on a stronger version of the demand-supply inequality of [10], which states that the volume of any matching is upper bounded by the sum of the total demand and the total supply at any price.

▶ **Theorem 8** (demand-supply inequality). *If M is a matching over a set of bids B and a set of asks A, then for all numbers p, we have*

$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B_{\geq p}) + \mathsf{Vol}(A_{\leq p})$$

where $B_{\geq p} \subseteq B$ consists of bids whose limit prices are at least p, $A_{\leq p} \subseteq A$ consists of asks whose limit prices are at most p.

Our strengthening, which is proved in Section 2.2, comes from observing that the above inequality holds even if we replace $\mathsf{Vol}(B_{\geq p})$ by $\mathsf{Vol}(B_{>p})$ or $\mathsf{Vol}(A_{\leq p})$ by $\mathsf{Vol}(A_{<p})$, and can be stated as follows.

[4] For $n \geq 1$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$.

▶ **Theorem 9** (strong demand-supply inequality)**.** *If M is a matching over a set of bids B and a set of asks A, then for all numbers p, we have*

$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(A_{<p}) + \min(\mathsf{Vol}(B_{=p}), \mathsf{Vol}(A_{=p})),$$

where $B_{>p} \subseteq B$ consists of bids whose limit prices are greater than p, $A_{<p} \subseteq A$ consists of asks whose limit prices are less than p, and $B_{=p} \subseteq B$ (or $A_{=p} \subseteq A$) consists of bids (or asks) whose limit prices are exactly p.

We are now ready to do our proof.

Proof of Theorem 6. Let MM be an algorithm for the maximum matching problem. We reduce the element distinctness problem on a small domain to MM in linear time.

The reduction. Given an instance $X = (x_1, \cdots, x_n)$ for the element distinctness problem, we construct two sets of orders $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$ and $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ such that the quantity of each order in $\Omega \cup \Lambda$ is set to 1. We set the prices as follows. $\mathsf{price}(\omega_i) = i$ and $\mathsf{price}(\lambda_i) = x_i$. Observe that the prices of orders in Ω are all distinct. Next, we run the maximum matching algorithm MM twice on these inputs by first treating Ω as the set of bids and then treating Ω as the set of asks to obtain two matchings M_1 and M_2, respectively; $M_1 = \mathsf{MM}(\Omega, \Lambda)$ and $M_2 = \mathsf{MM}(\Lambda, \Omega)$.

We now claim that if elements of X are all distinct then $\mathsf{Vol}(M_1) = \mathsf{Vol}(M_2) = n$, otherwise $\mathsf{Vol}(M_1) < n$ or $\mathsf{Vol}(M_2) < n$. Note that if we show this claim the reduction is complete, as we can then solve the element distinctness problem mentioned above in time $O(n)$ plus twice the time taken by MM on $2n$ orders.

It is easy to see that if the x_i's are all distinct, then $\mathsf{Vol}(M_1) = \mathsf{Vol}(M_2) = n$; the bid with price i is matched with the ask with price i.

Now, we show that if the x_i's are not distinct, then $\mathsf{Vol}(M_1) \leq n-1$ or $\mathsf{Vol}(M_2) \leq n-1$. From the pigeon-hole principle, if two elements are repeating in X, then one of the elements from $[n]$ must be missing from X. Let the smallest missing element in X be m and the smallest repeating element in X be r.

For a set of orders Ω, we define $\Omega_{=t} := \{\omega \in \Omega \mid \mathsf{price}(\omega) = t\}$, $\Omega_{<t} := \{\omega \in \Omega \mid \mathsf{price}(\omega) < t\}$, and $\Omega_{>t} := \{\omega \in \Omega \mid \mathsf{price}(\omega) > t\}$. Now, observe that $\mathsf{Vol}(\Lambda_{=m}) = 0$ (since m is missing from X which is the set of prices of Λ), which implies $\min(\mathsf{Vol}(\Omega_{=m}), \mathsf{Vol}(\Lambda_{=m})) = 0$.

We now consider two cases: $m < r$ or $m > r$.

Case: $m < r$. In this case we will show that $\mathsf{Vol}(M_1) < n$. Observe that the number of elements in X that are at most m is precisely $m-1$, since no element smaller than m is repeating and m itself is missing. Here, the input x_i's are set to be the ask prices, i.e., Λ is the set of asks, and Ω is the set of bids. Therefore, $\mathsf{Vol}(\Lambda_{<m}) = |\{\lambda \in \Lambda \mid \mathsf{price}(\lambda) < m\}| = m-1$ and $\mathsf{Vol}(\Omega_{>m}) = n - m$. Hence, from Theorem 9,

$$\begin{aligned}\mathsf{Vol}(M_1) &\leq \mathsf{Vol}(\Omega_{>m}) + \mathsf{Vol}(\Lambda_{<m}) + \min(\mathsf{Vol}(\Omega_{=m}), \mathsf{Vol}(\Lambda_{=m})) \\ &= (n-m) + (m-1) + 0 = n-1.\end{aligned}$$

Case: $m > r$. In this case we will show that $|M_2| < n$. Note that no element in X is missing below m but some elements are repeating. Thus, the number of elements that are greater than m are at most $n-m$. Hence, similar to the above case, applying Theorem 9, we have $\mathsf{Vol}(M_2) \leq \mathsf{Vol}(\Lambda_{>m}) + \mathsf{Vol}(\Omega_{<m}) + \min(\mathsf{Vol}(\Lambda_{=m}), \mathsf{Vol}(\Omega_{=m})) \leq (n-m) + (m-1) + 0 = n-1$. ◀

2.1 Lower Bound for Element Distinctness on Small Domain

We now show a lower bound on the number of comparisons needed to solve the element distinctness problem on a small domain. This is a known result, but the proof that we could find has gaps, so we decided to include a complete proof.

▶ **Theorem 10.** *Any algorithm that takes as input $(x_1, \cdots, x_n)$, where each $x_i \in [n]$, and can decide whether there exist i and j such that $i \neq j$ and $x_i = x_j$, takes $\Omega(n \log n)$ time in the comparison model.*

Proof. Any algorithm for element distinctness can be seen as a decision tree where each internal node is labeled by a comparison $X_i : X_j$, for some $i \neq j$, and each leaf is labeled either Yes or No. On a given input $(x_1, \cdots, x_n)$ the decision tree is applied as follows. We start at the root node, traverse a path to a leaf, and declare the label of the leaf as the output. On encountering a node labeled $X_i : X_j$, if $x_i \leq x_j$, we take its left child as the next node in the path, otherwise, we take its right child. Inputs corresponding to permutations on $[n]$ must receive the answer Yes, and inputs where $x_i = x_j$ for some $i \neq j$ must receive the answer No.

We fix a decision tree T for the element distinctness problem and prove that its height is at least $\Omega(n \log n)$. We make the following claim.

▷ Claim 11. Each permutation on $[n]$ ends up in a distinct leaf of T.

This immediately implies that the number of leaves is at least $n!$ and hence the height of the tree is at least $\Omega(\log n!) = \Omega(n \log n)$. We now turn to proving the above claim.

For the sake of contradiction we assume two distinct permutations $\sigma = (\sigma_1, \cdots, \sigma_n)$ and $\tau = (\tau_1, \cdots, \tau_n)$ end up at the same leaf L of T. To obtain a contradiction we will find an input $(S_1, \cdots, S_n)$ where $S_i = S_j$ for some $i \neq j$ which also ends up in the leaf L.

We define a poset $P(L)$ on the set of symbols $X = \{X_1, \cdots, X_n\}$. In our poset when X_i and X_j are related, we will write $X_i \leq X_j$ and call it a constraint. For each node labeled $X_i : X_j$ in the path from the root to the leaf L in the tree T, we have a constraint of the form $X_i \leq X_j$ or $X_j \leq X_i$ (in fact, $X_j < X_i$ suffices, but we use the weaker form) which must be satisfied for a computation to take this path. Let C be the set of all such constraints. We define $C(L) = \bigcap\{\hat{C} \mid C \subseteq \hat{C} \text{ such that } (X, \hat{C}) \text{ is a poset}\}$; notice that this is a non-empty intersection, as we have at least one poset, namely the total order $X_{\sigma(1)} \leq X_{\sigma(2)} \leq \cdots \leq X_{\sigma(n)}$ which satisfies all constraints in C as σ ends up in the leaf L. We define $P(L) := (X, C(L))$.

We say an input $(x_1, \cdots, x_n)$ *respects* a poset P if $x_i \leq x_j$ whenever $X_i \leq X_j$ is a constraint in P. The following proposition is easy to prove.

▶ **Proposition 12.** *An input reaches the leaf L iff it respects the poset $P(L)$.*

Next, observe that since σ and τ are two distinct permutations that respect $P(L)$, the length of a largest chain in $P(L)$ is at most $n - 1$.[5] Mirsky's theorem [8] then implies that there is an antichain decomposition of $P(L)$ with at most $n - 1$ antichains. Since X has n elements, there must be an antichain in $P(L)$ that has at least two elements. Let A be a maximal antichain in $P(L)$ that has at least two elements. We can write X as a disjoint union of three sets $X = Q \uplus A \uplus R$, where Q consists of elements below A and R consists of elements

[5] At this point, the proof in [7] instantly concludes that there must be two elements in the poset that are not related and fixes an input which respects the poset and where these two coordinates are equal, to complete the proof, without specifying why such an input exists. We fill such gaps in the proof.

above A in the poset $P(L)$: $Q = \{X_i \in X \setminus A \mid$ there exists $X_j \in A$ such that $X_i \leq X_j\}$ and $R = \{X_i \in X \setminus A \mid$ there exists $X_j \in A$ such that $X_j \leq X_i\}$ (observe that an element cannot be both above and below A). The constraints in $C(L)$ restricted to Q and restricted to R give rise to the posets: P_Q and P_R. Now we define the poset (X, D) such that $C(L) \subseteq D$ as follows. Take a linear extension of P_Q: $X_{t_1} \leq \cdots \leq X_{t_q}$ and add all the implied constraints to D. Do not add any constraints between the elements of the antichain $A = \{X_{t_{q+1}}, X_{t_{q+2}}, \cdots, X_{t_{q+a}}\}$. Take a linear extension of P_R: $X_{t_{q+a+1}} \leq \cdots \leq X_{t_n}$ and add all the implied constraints to D. Furthermore, add the following constraints to D: For each $X_i \in X$, add $X_i \leq X_i$. For each $X_i \in Q$ and $X_j \in A \cup R$, add $X_i \leq X_j$. For each $X_i \in A$ and each $X_j \in R$, add $X_i \leq X_j$. Clearly, $C(L) \subseteq D$. Now fix the input S such that $S_{t_1} = 1, S_{t_2} = 2, \cdots, S_{t_q} = q$, $S_{t_{q+1}} = S_{t_{q+2}} = \cdots = S_{t_{q+a}} = q + 1$, $S_{t_{q+a+1}} = q + 2, S_{t_{q+a+2}} = q + 3, \cdots, S_{t_n} = n - (a - 1)$. Now, S clearly respects the poset (X, D). Thus, it also respects the poset $P(L)$ (as $C(L) \subseteq D$) and reaches the leaf L. Since $a = |A| \geq 2$, S is a No instance which reaches L, a contradiction. ◀

2.2 Stronger Demand-Supply Inequality

In this subsection, we prove Theorem 9.

▶ **Theorem 9** (strong demand-supply inequality)**.** *If M is a matching over a set of bids B and a set of asks A, then for all numbers p, we have*

$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(A_{<p}) + \min(\mathsf{Vol}(B_{=p}), \mathsf{Vol}(A_{=p})),$$

where $B_{>p} \subseteq B$ consists of bids whose limit prices are greater than p, $A_{<p} \subseteq A$ consists of asks whose limit prices are less than p, and $B_{=p} \subseteq B$ (or $A_{=p} \subseteq A$) consists of bids (or asks) whose limit prices are exactly p.

Proof of Theorem 9. First observe that the volume of any matching is upper bounded by the volume of all bids as well as the volume of all asks, i.e., if M is a matching over (B, A), then

$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B) \quad \text{and} \quad \mathsf{Vol}(M) \leq \mathsf{Vol}(A).$$

To prove Theorem 9, it suffices to prove the following two inequalities.

$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(A_{<p}) + \mathsf{Vol}(B_{=p});$$
$$\mathsf{Vol}(M) \leq \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(A_{<p}) + \mathsf{Vol}(A_{=p}).$$

To prove the first inequality we partition the matching M into two sets: $M_1 = \{(b, a, q, p') \in M \mid \mathsf{price}(b) \geq p\}$ consisting of all transactions in M whose participating bid price is at least p and $M_2 = \{(b, a, q, p') \in M \mid \mathsf{price}(b) < p\}$ consisting of all transactions in M whose participating bid price is strictly less than p. Thus, $\mathsf{Vol}(M) = \mathsf{Vol}(M_1) + \mathsf{Vol}(M_2)$.

It is easy to see that M_1 is a matching over sets of bids $B_{\geq p}$ and asks A, and hence from the above observation, $\mathsf{Vol}(M_1) \leq \mathsf{Vol}(B_{\geq p}) = \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(B_{=p})$.

Next, we prove that M_2 is a matching over sets of bids B and asks $A_{<p}$. Consider a transaction $m = (b, a, q, p')$ from M_2, which is between a bid b and an ask a. Since $m \in M$, $\mathsf{price}(b) \geq \mathsf{price}(a)$, and from the definition of M_2, we have $\mathsf{price}(b) < p$. This implies $\mathsf{price}(a) < p$, i.e., asks of M_2 come from $A_{<p}$. Hence, M_2 is a matching over $(B, A_{<p})$, and applying the above observation again, we have $\mathsf{Vol}(M_2) \leq \mathsf{Vol}(A_{<p})$.

Combining, we have $\mathsf{Vol}(M) = \mathsf{Vol}(M_1) + \mathsf{Vol}(M_2) \leq \mathsf{Vol}(B_{>p}) + \mathsf{Vol}(B_{=p}) + \mathsf{Vol}(A_{<p})$, which completes the proof of first inequality.

Similarly, we can prove the second inequality by first partitioning the matching M into $M_1 = \{(b, a, q, p') \in M \mid \mathsf{price}(b) > p\}$ and $M_2 = \{(b, a, q, p') \in M \mid \mathsf{price}(b) \leq p\}$, noticing that M_1 is a matching over $(B_{>p}, A)$ and M_2 is matching over $(B, A_{\leq p})$, and finally, using the observation again to obtain the desired inequality. ◀

3 A Linear-Time Algorithm for Uniform-Price Matching

In this section, we prove Theorem 7, which we first restate.

▶ **Theorem 7** (Result 2). *There exists an algorithm that takes as input a set of bids B and a set of asks A, and outputs a uniform and fair matching with maximum volume in $O(n)$ time, where $n = |B| + |A|$.*

We begin by first describing the previous $O(n \log n)$ algorithm as described in [9, 4], which we will critically use in establishing the correctness of our linear time algorithm.

We may assume that the input lists of asks A and bids B are such that $\mathsf{Vol}(A) = \mathsf{Vol}(B)$. This can be achieved by adding a dummy order. Say $\mathsf{Vol}(A) < \mathsf{Vol}(B)$, then add a dummy ask with $\mathsf{price} = \infty$ and $\mathsf{qty} = \mathsf{Vol}(B) - \mathsf{Vol}(A)$. If $\mathsf{Vol}(A) > \mathsf{Vol}(B)$, then add a dummy bid with $\mathsf{price} = -1$ and $\mathsf{qty} = \mathsf{Vol}(A) - \mathsf{Vol}(B)$. Strictly speaking, according to our definition, we cannot set the price to be ∞ or -1, but our definitions can be adjusted to allow for this.

3.1 Previous Algorithm

We describe the algorithm in [9, 4], which we denote by UM. The transaction prices that are set by UM are not guaranteed to be uniform initially. After the matching is produced, a linear time algorithm is employed to put a uniform transaction price for all the transactions. For example, the maximum of the limit prices of the asks that participate in the output matching can be set as the transaction price of all the transactions in the output matching. Consequently, in the algorithms presented below, we will not worry about putting a transaction price in the transactions.

Given the list of bids B and asks A, UM first sorts the lists based on their respective competitiveness with the most competitive order on the top. It then invokes Match on the sorted lists B, A, and an empty matching M. Throughout the algorithm, M can only grow, and at the end of the algorithm, M will contain a desired uniform-price matching.

Algorithm 1 Uniform Matching UM.

Function `UM`(*Bids B, Asks A*)**:**
  Sort the lists B and A based on their respective competitiveness;
  return Match$(B, A, \emptyset)$;
end

Match on (B, A, M) picks the most competitive bid b and the most competitive ask a from B and A respectively, and checks if they are tradable. If they are not tradable, it returns M, and the algorithm terminates.

Otherwise, if they are tradable, it adds a transaction between b and a with transaction quantity $q = \min\{\mathsf{qty}(a), \mathsf{qty}(b)\}$ to M, reduces the quantity of a and b by q in the lists B and A. Note that at least one of b or a must be fully exhausted and removed completely from B or A. It then recursively calls Match on (B, A, M).

Clearly `UM` takes $O(n \log n)$ time, whereas Match takes $O(n)$ time, where $n = |A| + |B|$.

The algorithm can be simply described as first sorting the lists B and A based on competitiveness, and then doing a top-down greedy matching as long as the current bid and ask are tradable.

Algorithm 2 The Match subroutine.

Function `Match`(*Bids* B*, Asks* A*, Matching* M) `// Initially,` $M = \emptyset$`.`:
  if $|B| = 0$ ***or*** $|A| = 0$ **then**
    return M;
  $b \leftarrow \mathsf{pop}(B)$;
  $a \leftarrow \mathsf{pop}(A)$;
  if $\mathsf{price}(b) < \mathsf{price}(a)$ **then**
    $\mathsf{push}(B, b)$;
    return `Match`(B, A, M);
  `// Otherwise,` b `and` a `are tradable`
  $q \leftarrow \min\{\mathsf{qty}(a), \mathsf{qty}(b)\}$;
  $\mathsf{push}(M, \{(\mathsf{id}(b), \mathsf{id}(a), q, \mathsf{price}(a))\})$;
  if $\mathsf{qty}(b) - q > 0$ **then**
    $\mathsf{push}(B, (\mathsf{id}(b), \mathsf{timestamp}(b), \mathsf{qty}(b) - q, \mathsf{price}(b)))$;
  if $\mathsf{qty}(a) - q > 0$ **then**
    $\mathsf{push}(A, (\mathsf{id}(a), \mathsf{timestamp}(a), \mathsf{qty}(a) - q, \mathsf{price}(a)))$;
  return `Match`(B, A, M);
end

The following result states that the above result is correct.

▶ **Theorem 13** (Proved in [9, 4])**.** `UM`(B, A) *outputs a uniform price matching.*

We give a brief intuition of the proof of the above result. Let $M = \texttt{UM}(B, A)$. To see that M is a fair matching is trivial, as it starts matching the bids and asks in the decreasing order of competitiveness. Similarly, it is easy to convince oneself that M has a uniform price; any price between the limit prices of the last paired bid and ask is acceptable to all orders that participate in the matching; this again follows from the fact that matching is done top down in the order of decreasing competitiveness. Finally, to see that M has maximum volume needs some work: fix an optimal matching OPT. We can gradually transform OPT into M without altering its volume. The illuminating case to consider is when the most competitive orders b and a are fully traded in M. Note that the transaction quantity between b and a in M is $q = \min\{\mathsf{qty}(b), \mathsf{qty}(a)\}$. Since OPT is fair, b and a must also be fully traded in OPT. In particular $\mathsf{Qty}(b, \mathsf{OPT}) \geq q$ and $\mathsf{Qty}(a, \mathsf{OPT}) \geq q$. Now, if the transaction quantity between b and a in OPT is strictly less than q, then OPT can be modified by making the transaction quantity between b and a equal to q. Let us say b is matched with a' and a is matched with b' in OPT, then we can reduce the quantity of these transactions by a unit quantity, increase the transaction quantity between b and a by unit quantity, and increase the transaction quantity between b' and a' by a unit quantity (this is possible since every matched bid is tradable with every matched ask as all the transaction prices are identical). Note that in the end, $\mathsf{Vol}(\mathsf{OPT})$ remains the same. We can repeat this surgery over and over again till the matched quantity between b and a in OPT becomes equal to q. We remove this transaction from both OPT and M and apply the same argument repeatedly till $M = \mathsf{OPT}$.

3.2 Useful Lemmas and Subroutines

Before we proceed to describe our improved algorithm, we establish certain lemmas and subroutines that will be useful in describing our algorithm and proving its correctness.

We first need some definitions.

▶ **Definition 14** (Split, Range). *For a set of bids or a set of asks Ω and an element $\omega \in \Omega$, $\mathsf{Split}(\Omega, \omega)$ returns a partition of $\Omega = (\Omega_{\succeq}, \Omega_{\prec})$, where $\Omega_{\succeq} = \{x \in \Omega \mid x \succeq \omega\}$ and $\Omega_{\prec} = \{x \in \Omega \mid x \prec \omega\}$. Thus, Split splits Ω into two parts, one containing orders that are at least as competitive as ω, and the other containing orders that are less competitive than ω. Clearly, Split can be implemented in linear time.*

We now define the range of an order in a set of orders Ω which are all bids or all asks. Let ω be an order in Ω. Let $\mathsf{Split}(\Omega, \omega) = (\Omega_{\succeq}, \Omega_{\prec})$. Let $\Omega_{\succ} = \Omega_{\succeq} \setminus \{\omega\}$. $\mathsf{Range}_{\Omega}(\omega) = \{x \in \mathbb{N} \mid \mathsf{Vol}(\Omega_{\succ}) < x \leq \mathsf{Vol}(\Omega_{\succeq})\}$. If all orders in Ω have unit quantities, then the Range_{Ω} of the i^{th} most competitive order is the singleton $\{i\}$. In general, range of the most competitive order ω_1 is the set $\{1, \cdots, \mathsf{qty}(\omega_1)\}$. The range of the next most competitive order ω_2 is $\{\mathsf{qty}(\omega_1) + 1, \cdots, \mathsf{qty}(\omega_1) + \mathsf{qty}(\omega_2)\}$, and so on. Thus, the ranges of orders in Ω partition $[\mathsf{Vol}(\Omega)]$.

Observe the following fact about the previous algorithm `UM`.

▶ **Lemma 15.** *If a bid $b \in B$ and an ask $a \in A$ are matched in the matching output by $\mathtt{UM}(B, A)$, then $\mathsf{Range}_B(b) \cap \mathsf{Range}_A(a) \neq \emptyset$.*

This is trivial to see when all orders in $B \cup A$ have unit quantities since `UM` matches the i^{th} most competitive bid with the i^{th} most competitive ask for all $i \leq \mathsf{Vol}(M)$, where $M = \mathtt{UM}(B, A)$, and the Range of the i^{th} most competitive bid (and ask) is the singleton $\{i\}$. To see why the general statement is true observe that whenever $\mathtt{Match}(\hat{B}, \hat{A}, \hat{M})$ is called it potentially matches the most competitive bid $b \in \hat{B}$ with the most competitive ask $a \in \hat{A}$, and $\mathsf{Vol}(\hat{M}) + 1 \in \mathsf{Range}_B(b) \cap \mathsf{Range}_A(a)$.

Our main workhorse is the Select subroutine which works in linear time by employing the classical algorithm of [1]. Given a list of orders (all bids or all asks) Ω and a number $t \leq |\Omega|$, $\mathsf{Select}(\Omega, t)$ outputs the t^{th} most competitive order in Ω.

We also use a weighted version of Select called $\mathsf{SelectQ}$ which takes as input Ω and a quantity $q \leq \mathsf{Vol}(\Omega)$ and outputs the unique element $\omega \in \Omega$ such that $q \in \mathsf{Range}_{\Omega}(\omega)$. $\mathsf{SelectQ}$ can be implemented in linear time using the Select algorithm as subroutine as described below.

Algorithm 3 Weighted Selection where weights are quantities.

Function `SelectQ`(*Orders* Ω, q)**:**
  $\omega \leftarrow \mathsf{Select}\left(\Omega, \left\lceil \frac{|\Omega|}{2} \right\rceil\right)$;
  $(\Omega_{\succeq}, \Omega_{\prec}) \leftarrow \mathsf{Split}(\Omega, \omega)$;
  if $\mathsf{Vol}(\Omega_{\succeq}) - \mathsf{qty}(\omega) < q \leq \mathsf{Vol}(\Omega_{\succeq})$ **then**
    return ω;
  if $q \leq \mathsf{Vol}(\Omega_{\succeq}) - \mathsf{qty}(\omega)$ **then**
    return $\mathtt{SelectQ}(\Omega_{\succeq}, q)$;
  if $q > \mathsf{Vol}(\Omega_{\succeq})$ **then**
    return $\mathtt{SelectQ}(\Omega_{\prec}, q - \mathsf{Vol}(\Omega_{\succeq}))$;
end

We get the following recurrence relationship on the running time of $\mathsf{SelectQ}$: $T(n) \leq T(\lceil n/2 \rceil + 1) + O(n)$, which yields $T(n) = O(n)$, where $n = |\Omega|$.

The next subroutine we consider is $\mathsf{Split}(\Omega, \omega)$, which partitions Ω into two parts consisting of orders in Ω which are at least as competitive as ω, and orders which are strictly less competitive than ω. We now want to define a subroutine SplitQ which takes as input a list of orders Ω and a quantity $q \leq \mathsf{Vol}(\Omega)$. We want to "partition" Ω into two parts so that the volume of the more competitive part is precisely q. For this, we first find the element $\omega \in \Omega$ such that $q \in \mathsf{Range}_{\Omega}(\omega)$. We then $\mathsf{Split}(\Omega, \omega)$ to obtain $(\Omega_{\succeq}, \Omega_{\prec})$. If $\mathsf{Vol}(\Omega_{\succeq}) = q$, then we are immediately done. Else $\mathsf{Vol}(\Omega_{\succeq}) > q$ and $\mathsf{Vol}(\Omega_{\succeq} \setminus \{\omega\}) < q$. Thus we must break a part of ω and put it in $\Omega_{\prec}$ so that $\mathsf{Vol}(\Omega_{\succeq})$ is precisely q. This can be achieved in linear time by the following subroutine $\mathsf{SplitQ}(\Omega, q)$. Apart from outputting the partition as described above, it also returns the order ω. SplitQ clearly runs in linear time.

Algorithm 4 Splitting an order list by a particular quantity.

Function `SplitQ(`*Orders* Ω*, quantity* q`)` `// Promise:` $q \leq \mathsf{Vol}(\Omega)$**:**
  $\omega \leftarrow \mathsf{SelectQ}(\Omega, q)$;
  $(\Omega_{\succeq}, \Omega_{\prec}) \leftarrow \mathsf{Split}(\Omega, \omega)$;
  $\Omega_{\preceq} \leftarrow \Omega_{\prec}$;
  $q_{\text{extra}} \leftarrow \mathsf{Vol}(\Omega_{\succeq}) - q$;
  if $q_{extra} > 0$ **then**
    $\Omega_{\succeq} \leftarrow \left(\Omega_{\succeq} \setminus \{\omega\}\right) \cup \{(\mathsf{id}(\omega), \mathsf{timestamp}(\omega), \mathsf{qty}(\omega) - q_{\text{extra}}, \mathsf{price}(\omega))\}$;
    $\Omega_{\preceq} \leftarrow \Omega_{\prec} \cup \{(\mathsf{id}(\omega), \mathsf{timestamp}(\omega), q_{\text{extra}}, \mathsf{price}(\omega))\}$;
  return $(\omega,\ \Omega_{\succeq},\ \Omega_{\preceq})$;
end

We are now ready to state another lemma regarding the previous algorithm `UM`.

▶ **Lemma 16.** *Let* $\hat{b} \in B$ *be the* t^{th} *most competitive bid that gets completely traded in* $M = \mathtt{UM}(B, A)$*. Let* $\mathsf{Split}(B, \hat{b}) = (B_{\succeq}, B_{\prec})$ *and* $q = \mathsf{Vol}(B_{\succeq})$*. Let* $\mathsf{SplitQ}(A, q) = (\hat{a}, A_{\succeq}, A_{\preceq})$*. Then,* M *can be partitioned into* (M_1, M_2) *such that* $M_1 = \mathtt{UM}(B_{\succeq}, A_{\succeq})$ *and* $M_2 = \mathtt{UM}(B_{\prec}, A_{\preceq})$*.*

The above statement is easy to check. Since $\hat{b}$ gets completely traded in M, all orders in $B_{\succeq}$ must get traded with the most competitive asks in A whose total quantity is q, which is precisely the set $A_{\succeq}$. Let M_1 consist of the transactions produced while matching orders in $B_{\succeq}$. Then M_2 will be obtained by running the `UM` on $(B_{\prec}, A_{\preceq})$.

Note that a symmetric statement holds where we start with the assumption that the t^{th} most competitive ask gets traded in M.

We are now ready to describe our improved algorithm.

3.3 Improved Algorithm

Our improved algorithm $\mathsf{Uniform}$ takes as input a list of bids B and a list of asks A. It immediately invokes $\mathsf{Uniform}_{\mathsf{bid}}$ on $(B, A, M = \emptyset)$. M will grow into the final matching output by the algorithm.

Algorithm 5 Efficient Uniform Algorithm.

Function Uniform(*Bids* B, *Asks* A)**:**
  $\mathsf{Uniform}_{\mathsf{bid}}(B, A, \emptyset)$;
end

There are two main subroutines of our algorithm $\mathsf{Uniform}_{\mathsf{bid}}$ and $\mathsf{Uniform}_{\mathsf{ask}}$ which are symmetric in nature and they alternatively call each other. This is done to ensure that in two successive return calls, the problem size, i.e., $|B| + |A|$, reduces by a factor of two, as $\mathsf{Uniform}_{\mathsf{bid}}$ halves the number of the bids, whereas $\mathsf{Uniform}_{\mathsf{ask}}$ halves the number of the asks. So just understanding $\mathsf{Uniform}_{\mathsf{bid}}$ will be sufficient for understanding our algorithm.

On receiving (B, A, M), $\mathsf{Uniform}_{\mathsf{bid}}$ first finds the median bid b by invoking $\mathsf{Select}(B, \lceil \frac{B}{2} \rceil)$. It then splits B into two halves $(B_{\succeq}, B_{\prec})$ based on the median bid b by invoking $\mathsf{Split}(B, b)$. Let $q = \mathsf{Vol}(B_{\succeq})$. It then finds the element $a \in A$ such that $q \in \mathsf{Range}_A(a)$ by calling `SplitQ(`A`,`q`)` and splits A into its most competitive and least competitive asks $(A_{\succeq}, A_{\preceq})$ such that $\mathsf{Vol}(A_{\succeq}) = q$ by applying $\mathsf{SplitQ}(A, q)$.

After that, the algorithm checks if b and a are tradable. Two cases arise. If b and a are tradable, then every order in $B_{\succeq}$ and $A_{\succeq}$ are tradable and they will exhaustively be matched to each other to produce a matching of volume q in linear time using the `Match` subroutine. $\mathsf{Uniform}_{\mathsf{bid}}$ adds this Vol q matching to M and recursively calls $\mathsf{Uniform}_{\mathsf{ask}}$ on $(B_{\prec}, A_{\preceq}, M)$. Intuitively, the previous algorithm would also match all orders in $B_{\succeq}$ and $A_{\succeq}$ exhaustively to each other to produce a matching of volume q and proceed to matching orders in $B_{\prec}$ and $A_{\preceq}$.

In the case b and a are not tradable, then $\mathsf{Uniform}_{\mathsf{bid}}$ discards $(B_{\prec}, A_{\preceq})$ and calls $\mathsf{Uniform}_{\mathsf{ask}}$ on $(B_{\succeq}, A_{\succeq}, M)$. Intuitively, the previous algorithm will halt (i.e., produce its last transaction) before it even comes down to examining orders in $B_{\succeq}$ and $A_{\succeq}$.

Algorithm 6 Uniform Matching by bisecting bids.

Function $\mathsf{Uniform}_{\mathsf{bid}}$(*Bids* B, *Asks* A, *Matching* M)**:**
  if $|B| = 0$ *or* $|A| = 0$ *or* $\Big(B = \{b\}$ *and* $A = \{a\}$ *and* $\mathsf{price}(a) > \mathsf{price}(b)\Big)$ **then**
    return M;
  $b \leftarrow \mathsf{Select}\Big(B, \Big\lceil \frac{|B|}{2} \Big\rceil\Big)$;
  $(B_{\succeq}, B_{\prec}) \leftarrow \mathsf{Split}(B, b)$;
  $(a, A_{\succeq}, A_{\preceq}) \leftarrow \mathsf{SplitQ}\Big(A, \mathsf{Vol}(B_{\succeq})\Big)$;
  if $\mathsf{price}(a) \leq \mathsf{price}(b)$ **then**
    $M \leftarrow M \cup \mathsf{Match}(B_{\succeq}, A_{\succeq})$;
    return $\mathsf{Uniform}_{\mathsf{ask}}(B_{\prec}, A_{\preceq}, M)$;
  `// Otherwise,` b `and` a `are not tradable`
  return $\mathsf{Uniform}_{\mathsf{ask}}(B_{\succeq}, A_{\succeq}, M)$;
end

Algorithm 7 Uniform Matching by bisecting asks.

Function $\mathsf{Uniform}_{\mathsf{ask}}$(*Bids* B, *Asks* A, *Matching* M)**:**
  if $|B| = 0$ *or* $|A| = 0$ *or* $\Big(B = \{b\}$ *and* $A = \{a\}$ *and* $\mathsf{price}(a) > \mathsf{price}(b)\Big)$ **then**
    return M;
  $a \leftarrow \mathsf{Select}\Big(A, \Big\lceil \frac{|A|}{2} \Big\rceil\Big)$;
  $(A_{\succeq}, A_{\prec}) \leftarrow \mathsf{Split}(A, a)$;
  $(b, B_{\succeq}, B_{\preceq}) \leftarrow \mathsf{SplitQ}\Big(B, \mathsf{Vol}(A_{\succeq})\Big)$;
  if $\mathsf{price}(a) \leq \mathsf{price}(b)$ **then**
    $M \leftarrow M \cup \mathsf{Match}(B_{\succeq}, A_{\succeq})$;
    return $\mathsf{Uniform}_{\mathsf{bid}}(B_{\preceq}, A_{\prec}, M)$;
  `// Otherwise,` b `and` a `are not tradable`
  return $\mathsf{Uniform}_{\mathsf{bid}}(B_{\succeq}, A_{\succeq}, M)$;
end

$\mathsf{Uniform}_{\mathsf{ask}}$ is similar to $\mathsf{Uniform}_{\mathsf{bid}}$. Observe that, as opposed to the earlier $O(n \log n)$-time algorithm, the `Match` subroutine is not being provided sorted lists of bids and asks, but instead it is given a list of bids and asks of equal volume where each bid and each ask are tradable, and consequently it produces an exhaustive matching. Having described our algorithm $\mathsf{Uniform}$, we now turn to prove its correctness. The main theorem that establishes the correctness is as follows.

▶ **Theorem 17.** *Given a list of bids* B *and a list of asks* A*, let* $\mathsf{OPT} = \mathtt{UM}(B, A)$ *and* $M = \mathsf{Uniform}(B, A)$*. Then, for each order* $w \in B \cup A$*,* $\mathsf{Qty}(w, \mathsf{OPT}) = \mathsf{Qty}(w, M)$*.*

Once we prove the above theorem, the correctness follows from the following proposition and Theorem 13.

▶ **Proposition 18.** *If* M_1 *is a uniform price matching over* (B, A) *and* M_2 *is a matching over* (B, A) *such that for all* $w \in B \cup A$*,* $\mathsf{Qty}(w, M_1) = \mathsf{Qty}(w, M_2)$*, then* M_2 *is a uniform price matching.*

The proposition is obvious: the volumes of M_1 and M_2 must be the same from the condition above. Also, since the same orders participate in both M_1 and M_2, transactions in M_2 can be assigned the same uniform price that is in M_1. Finally, fairness also follows immediately since the more competitive orders are fully traded in M_1, they must be fully traded in M_2 as $\mathsf{Qty}(w, M_1) = \mathsf{Qty}(w, M_2)$ for all orders w.

We now turn to proving Theorem 17.

Proof of Theorem 17. For a list of orders Ω, we use $\Omega\!\downarrow$ to denote the list obtained by sorting Ω by decreasing competitiveness.

We make the following claim.

▷ Claim 19. Let B be a list of bids, and A be a list of asks, with $\mathsf{Vol}(B) = \mathsf{Vol}(A)$, and let M be a matching over (B, A). Let $M_1 = \mathtt{Match}(B\!\downarrow, A\!\downarrow, M)$, $M_2 = \mathsf{Uniform}_{\mathsf{bid}}(B, A, M)$, and $M_3 = \mathsf{Uniform}_{\mathsf{ask}}(B, A, M)$. Then, for all $\omega \in B \cup A$, $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_2) = \mathsf{Qty}(\omega, M_3)$.

If the claim is true, then Theorem 17 follows immediately by observing that without loss of generality we had assumed that our inputs B and A are such that $\mathsf{Vol}(B) = \mathsf{Vol}(A)$ (which was achieved by adding a dummy untradable order), $\mathtt{UM}(B, A) = \mathtt{Match}(B\!\downarrow, A\!\downarrow, \emptyset)$, and $\mathsf{Uniform}(B, A) = \mathsf{Uniform}_{\mathsf{bid}}(B, A, \emptyset)$.

We now prove the claim by induction on $|B| + |A|$.

We focus on showing the following part: for all $\omega \in B \cup A$, $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_2)$. A symmetric argument will yield $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_3)$.

The base cases include $|B| = |A| = 0$ and $|B| = |A| = 1$. The proof in these cases follows easily.

Thus, we are left with cases where $|B| \geq 1$, $|A| \geq 1$, and $|B| + |A| \geq 3$. Now, we argue that it suffices to consider cases where $|B| \geq 2$. Let us analyze what happens when we run $\mathsf{Uniform}_{\mathsf{bid}}$ on (B, A, M), where $|B| = 1$ and $|A| \geq 2$. $\mathsf{Uniform}_{\mathsf{bid}}$ will compute $B_{\succeq} = B$ and $A_{\succeq} = A$ (as B is a singleton and $\mathsf{Vol}(B) = \mathsf{Vol}(A)$). $\mathsf{Uniform}_{\mathsf{bid}}$ checks whether the "median" bid b (the only bid in B) and the ask a (the least competitive ask in A) are tradable or not. If they are tradable, then each order in $\omega \in B \cup A$ will be exhaustively matched. Since it is a uniform price matching, every bid-ask pair is tradable, so $\mathtt{Match}$ on $(B{\downarrow}, A{\downarrow}, M)$ will also match every order in $B \cup A$ exhaustively, and the claim follows easily. If they are not tradable, then $\mathsf{Uniform}_{\mathsf{bid}}$ will return $\mathsf{Uniform}_{\mathsf{ask}}(B, A, M)$, i.e., $\mathsf{Uniform}_{\mathsf{bid}}(B, A, M) = \mathsf{Uniform}_{\mathsf{ask}}(B, A, M)$, and in this case $|A| \geq 2$ and will be handled when we apply the symmetric argument to prove $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_3)$.

Thus, we may assume that $|B| \geq 2$ and this will imply that both $B_{\succeq}$ and $B_{\prec}$ will turn out to be proper subsets of B when we run $\mathsf{Uniform}_{\mathsf{bid}}$ on (B, A, M).

We fix sets B, A, and M such that $|B| \geq 2$. Also, $M_1 = \mathtt{Match}(B{\downarrow}, A{\downarrow}, M)$ and $M_2 = \mathsf{Uniform}_{\mathsf{bid}}(B, A, M)$. We need to show that for all $\omega \in B \cup A$, $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_2)$.

In the proof below, we will be using the following facts that hold for arbitrary B', A', and M'.

- $\mathtt{UM}(B', A') = \mathtt{Match}(B'{\downarrow}, A'{\downarrow}, \emptyset)$;
- $\mathtt{Match}(B', A', M') = M' \cup \mathtt{Match}(B', A', \emptyset)$;
- $\mathsf{Uniform}_{\mathsf{bid}}(B', A', M') = M' \cup \mathsf{Uniform}_{\mathsf{bid}}(B', A', \emptyset)$;
- $\mathsf{Uniform}_{\mathsf{ask}}(B', A', M') = M' \cup \mathsf{Uniform}_{\mathsf{ask}}(B', A', \emptyset)$, where the union is a disjoint union (which is a list concatenation operation when thinking of the matchings as lists).

$\mathsf{Uniform}_{\mathsf{bid}}$ on (B, A, M) first finds the "median" bid b, and ask a obtained by running $\mathtt{SplitQ}(A, \mathsf{Vol}(B_{\succeq}))$, and the partitions $(B_{\succeq}, B_{\prec})$ of B, $(A_{\succeq}, A_{\preceq})$ of A. Let $q = \mathsf{Vol}(B_{\succeq}) = \mathsf{Vol}(A_{\succeq})$. $\mathsf{Uniform}_{\mathsf{bid}}$ then checks whether b and a are tradable which gives rise to two cases.

Case: b and a are tradable. In this case, $B_{\succeq}$ is matched completely with $A_{\succeq}$ to produce a matching M' with quantity q and the final output matching is

$$M_2 = \mathsf{Uniform}_{\mathsf{ask}}(B_{\prec}, A_{\preceq}, M \cup M') = M \cup M' \cup \mathsf{Uniform}_{\mathsf{ask}}(B_{\prec}, A_{\preceq}, \emptyset).$$

Also, $\mathtt{Match}(B{\downarrow}, A{\downarrow}, M) = M \cup \mathtt{Match}(B{\downarrow}, A{\downarrow}, \emptyset) = M \cup \mathtt{UM}(B, A)$. We now invoke Lemma 16 by setting $\hat{b}$ to b to argue that the matching output by $\mathtt{UM}$ on (B, A) is $\mathtt{UM}(B_{\succeq}, A_{\succeq}) \cup \mathtt{UM}(B_{\prec}, A_{\preceq})$. Thus,

$$M_1 = M \cup \mathtt{UM}(B_{\succeq}, A_{\succeq}) \cup \mathtt{UM}(B_{\prec}, A_{\preceq}) = M \cup \mathtt{UM}(B_{\succeq}, A_{\succeq}) \cup \mathtt{Match}(B_{\prec}{\downarrow}, A_{\preceq}{\downarrow}, \emptyset).$$

Note that we have expressed both M_1 and M_2 as a disjoint union (list concatenation) of three sets. Fix an $\omega \in A \cup B$. Now, $\mathsf{Qty}(\omega, M) = \mathsf{Qty}(\omega, M)$ (trivially), $\mathsf{Qty}(\omega, M') = \mathsf{Qty}(\omega, \mathtt{UM}(B_{\succeq}, A_{\succeq}))$ (as M' is obtained by exhaustively matching all orders in $(B_{\succeq}, A_{\succeq})$), and $\mathsf{Qty}(\omega, \mathsf{Uniform}_{\mathsf{ask}}(B_{\prec}, A_{\preceq}, \emptyset)) = \mathsf{Qty}(\omega, \mathtt{Match}(B_{\prec}{\downarrow}, A_{\preceq}{\downarrow}, \emptyset))$ (from induction). Thus, we have $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_2)$.

Case: b and a are not tradable. $\mathsf{Uniform}_{\mathsf{bid}}$ completely discards $B_{\prec}$ and $A_{\preceq}$ and outputs the matching

$$M_2 = \mathsf{Uniform}_{\mathsf{ask}}(B_{\succeq}, A_{\succeq}, M) = M \cup \mathsf{Uniform}_{\mathsf{ask}}(B_{\succeq}, A_{\succeq}, \emptyset).$$

Also, $M_1 = \texttt{Match}(B{\downarrow}, A{\downarrow}, M) = M \cup \texttt{Match}(B{\downarrow}, A{\downarrow}, \emptyset)$. Observe that no bid in $B_{\prec}$ is tradable with any ask in $A_{\preceq}$, as bids in $B_{\prec}$ are strictly less competitive than b and asks in $A_{\preceq}$ are at most as competitive as a, and b and a are not tradable. We further claim that $\texttt{Match}(B{\downarrow}, A{\downarrow}, \emptyset) = \texttt{UM}(B, A)$ does not match any orders from $B_{\prec} \cup A_{\preceq}$. To see this, we invoke Lemma 15. Note that except for bid b and ask a, the respective ranges of orders in $B_{\succeq}$ and $A_{\succeq}$ have numbers strictly less than q and the respective ranges of orders in $B_{\prec}$ and $A_{\preceq}$ have numbers that are all strictly greater than q, as $q \in \mathsf{Range}_A(a) \cap \mathsf{Range}_B(b)$. Thus, any potential matches between $B_{\prec}$ and $A_{\succeq}$ or between $A_{\preceq}$ and $B_{\succeq}$ can only happen between b and a, but they are not tradable (as per the case). Thus, we conclude that the $\texttt{UM}(B, A) = \texttt{UM}(B_{\succeq}, A_{\succeq})$. Therefore,

$$M_1 = M \cup \texttt{Match}(B_{\succeq}{\downarrow}, A_{\succeq}{\downarrow}, \emptyset).$$

From induction, arguing as before, we get for all orders $\omega \in A \cup B$, $\mathsf{Qty}(\omega, M_1) = \mathsf{Qty}(\omega, M_2)$, and we are done. ◀

We now analyze the running time of Uniform. Let $T(n)$ represent the running time of Uniform, where the number of orders $|B| + |A| = n$.

Uniform calls $\mathsf{Uniform}_{\mathsf{bid}}$, which in turn calls $\mathsf{Uniform}_{\mathsf{ask}}$ after decreasing the number of the bids by a factor of two. $\mathsf{Uniform}_{\mathsf{ask}}$ then calls $\mathsf{Uniform}_{\mathsf{bid}}$ again after decreasing the number of the asks by a factor of two. So after two successive returns, we can see that the number of bids and asks decreases by a factor of two. Also, since all the subroutines take linear time, by simple inspection, we conclude

$T(n) \leq T(\lceil \frac{n}{2} \rceil + 1) + cn$, where c is an absolute constant. Thus, clearly $T(n) = O(n)$.

This completes the proof of Theorem 7.

4 Conclusions

The problems we consider are clearly of fundamental interest and we achieve asymptotically tight results for them using elementary techniques. Surprisingly, despite their fundamental nature and wide practical applicability, prior to this work, the complexity aspects of such problems were not deeply studied. The following natural questions arise from our work.

- In this work, the most classical exchange model is assumed; there are execution principles other than price-time priority (like pro-rata matching) which are also being employed in the real world. These alternative principles present opportunities for studying algorithmic complexity beyond the traditional price-time priority model.
- Furthermore, it might be interesting to consider similar problems in the context of decentralized exchanges.
- Finally, bridging the gap between the upper and lower bounds on the time complexity of Problem 1 in the binary query model remains open.

References

1 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

2 Ravi B. Boppana. The decision-tree complexity of element distinctness. *Inf. Process. Lett.*, 52(6):329–331, 1994. doi:10.1016/0020-0190(94)00154-5.

3 Kaustubh Deshmukh, Andrew V. Goldberg, Jason D. Hartline, and Anna R. Karlin. Truthful and competitive double auctions. In Rolf H. Möhring and Rajeev Raman, editors, *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*, volume 2461 of *Lecture Notes in Computer Science*, pages 361–373, Rome, Italy, 2002. Springer. doi:10.1007/3-540-45749-6_34.

4 Mohit Garg, N. Raja, Suneel Sarswat, and Abhishek Kr Singh. Double auctions: Formalization and automated checkers. *J. Autom. Reason.*, 69(3):17, 2025. doi:10.1007/S10817-025-09732-X.

5 Mohit Garg and Suneel Sarswat. The design and regulation of exchanges: A formal approach. In Anuj Dawar and Venkatesan Guruswami, editors, *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022, December 18-20, 2022, IIT Madras, Chennai, India*, volume 250 of *LIPIcs*, pages 39:1–39:21, Chennai, India, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2022.39.

6 Mohit Garg and Suneel Sarswat. Efficient and verified continuous double auctions. In Nikolaj S. Bjørner, Marijn Heule, and Andrei Voronkov, editors, *LPAR 2024 Complementary Volume, Port Louis, Mauritius, May 26-31, 2024*, volume 18 of *Kalpa Publications in Computing*, pages 1–13, Port Louis, Mauritius, 2024. EasyChair. doi:10.29007/92JT.

7 Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*. Monographs in Theoretical Computer Science. An EATCS Series. Springer Berlin, Heidelberg, Berlin, Heidelberg, 1 edition, 1984. doi:10.1007/978-3-642-69672-5.

8 L. Mirsky. A dual of dilworth's decomposition theorem. *The American Mathematical Monthly*, 78(8):876–877, 1971. URL: http://www.jstor.org/stable/2316481.

9 Raja Natarajan, Suneel Sarswat, and Abhishek Kr Singh. Verified double sided auctions for financial markets. In Liron Cohen and Cezary Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference)*, volume 193 of *LIPIcs*, pages 28:1–28:18, Rome, Italy, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICS.ITP.2021.28.

10 Jinzhong Niu and Simon Parsons. Maximizing matching in double-sided auctions. In Maria L. Gini, Onn Shehory, Takayuki Ito, and Catholijn M. Jonker, editors, *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 1283–1284, Saint Paul, MN, USA, 2013. IFAAMAS. URL: http://dl.acm.org/citation.cfm?id=2485184.

11 Jaikumar Radhakrishnan. Entropy and counting. *Computational mathematics, modelling and algorithms*, 146, 2003.

12 Peter R. Wurman, William E. Walsh, and Michael P. Wellman. Flexible double auctions for electronic commerce: theory and implementation. *Decis. Support Syst.*, 24(1):17–27, 1998. doi:10.1016/S0167-9236(98)00060-8.

13 Dengji Zhao, Dongmo Zhang, Md Khan, and Laurent Perrussel. Maximal matching for double auction. In Jiuyong Li, editor, *AI 2010: Advances in Artificial Intelligence - 23rd Australasian Joint Conference, Adelaide, Australia, December 7-10, 2010. Proceedings*, volume 6464 of *Lecture Notes in Computer Science*, pages 516–525, Adelaide, Australia, 2010. Springer. doi:10.1007/978-3-642-17432-2_52.

Measuring CEX-DEX Extracted Value and Searcher Profitability: The Darkest of the MEV Dark Forest

Fei Wu
King's College London, UK

Danning Sui
Flashbots, San Francisco, CA, USA

Thomas Thiery
Ethereum Foundation, Lisbon, Portugal

Mallesh Pai
Rice University, Houston, TX, USA
Tempo Labs, Houston, TX, USA
Paradigm, Houston, TX, USA

Abstract

This paper provides a comprehensive empirical analysis of the economics and dynamics behind arbitrages between centralized and decentralized exchanges (CEX-DEX) on Ethereum. We refine heuristics to identify arbitrage transactions from on-chain data and introduce a robust empirical framework to estimate arbitrage revenue without knowing traders' actual behaviors on CEX. Leveraging an extensive dataset spanning 19 months from August 2023 to March 2025, we estimate a total of 233.8M USD extracted by 19 major CEX-DEX searchers from 7,203,560 identified CEX-DEX arbitrages. Our analysis reveals increasing centralization trends as three searchers captured three-quarters of both volume and extracted value. We also demonstrate that searchers' profitability is tied to their integration level with block builders and uncover exclusive searcher-builder relationships and their market impact. Finally, we correct the previously underestimated profitability of block builders who vertically integrate with a searcher. These insights illuminate the darkest corner of the MEV landscape and highlight the critical implications for Ethereum's decentralization.

2012 ACM Subject Classification Applied computing → Electronic commerce

Keywords and phrases Decentralized Finance, Maximal Extractable Value, CEX-DEX arbitrages

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.26

Related Version *Full Version*: `arXiv:2507.13023`

Supplementary Material *Dataset*: `https://dune.com/rig_ef/cex-dex-dash`

Funding The work is supported by The Latest in DeFi Research (TLDR) Fellowship 2024 funded by Uniswap Foundation.

Acknowledgements We appreciate Gaussian Process, Kevin Pang, Bill Zhang, Tom Zhao, Romain Butteaud, Burak Öz and Max Resnick for for helpful discussions and comments. We also thank Alexander Tesfamichael from Ultra Sound Money and Žan Knafelc from Eden Network for their help with data collection.

1 Introduction

Over recent years, the focus of study around Maximal Extractable Value (MEV) extraction has broadened beyond on-chain atomic strategies to a richer landscape of more sophisticated strategies across different venues. Early studies primarily investigated MEV extracted solely from the blockchain's internal state and mempool, termed atomic MEV [19, 44, 45]. Subsequent research has turned attention towards non-atomic MEV, strategies that leverage

© Fei Wu, Danning Sui, Thomas Thiery, and Mallesh Pai;
licensed under Creative Commons License CC-BY 4.0

external price information alongside on-chain execution, most notably arbitrage between centralized exchanges (CEX) and decentralized exchanges (DEX) [42, 34, 62]. These CEX-DEX arbitrages have emerged as an economically substantial type of MEV.

CEX-DEX arbitrageurs (searchers) capitalize on temporary price discrepancies arising from asynchronous price discovery across venues. While centralized exchanges provide high liquidity and near-instantaneous trade execution, decentralized exchanges experience liquidity constraints and inherent latency due to blockchain settlement delays. These structural frictions regularly produce significant, short-lived deviations in asset prices on DEXes from their fair value, offering lucrative arbitrage opportunities for sophisticated market participants.

Despite high entry barriers such as capital requirements, low-latency infrastructure, inventory risk, and uncertainty of block inclusion, CEX-DEX arbitrage remains remarkably profitable [48, 17]. Such trades often take up less than 2% of block space but contribute more than 15% of the total block value [34, 43]. Consequently, under Ethereum's Proposer-Builder Separation (PBS) framework, block builders with privileged access to such arbitrage flows benefit from a decisive advantage in winning block auctions. At the time of writing, three builders `beaverbuild`, `Titan`, and `rsync` dominate the Ethereum builder market, two of which vertically integrate their own CEX-DEX searchers [43, 34, 58]. Such vertical integration raises important concerns for Ethereum's decentralization and security: it fosters economies of scale that strengthen dominant players, enables monopoly pricing that causes proposer loss [58], and increases vulnerability to censorship and commitment attacks [53, 33, 30].

Although theoretical models [40, 39, 41] and prior empirical studies [34, 43, 58, 17] have illuminated the existence and broader impact of CEX-DEX arbitrage on the Ethereum builder market, the economic details remain unclear. Accurately identifying such transactions on-chain and estimating realized revenue without knowing searchers' off-chain behaviors is particularly challenging, rendering CEX-DEX arbitrage arguably "the darkest part of the MEV dark forest."

This paper provides a rigorous empirical investigation into value extraction, searcher profitability, and market structure effects of CEX-DEX arbitrage on Ethereum. Integrating DEX trades data, CEX pricing, and MEV-Boost data over 19 months from August 2023 to March 2025 and quantifying *who* earns *what*, our results supply the missing inputs needed to reason about decentralization guarantees. Our contributions are summarized as follows:

1. We refine existing heuristics for identifying CEX-DEX arbitrage transactions [34, 43], significantly expanding coverage and accuracy.
2. Without directly observing the CEX leg of searchers' CEX-DEX arbitrages, we develop an empirical framework to infer the likely CEX execution by tracking when each searcher's information advantage maximizes and begins to erode, thereby inferring from each on-chain swap a realized arbitrage revenue. This data-driven proxy matches theoretical price-impact intuition and enables consistent comparisons between CEX-DEX searchers.
3. We demonstrate that token liquidity influences how searchers extract value and hedge: high-liquidity tokens enable searchers to place large orders at tight spreads with minimal market impact, whereas low-liquidity tokens present wider spreads but restrict them to smaller trades that incur larger price impact.
4. We estimate that 19 major searchers extracted a total value of 233.8M USD from 7,203,560 CEX-DEX arbitrages during the observed period and uncover an increasing centralization trend, with the three leading searchers capturing approximately three-quarters of the total arbitrage volume and extracted value.
5. Our results reveal a clear profit-sharing pattern tied to searchers' integration levels with block builders. Neutral searchers, distributing flow among multiple builders, retain higher profit margins. Conversely, exclusive searchers maintain lower margins, sharing

most of their revenue with affiliated builders, sometimes even operating at negative net profit. Additionally, we find that the searcher-builder exclusivity deal is likely to mutually reinforce competitive positions for both the searcher and the builder.

6. By accounting for searcher profits, we correct previously understated estimates of integrated builder profits and subsidies and better illuminate their profitability.

2 Background and Related Works

Ethereum, Proposer-Builder Separation, and MEV-Boost. Ethereum advances in discrete 12-second slots. At each slot, a randomly selected validator – the beacon proposer – publishes a new block. To enhance validator decentralization and censorship resistance, Proposer-Builder Separation (PBS) allows proposers to outsource block construction tasks to specialized entities called *builders*. PBS is currently implemented via MEV-Boost [27], an out-of-protocol solution where builders compete in MEV-Boost auctions by submitting blocks alongside bids to trusted intermediaries known as *relays*. Builders submit bids for slot n starting from the start of slot $n-1$, funded by tips and payments alongside transactions from users and MEV searchers. At the start of slot n, the proposer signs the highest-bid block header from any relay to which the proposer is connected, after which the winning relay publishes the block.

Maximal Extractable Value and its supply chain. Maximal Extractable Value (MEV) refers to profits obtainable through strategic transaction ordering, inclusion, or censorship within a block. MEV extraction can rely solely on internal blockchain state (atomic MEV) [19, 45, 61, 44], or incorporate external market information (non-atomic MEV), such as prices from centralized exchanges and decentralized exchanges on other blockchains [34, 62, 42].

Under PBS, MEV extraction is typically performed by specialized searchers. The competitive nature of MEV extraction gives rise to a structured MEV supply chain: searchers share with builders part of their extracted value to ensure prioritized inclusion of their transactions. Builders subsequently share a portion of these MEV revenues with proposers through competitive bids in MEV-Boost auctions. Such intense competition incentivizes *vertical integration*, where entities optimize MEV extraction by simultaneously controlling upstream (searcher-level strategies) and downstream (builder and relay infrastructure) activities, thus securing a strategic advantage in capturing valuable MEV opportunities.

CEX-DEX Arbitrages. CEX-DEX arbitrages represent a prominent category of non-atomic MEV. Typically, these arbitrages involve two complementary trades: one executed on-chain via a decentralized exchange, and the other executed off-chain via a centralized exchange, in this order. Figure 1 illustrates an example of a CEX-DEX operation. First, searchers identify price discrepancies between assets listed on a DEX and a CEX. Based on the observed price differences, searchers form an *ex-ante expectation* of the extractable value (i.e., arbitrage revenue) once hedged. They then initiate the arbitrage by submitting their DEX trade to builders, offering a fraction of their expected arbitrage revenue as payments to the builder to secure prioritized block inclusion. Once the DEX trade is confirmed on-chain, searchers swiftly execute an offsetting hedge trade on the CEX to realize the profit.

Related Works. [40, 39, 41] introduced the concept of Loss-Versus-Rebalancing (LVR), modeling losses of DEX liquidity providers due to informed arbitrageurs, establishing a theoretical benchmark for CEX-DEX arbitrage revenue. [18] characterized the extractable value by informed searchers and showed that MEV extraction grows sophisticated with more

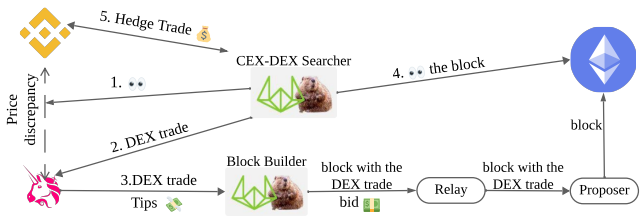


Figure 1 An example of a CEX-DEX arbitrage operation.

informed trading, notably CEX-DEX arbitrages. The authors further elaborated on the risks and entry barriers of CEX-DEX arbitrage operation and observed that CEX-DEX arbitrage primarily involves trading high-liquidity tokens [17]. [16] provided initial insights into CEX-DEX arbitrages and searcher-builder relationships, and estimated searcher profits from trades in the Uniswap V2 WETH-USDC pool. Expanding this, [47] investigated CEX-DEX arbitrage activities in all Uniswap V2 and V3 pools, highlighting searcher-builder integration and economies of scale in the Ethereum builder market. [34] empirically studied CEX-DEX arbitrages identified through comprehensive heuristics they introduce, uncovering a strong correlation between arbitrage volumes and token price volatility, and demonstrating builders integrated with CEX-DEX searchers perform better in MEV-Boost auctions during volatile periods. The work also provided evidence for vertical integration between specific searchers and builders, such as `beaverbuild` and `rsync`. Similarly, [32] revealed how builders operated by high-frequency trading firms leverage access to CEX-DEX arbitrage flows to gain competitive advantages in block building. [43] empirically demonstrated that builders with exclusive providers, such as CEX-DEX searchers, enjoy increased market share and profitability. Similar to [34], the work found that CEX-DEX arbitrage flow contributes to a prominent share of the block value. [57, 56] conducted simulations and empirical game-theoretic analysis to show builders' strategic advantages in block auctions with exclusive access to high-value orderflow. [58] further analyzed how vertical integration adversely impacts proposer revenue, block optimality, and censorship resistance. Lastly, longitudinal studies by [54, 33] provided empirical overviews of Ethereum's evolving builder market, highlighting centralization and censorship pressures under PBS.

3 Identifying CEX-DEX Arbitrage Transactions

CEX-DEX arbitrage transactions are typically more challenging to identify than other types of MEV transactions [61, 45, 44], primarily because only the DEX leg of such arbitrages is explicitly observable on-chain. Nonetheless, by exploiting certain properties of the transaction and behavioral regularities of CEX-DEX MEV searchers, we can infer whether a given DEX trade can be part of a CEX-DEX arbitrage. Specifically, we assume that CEX-DEX arbitrage transactions are private and prioritized trades in the block executed by specialized MEV searchers, who generally do not interact with regular users.

Our methodology builds upon and extends previous approaches [34, 43, 16], refining existing heuristics and introducing additional filtering mechanisms to detect CEX-DEX arbitrage transactions from the broader set of DEX trades. We implement these detection methods using Dune Analytics [2], leveraging established datasets including [9, 35, 26, 25].

A significant advancement in our approach is the expansion beyond the constraints of previous research, which typically limited CEX-DEX trades to transactions that contained exactly one on-chain swap (or two ERC-20 token transfers) [34, 43]. Our investigation reveals

that certain CEX-DEX searchers often execute multiple swaps across different DEX pools within a single transaction to optimize on-chain liquidity utilization.[1] These transactions do not fit into the traditional heuristic of one swap and two ERC-20 token transfers. Our methodology accommodates these multi-swap transactions by aggregating sequential swaps and reconstructing the final effective trading token pair with corresponding aggregate traded amounts. Specifically, we apply the following set of formulated heuristics:

Heuristic 1. The transaction is *private*, i.e., it is not observed in any public mempool [26] prior to its on-chain inclusion.

Heuristic 2. At least one of the swaps included in the transaction is the first swap executed in the respective direction and DEX pool for the given block.

Heuristic 3. The transaction is not categorized by existing frameworks [5, 35] or our own Dune query as any type of atomic MEV activity (e.g., sandwich attacks or atomic arbitrages). The transaction does not include a liquidation event.

Heuristic 4. The transaction is not a backrun transaction associated with an Orderflow Auction (OFA) bundle.

Heuristic 5. The transaction is not submitted to any known router smart contract identified in [25], nor to any trading bot that is labeled in [24] or controls an Externally Owned Account (EOA) with an Ethereum Name Service (ENS) name attached.

Heuristic 6. The transaction contains no ERC-721 token transfer and settles as a swap between two tokens that are both listed on a major centralized exchange (e.g., Binance) after all intermediate swaps.

Heuristic 1 and 2 ensure transaction prioritization within blocks and favorable execution pricing in at least one DEX pool. Heuristics 3, 4, and 5 eliminate potential confusion with other MEV activities or retail trades executed by non-MEV bots. In particular, Heuristic 5 recognizes that bot contracts controlling EOAs with attached ENS names typically interact with regular users, suggesting they likely function as telegram bots or trading front-ends rather than specialized CEX-DEX MEV bots. For Heuristic 6, we cross-verified ERC-20 contract addresses for 287 Binance-listed tokens collected from Etherscan [3] and CoinMarketCap [1] with the Dune DEX trades dataset [9], ensuring that we filtered out meme tokens that share the same symbols with major tokens but are not listed on a CEX.

Applying the above criteria, we curate an empirical dataset comprising 8,723,233 transactions that are likely to be CEX-DEX arbitrages across multiple DEXes [9] from block 17866488 to block 21998438 spanning the period from August 8, 2023, to March 8, 2025 with their detailed information, including searcher contract address, fees, payments to the builder, trade volume, and the precise tokens and amounts traded. To the best of our knowledge, this constitutes the most comprehensive empirical dataset to date for analyzing CEX-DEX arbitrages in terms of observation duration, transaction count, and token coverage. We open source the Dune query for reproducibility [49].

3.1 Current CEX-DEX Arbitrage Landscape

Utilizing the curated dataset, we analyze temporal trends in CEX-DEX arbitrage activity over the observed period. To enhance readability and streamline the visualization, we assign intuitive labels to the 23 major searcher entities identified in the dataset with the highest

[1] Consider for example searcher contract address: `0x767c8bb1574bee5d4fe35e27e0003c89d43c5121`, with sample transaction: `0x0488cad7726ed6947be6af09d17b012fd7c986cb32a9d59cd7285b9a4e0926be`.

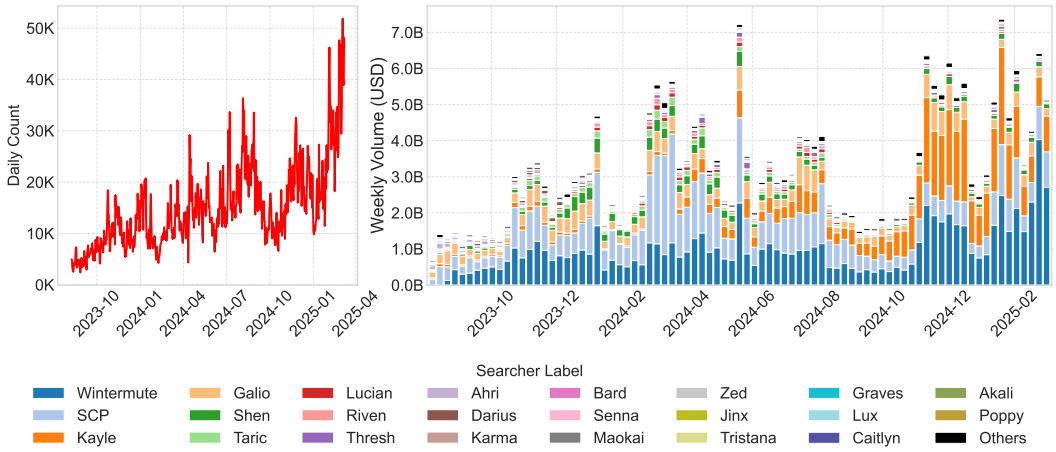


Figure 2 Daily count and weekly volume of detected CEX-DEX trades between August 2023 and March 2025.

total volume in place of their contract addresses, except for `Wintermute` and `SCP`, whose identities are retained as-is. Our subsequent analysis will focus on these 23 searcher entities, who collectively contribute to 99% of total volume.

Figure 2 presents the daily CEX-DEX transaction counts (left panel) alongside the weekly volumes of each searcher entity (right panel). Note that all trades flagged by our heuristics are shown; transactions later excluded by the stricter profitability filter (cf. Section 4.1), which roughly contribute to roughly 15% of the total count and 7% of the total volume, are still included here for landscape context. The daily count of CEX-DEX arbitrage transactions exhibits a clear upward trend over the 19-month period, despite occasional fluctuations. Notably, daily transaction counts increased by 7.2 times by Q1 2025 compared to Q3 2023.

Despite the overall increase in CEX-DEX arbitrage activity, we observe reduced diversity of successful searchers over time. Before October 2024, 23 labeled searchers collectively accounted for nearly 99% of the total CEX-DEX arbitrage volume on-chain. This landscape transformed substantially afterward, with just 14 labeled searchers remaining active and successful. The figure further reduced to 11 by Q1 2025, with three leading searchers capturing 90% of the total volume. Among all searchers, `Wintermute` and `SCP` have maintained dominant positions throughout the entire period. Interestingly, `Kayle`, initially a relatively minor searcher, began to rise significantly in influence around June 2024 and emerged as a leading searcher.

4 Estimating CEX-DEX Arbitrage Revenue and Searcher Profit

We next proceed to introduce our methodology of estimating CEX-DEX arbitrage revenue and searcher profit. We assume that searchers first establish token inventory positions through on-chain DEX trades before subsequently hedging these accumulated positions on off-chain CEX venues. This sequential execution assumption stems from the inherently higher execution uncertainty on-chain compared to centralized venues. When multiple searchers target the same arbitrage opportunity in a specific DEX pool, the probability of any searcher's on-chain transaction being included in a block is uncertain. Consequently, searchers initiate off-chain hedging operations *only after* receiving confirmation that their on-chain trades have been successfully executed, thereby avoiding potential unhedged inventory risk [17].

In addition, we assume that searchers execute the hedge for *both directions* on CEX within a short timeframe or even instantaneously after confirming successful on-chain trade. Concretely, if a searcher buys BTC and sells ETH on the DEX, they will – within a very short window – sell the newly acquired BTC and repurchase ETH on the CEX. This behavior is economically rational given the inherent volatility risk of token inventory left unhedged, particularly for low-liquidity tokens that exhibit higher price volatility [17].

Since searchers' actual hedging details on CEX are unobservable, we present a methodology to estimate their realized arbitrage revenues based on empirical observations.

4.1 Searcher-Specific Optimal Execution Horizons

A key challenge in estimating realized revenue is identifying the timing when searchers typically hedge their DEX positions off-chain. Searchers exhibit notable variations in the timing of their observations of DEX trade confirmations and subsequent CEX hedge, contrasting with standardized `slot time` reported by platforms such as Dune [2] and Etherscan [3]. These timing variations stem from several factors:

1. **Block confirmation:** Searchers observe the block and on-chain transaction executions at varying times due to differences in network connectivity and block propagation path.
2. **Information asymmetry:** Searchers possess different levels of market information (also known as *alpha* signal) affecting their execution timing decisions.
3. **Execution infrastructure:** Variations in the infrastructure of different searchers result in different processing speed. Certain integrated searchers have privileged access to the block building process and can execute their strategy closer to the actual block time.

To capture these searcher-specific timing characteristics, we analyze each searcher's historical trades by examining when their information advantage maximizes, reflected in the maximum achievable spread. To facilitate fair comparisons across different trade sizes, we define the *gross return* (GR) at time t (the *markout horizon*) as the markout revenue per unit of trade volume, i.e., the spread captured per dollar deployed in the arbitrage.

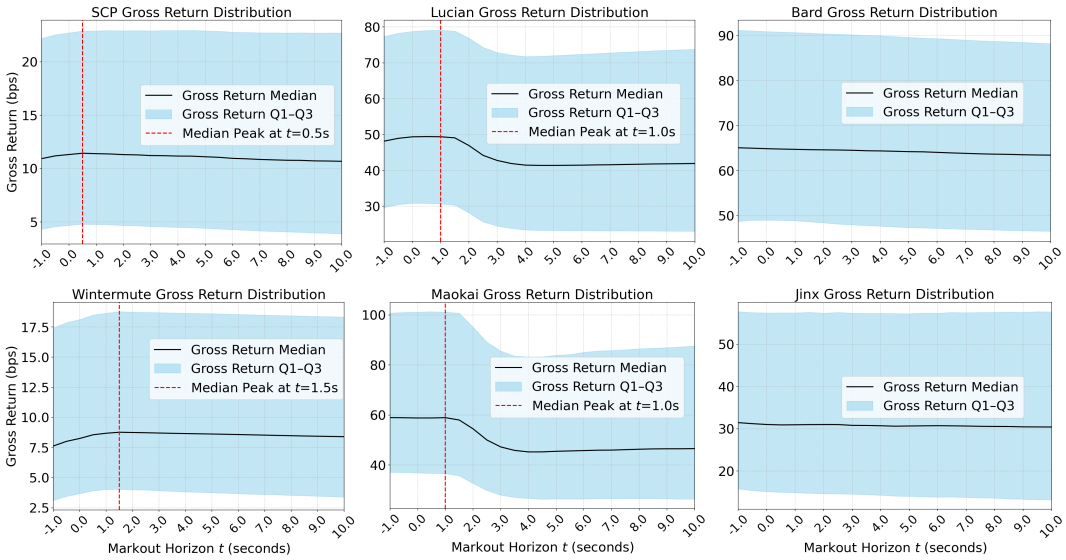
Formally, consider a CEX-DEX arbitrage transaction i with volume V_i that purchases x amount of token A and sells y amount of token B on the DEX, net of liquidity provider fees. Let $P_A(t)$ and $P_B(t)$ represent the respective CEX USD prices at markout horizon t. The markout revenue (MR) and gross return (GR) achievable by flattening the DEX-acquired inventory on CEX at time t can be estimated as:

$$\mathrm{MR}_i(t) = x \cdot P_A(t) - y \cdot P_B(t) - \text{CEX taker fees},^{2} \quad \text{and } \mathrm{GR}_i(t) = \frac{\mathrm{MR}_i(t)}{V_i}. \tag{1}$$

We value all tokens using USD to ensure consistent and accurate pricing by leveraging the superior liquidity of USDT pairs on Binance. To derive accurate USD prices for each token at specific markout horizons, we collect Binance historical quote data for USDT pairs from Tardis.dev [51] and employ the *mid-price* at each relevant markout timestamp.

▶ Remark 1. We assume 1 USDT = 1 USD, as most tokens are priced in USDT on Binance. In contrast, other USD stablecoins (e.g., USDC) may not equal 1 USD due to potential arbitrage opportunities between stablecoins.

[2] We assume CEX-DEX searchers operate at the highest user tier on Binance, thereby benefiting from the lowest applicable fee rate of 0.01725% [12].



(a) Pattern 1: `SCP` & `Wintermute`. **(b)** Pattern 2: `Lucian` & `Maokai`. **(c)** Pattern 3: `Bard` & `Jinx`.

Figure 3 Three empirically observed patterns of gross return distributions across all labeled searchers. Each *column* corresponds to one pattern that contains two representative searchers that exhibit that pattern. The black line is the median, the shaded band the interquartile range, and the dashed red line indicates the markout horizon where the median gross return peaks. $t = 0$ second denotes the `slot time`. The labels of the x-axis and y-axis are shared between plots.

We then calculate the gross return for all trades across a time window spanning from `slot time` $-$ 1 second to `slot time` $+$ 10 seconds, measured at an interval of 0.5 seconds. This interval range was chosen to represent searchers' typical submission time of DEX trades to builders, suggested by builders' bidding timing in the MEV-Boost auctions [8], and a reasonable upper bound for immediate hedging activity following DEX execution without introducing further noise caused by unrelated trades on CEX.

We exclude trades whose token price data is not recorded on Tardis on certain dates [50] and transactions whose markout revenue persistently fails to cover the base fees throughout the interval, indicative of inventory adjustments rather than genuine arbitrage opportunities.[3]

The results reveal distinct temporal patterns in gross return across different searchers. Figure 3 presents the gross return distributions for all trades of 6 representative searchers over the interval of $t \in [-1, 10]$, where $t = 0$ denotes the `slot time`, and reveals 3 different patterns that span all 23 labeled searchers. We here summarize the three patterns observed:

Pattern 1 – Rising or plateau then gentle decay. Gross return climbs (or maintains) at the peak then tapers off gradually (Figure 3a). Representative searchers: `SCP`, `Wintermute`.

Pattern 2 – Rising or plateau then abrupt drop. Gross return climbs (or maintains) at the peak, followed by a rapid decline and a flat tail (Figure 3b). Representative searchers: `Lucian`, `Maokai`.

Pattern 3 – Flat or Minimal decline. Gross return remains flat or declines slightly across the interval with no discernible peak (Figure 3c). Representative searchers: `Bard`, `Jinx`.

For simplicity, we refer to searchers whose gross return behaviors follow Pattern 1, 2, or 3 as Pattern 1, 2, or 3 searchers, respectively.

[3] We observe searchers pay much lower tips to builders for these non-arbitrage transactions. For instance, `SCP` pays on average only 0.0021 ETH for these transactions compared to 0.0081 ETH for arbitrage transactions; similarly, `Wintermute` pays 0.0011 ETH versus 0.0089 ETH for arbitrage transactions.

Pattern 1&2. 19 out of 23 labeled searchers follow the first two patterns and exhibit a characteristic *optimal execution horizon*, identified by their (median) gross return rising to or maintaining at its maximum before declining. These patterns align with theoretical expectations: before hedge execution begins, the spread typically increases as the trader's information advantage manifests in favorable price movements. Once hedging commences, gross return declines as the trade's market impact pushes prices against their position. For instance, we observe that the median gross return of `SCP`'s trades reaches its maximum 0.5 seconds after the `slot time`, while the optimal horizon for `Lucian` and `Maokai` occurs at 1.0 second, for `Wintermute` at 1.5 seconds.

We interpret the *optimal execution horizon* as the most plausible empirical proxy for when searchers initiate their CEX hedge – the instant at which their information advantage is maximized and before the profit opportunity begins to erode due to delaying execution and market impact. Accordingly, we assume each searcher initiates their hedge at their optimal execution horizon empirically identified from all their historical trades. Although a single horizon cannot capture the exact timing of every trade, it summarizes the dominant behavior across all historical trades for each searcher: the cross-trade median peaks are sharp, and the interquartile bands around them are flat, indicating the estimate is not driven by outliers.

Formally, For each searcher j we define the optimal execution horizon t_j^* as the markout horizon that maximizes the cross-trade median of gross return:

$$t_j^* = \arg\max_{t \in \mathcal{T}} \ \text{Median}_{\,i \in \mathcal{I}_j} \big\{ \text{GR}_i(t) \big\},$$

where $\mathcal{T} = \{-1.0, -0.5, 0, \ldots, 10.0\}$ seconds is set of markout horizons around the `slot time` and $\mathcal{I}_j$ denotes the full set of historical trades of searcher j. If multiple markout horizons attain the same maximum before the median gross return decreases, we select the largest t. Given t_j^*, the revenue (i.e., extractable value) and net *profit and loss* (PnL) for any trade $i \in \mathcal{I}_j$ are estimated as

$$\widehat{\text{EV}}_i = \text{MR}_i\big(t_j^*\big) - \text{base_fees}_i, \ \text{and} \ \widehat{\text{PnL}}_i = \widehat{\text{EV}}_i - \text{builder_tips}_i, \tag{2}$$

where builder tips include priority fees and coinbase transfer.[4] The *profit margin* is thus:

$$\text{PM}_i = \frac{\widehat{\text{PnL}}_i}{\widehat{\text{EV}}_i}, \widehat{\text{EV}}_i > 0.$$

We compute the profit margin only for trades whose estimated revenue is positive. Trades with $\widehat{\text{EV}}_i \leq 0$ are labeled as "N/A" for margin calculations and excluded from profit margin analysis, but they are retained in all other analyses related to PnL directly.[5]

Our approach to inferring CEX-DEX searchers' optimal execution horizon builds on methodologies from informed trading and market microstructure literature. Specifically, to identify this timing for each searcher, we adapt strategies from [10, 37], who empirically recover hidden execution timing of informed traders from observable post-trade price movements, by observing how CEX price movements affect the spread of their trades around the `slot time`.

[4] For calculation, we convert the price unit of base fees, priority fees, and coinbase transfer from ETH to USD using the ETH-USDT mid-price at the `slot time`.

[5] For example, consider a trade with an estimated revenue of -1 USD, for which the searcher pays 10 USD to the builder. This results in a net PnL of -11 USD. Although the trade incurs a loss, computing the profit margin would yield an illogical value of +1100%, which motivates its exclusion from margin-based analysis.

Moreover, our data-driven framework aligns with theoretical predictions by [7, 11], suggesting each trader possesses a unique, liquidity-driven optimal execution horizon characterized by a distinct "peak-then-decay" pattern in returns (cf. Section 5.2). While our method necessarily provides a simplified approximation due to unobservable hedging price-impact costs, it nonetheless yields a consistent, empirically grounded upper-bound estimate of extracted value and PnL across searchers.

Pattern 3. However, for four searchers – `Bard`, `Jinx`, `Tristana` and `Lux` – their gross return curve is essentially flat (or drifts only marginally downward) across the $[-1, 10]$ seconds window, so no clear "peak-then-decay" point emerges. The most plausible interpretation is that these searchers hold their inventory and do not hedge their positions on CEX within this window: if they were unwinding inventory promptly, the resulting price impact on the CEX leg should depress the gross return. More explanations can be found in Section 5.2.

While extending the window further than +10 seconds might eventually reveal such a decline, that comes at the cost of substantial noise – price moves unrelated to the original trade. As our revenue and PnL estimation relies on identifying an optimal execution horizon, it cannot be applied reliably to these searchers. We therefore exclude these four searchers from subsequent revenue and PnL calculations, whose trades together account only for 2.7% of total transactions detected; they remain in the descriptive landscape figures but are omitted from revenue and PnL analyses.

5 Token Liquidity as the Driver

In this section, we demonstrate that the distinct patterns of the gross return curve observed across searchers are associated with the difference in the token liquidity they trade, which shapes searchers' value extraction regimes and subsequent hedge execution.

5.1 Liquidity and Value Extraction Regimes

As shown in Figures 3, Pattern 2 searchers earn markedly higher gross returns compared to Pattern 1 searchers. To highlight the role of token liquidity, we group tokens into "Major" – large-cap assets and commonly traded stablecoins (WETH, WBTC, USDT, USDC, TUSD, FDUSD, BUSD, DAI) – and "ALT", the remaining long-tail assets, and compare the trading pairs between these searchers.

We find that Pattern 1 searchers such as `SCP` and `Wintermute` predominantly engage with highly liquid major tokens characterized by deep order books. In contrast, Pattern 2 searchers – typically smaller and less competitive – tend to focus more on lower-liquidity ALT tokens. Prices of these assets show higher volatility and suffer significant slippage even with modest trade volumes. This suggests that differences in token liquidity may influence both the achievable spread and the revenue strategies adopted by different searchers. We illustrate the empirical relationship between trade size and gross return in Figure 4.

From Figure 4a, we observe that Pattern 1 searchers frequently execute trades exceeding 10,000 USD, leveraging the liquidity depth of major tokens to deploy substantial capital without significant price impact. However, they only earn single-digit basis-point gross returns. Figure 4b confirms that nearly 90% of their trades earn less than 20 bps gross return. Despite lower spreads per trade, these searchers accumulate substantial total revenue by leveraging large-scale execution. This pattern is particularly evident for leading searchers.

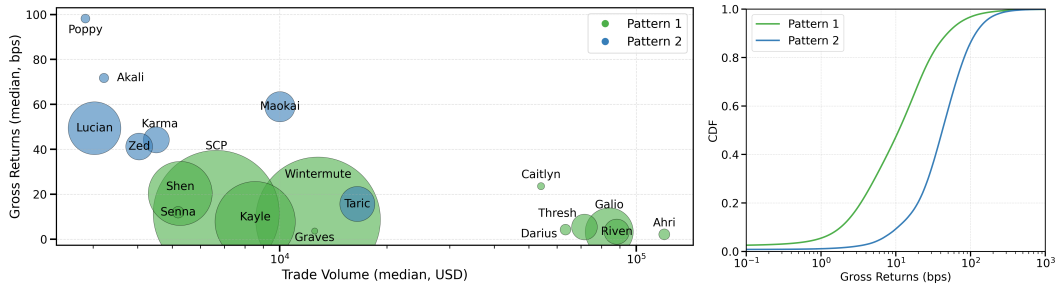


Figure 4 **a)** Median trade volume and gross returns for each searcher, with bubble size indicating searchers' total estimated revenue and color encoding their execution pattern. **b)** Cumulative Distribution Function (CDF) of gross returns across all trades between Pattern 1 and 2 searchers.

Conversely, Pattern 2 searchers face constrained liquidity conditions, typically restricting median trade sizes below 5,000 USD to avoid significant self-induced slippage. Nonetheless, limited liquidity frequently creates greater mispricing opportunities for long-tail assets, resulting in markedly higher gross returns ranging from 20 to 120 bps per trade. Figure 4b confirms that 50% of these searchers' trades exceed 40 bps in gross return.

This divergence reveals two distinct strategies to extract value: large-volume trade with narrow-spread targeting major tokens (Pattern 1) versus small-volume trade with wider-spread targeting long-tail tokens (Pattern 2). Interestingly, certain searchers exhibit hybrid strategies: `Maokai` and `Taric`, while categorized as Pattern 2, sustain relatively large trading volumes by maintaining significant activity in Major-Major pairs. `Shen` and `Senna`, categorized as Pattern 1, engage in notable ALT token trading, resulting in comparatively smaller volumes.

5.2 Liquidity and Hedge Execution

Variations in the speed and magnitude of spread closure after peaking between Pattern 1 and 2 observed in Figure 3 can now be better understood as the value extraction regimes above manifest mechanically in how quickly spreads collapse once the hedge is executed.

As described above, Pattern 1 searchers focus primarily on deeply liquid major tokens. Consequently, after the initial peak, their gross returns decline smoothly and gradually, indicating that their hedge executions generate limited market impact. Conversely, Pattern 2 searchers, trading primarily lower-liquidity ALT tokens, experience rapid, pronounced closures of spreads immediately after peak returns. The thin liquidity conditions mean their hedge executions will quickly push prices against their positions, collapsing the spread.

To quantitatively validate this link, we examine the correlation between token liquidity and how rapidly and significantly gross return declines. As is illustrated in Figure 5, we find a strong negative correlation between the proportion of major token trade and the percentage decline in median gross returns within 3 seconds after peaking for each searcher. This confirms that searchers predominantly trading major tokens experience slower and smaller gross return declines, reflective of reduced price impact. These findings align with the theoretical predictions in [7, 11].

This rapid and significant spread closing after the peak for Pattern 2 searchers strongly indicates that they prioritize immediate position neutralization despite significant price impact, as they face a trade-off in less liquid markets: extending hedge execution may

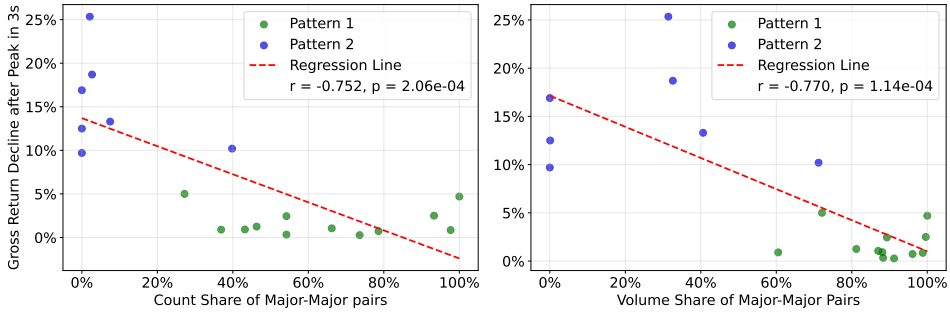


Figure 5 Correlation between the decline in median gross returns within 3 seconds after peak and the proportion of trade count (left panel) and trade volume (right panel) involving Major–Major token pairs for each searcher. Each scatter represents one searcher, with color indicating their pattern. The y-axis label is shared between the two subplots.

increase inventory risk compared to potential hedge cost savings. The observed flat tail of their gross return after the drop further confirms their emphasis on execution speed and inventory risk reduction over slippage minimization. In contrast, the muted, gradual spread compression for Pattern 1 searchers is consistent with deeper liquidity markets: even if hedging immediately with substantial volume, high liquid markets typically absorb size with only limited price impact. These searchers are further enabled to fragment large-size hedge trade into smaller pieces to minimize price impact at the cost of holding the inventory longer.

Finally, we note that Pattern 3 searchers exclusively trade ALT tokens, yet exhibit minimal to no spread closure within the observed interval. This reinforces our earlier interpretation that these searchers likely delay hedging or choose not to hedge within our observation window, as immediate hedging would otherwise generate pronounced declines in their gross returns similar to those of Pattern 2.

Overall, our analysis empirically highlights how token liquidity profiles shape distinct value extraction and hedging regimes for CEX-DEX searchers. While token liquidity appears central to explaining observed patterns, we acknowledge other factors not presented here – such as individual searcher strategies or risk preferences – may also influence these outcomes.

6 Extracted Value from CEX-DEX Arbitrages

Using the methodology outlined in Section 4, we analyze a total of 7,203,560 CEX-DEX arbitrages with a total volume of 241.7B USD executed by 19 labeled searchers between August 8, 2023 and March 8, 2025. These figures significantly surpass atomic MEV activities during the same timeframe. Figure 6 illustrates the cumulative CEX-DEX arbitrage revenue (i.e., extracted value) and weekly distribution of revenue among searchers over these 19 months. According to our estimation, a total of 233.8M USD is extracted by these 19 searchers through CEX-DEX arbitrages. At the time of writing, this figure is comparable to the total value extracted from atomic arbitrages *since The Merge* [4].

Beyond the revenue figures, our result highlights a structural shift towards market centralization. Both the volume of CEX-DEX arbitrages and the associated extracted value have become increasingly concentrated among a smaller subset of leading searchers. To quantify this trend, we compute the Herfindahl-Hirschman Index (HHI) for both trade volume and extracted value, revealing consistently rising concentration and a highly centralized market by the end of the observed period.

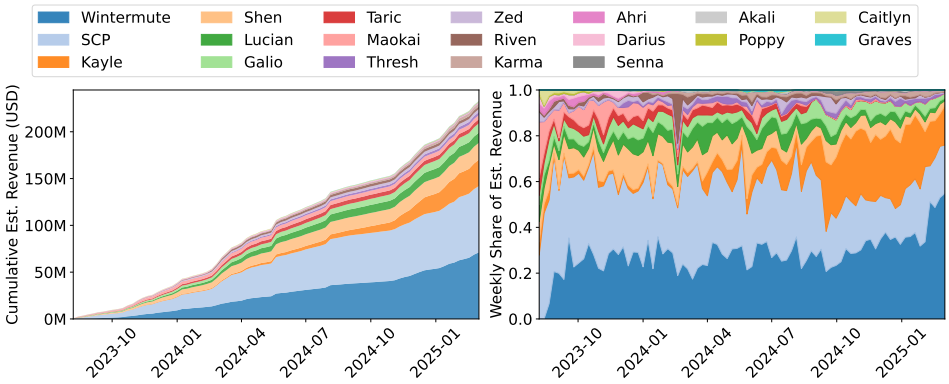


Figure 6 Cumulative CEX-DEX arbitrage revenue (left panel) and daily share of revenue (right panel) by 19 labeled searchers. A total of 233.8M USD is extracted from 7,203,560 CEX-DEX arbitrages by these 19 searchers between August 8, 2023 and March 8, 2025.

The two largest searchers, `Wintermute` and `SCP` consistently extracted over 50% of total revenue throughout the entire observation period. A notable turning point occurred around June 2024, as the previously smaller searcher `Kayle` rapidly expanded its market presence and altered the competitive landscape. By October 2024, we observe a sharp reduction in searcher participation, with only 12 out of 19 labeled searchers remaining active. The market thus resembles a clear trident, with three searchers – `Wintermute`, `SCP`, and `Kayle` – collectively capturing 170.8M USD, representing approximately 73% of the total cumulative extracted value. Furthermore, these top three players dominate arbitrage opportunities even more decisively, jointly accounting for around 90% of the extracted value by Q1 2025. In contrast, smaller searchers' volume and revenue share have been compressed over time, and some searchers eventually stopped their operations and retreated from the market.

7 Searcher-Builder Integration and Profitability

Searchers' net profitability ultimately depends on the fraction of revenue shared with the block builder. In this section, we examine searchers' integration level with block builders, and analyze its impact on searchers' PnL and profit margin.

7.1 Searcher-Builder Integration

We start by assessing the level of integration between each of the 19 labeled searchers and major block builders by measuring the proportion of CEX-DEX arbitrage volume that each searcher directs to specific builders. Figure 7 visualizes these relationships through color intensity – darker cells indicate higher volume proportions and thus stronger integration between two entities. It is well-established that searcher `SCP` and builder `beaverbuild` are vertically integrated, as do `Wintermute` and `rsync` [34, 43, 58].

Our analysis further reveals exclusivity patterns between specific searcher-builder pairs:

- `Thresh` and `Senna` direct 60% and 89% of their volume to `beaverbuild`, respectively.
- `Kayle` and `Graves` send 52% and 100% of their volume to `Titan`, respectively.

We refer to these searchers, who send more than 50% of their volume to one builder, as the *exclusive searcher* (including *integrated searchers* `SCP` and `Wintermute`) for that builder, and correspondingly label their primary recipient builder as their *exclusive builder*. All other searchers, who spread their trades evenly among multiple builders, we classify as

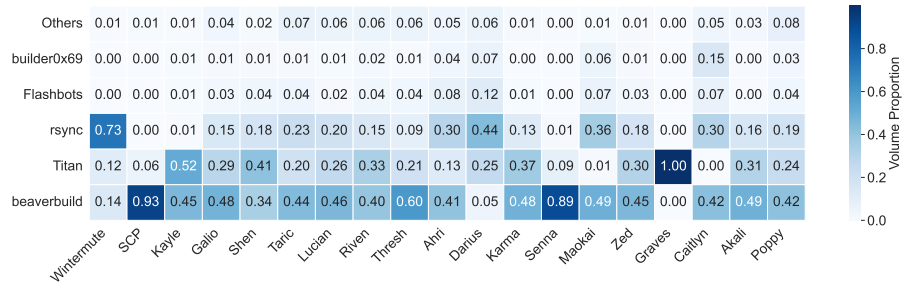
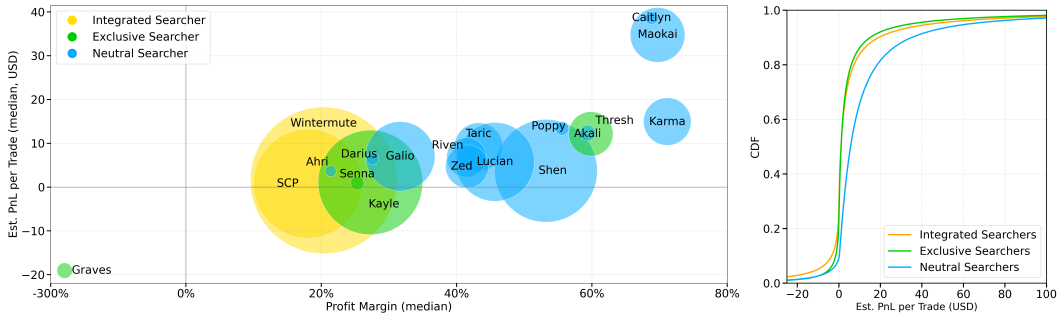


Figure 7 The proportion of CEX-DEX arbitrage volume from each labeled searcher (x-axis) in the blocks of top 5 builders (y-axis).



(a) **(b)**

Figure 8 a) Bubble plot illustrating searchers' median profit margin (x-axis) and median estimated PnL per trade (y-axis). The color shows the searcher's integration level. The bubble size represents the searcher's total PnL. Due to `Graves` having a negative total PnL, we show its absolute value. The negative tick of the x-axis (-300%) is not proportionally spaced. **b)** Cumulative Distribution Function (CDF) of estimated PnL across all trades between the three integration levels.

neutral searchers. Notably, although `Kayle` primarily splits trades between `Titan` (52%) and `beaverbuild` (45%), subsequent analysis will demonstrate a higher integration level with `Titan` than with `beaverbuild`.

7.2 Searcher Profitability Analysis

We then proceed to analyze the profitability of the searchers. In Figure 8a, we find that neutral searchers generally have much higher profitability than most exclusive searchers: their median profit margins cluster around 30% - 70%, whereas those of most exclusive searchers are within 20% - 40%. Neutral searchers' median PnL per trade often exceeds 10 USD, while those of most exclusive searchers rarely exceeds 5 USD. Figure 8b confirms the edge: 80% of exclusive and integrated searchers' trades net fewer than 10 USD, whereas nearly half of neutral searchers' trades exceed this threshold.

Table 1 summarizes the profitability status of the searchers. Despite being market leaders, `Wintermute`, `SCP`, and `Kayle` clear the largest notional volume and earn the highest total revenues, yet their profit margin is among the lowest and their median PnL per trade mostly hovers only a few dollars above zero. `SCP`'s total PnL is barely larger than that of neutral searcher `Shen`, whose trade volume is much lower. Moreover, `Graves`, who sends almost all their trades to `Titan`, exhibits negative profitability according to our estimation, indicating payments to the builder often exceed their arbitrage revenue. We report the medians rather

Table 1 Searcher profitability summary sorted by total volume.

Searcher	Total Volume [USD]	Total Est. Rev. [USD]	Total Builder Tips [USD]	Total Est. PnL [USD]	Median Trade Rev. [USD]	Median Trade PnL [USD]	Median Profit Margin
Wintermute	74.8B	71.4M	47.1M	24.3M	11.36	1.65	20.9%
SCP	63.5B	71.1M	57.4M	13.7M	7.34	0.86	18.5%
Kayle	41.0B	28.3M	16.0M	12.3M	6.05	1.15	27.9%
Galio	25.9B	9.9M	4.5M	5.4M	24.93	7.10	31.7%
Shen	12.3B	18.1M	6.3M	11.8M	8.42	3.77	53.2%
Taric	5.0B	4.4M	1.8M	2.6M	22.92	9.28	43.3%
Lucian	3.9B	10.9M	3.9M	7.0M	14.74	5.83	45.8%
Riven	3.1B	3.1M	1.3M	1.9M	31.42	8.16	42.5%
Thresh	2.7B	3.2M	1.0M	2.2M	34.12	13.65	59.7%
Ahri	1.7B	541.4K	399.0K	162.4K	24.08	4.30	22.8%
Darius	1.1B	497.0K	331.3K	165.6K	29.62	6.62	27.9%
Karma	674.8M	3.0M	517.5K	2.5M	22.82	15.02	71.2%
Senna	596.3M	503.9K	268.1K	235.8K	5.22	1.06	27.2%
Maokai	588.4M	4.4M	1.1M	3.3M	52.40	35.13	69.7%
Zed	580.2M	3.1M	1.1M	2.1M	13.99	4.83	41.8%
Graves	229.3M	173.4K	353.1K	-179.7K	8.77	-19.34	-272.1%
Caitlyn	96.1M	253.7K	67.6K	186.1K	69.02	39.62	68.8%
Akali	49.4M	378.6K	100.5K	278.1K	22.20	12.37	59.4%
Poppy	35.5M	371.7K	171.0K	200.7K	25.00	13.43	55.6%

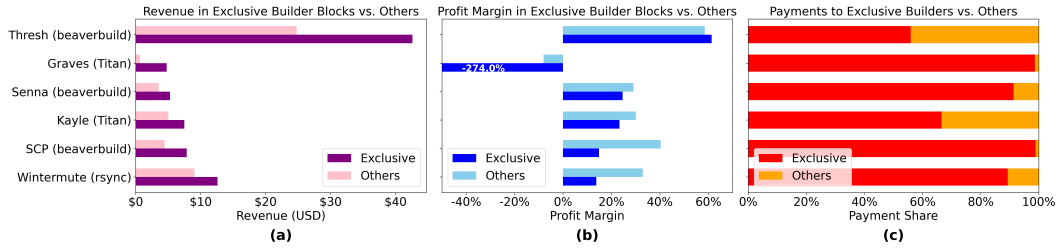


Figure 9 a) Revenue (median) of exclusive searchers' trades in their respective exclusive builders' blocks vs. in other blocks. **b)** Profit margin (median) of exclusive searchers' trades in their respective exclusive builders' blocks vs. in other blocks. **c)** Percentage of total payments from exclusive searchers to their exclusive builder vs. to other builders. The y-axis is shared between three plots and represents the exclusive pair of searcher and builder.

than the cumulative means for metrics of revenue per trade, PnL per trade, and profit margin to avoid the disproportionate influence of outlier trades that skew the means; the medians therefore represent the typical trade more faithfully.

Figure 9 offers deeper insights into these dynamics between exclusive searchers and their corresponding exclusive builders. In Figure 9a, we observe that exclusive searchers consistently send higher-revenue trades to their exclusive builder, compared to other builders. In Figure 9b, for most exclusive searchers, their trades in their respective exclusive builder's block consistently have lower median profit margins, indicating a higher proportion of revenue shared with the exclusive builder than other builders. For instance, both `Wintermute` and `SCP` transfer nearly 90% of their arbitrage revenue to their integrated builder `rsync` and `beaverbuild`, retaining lower profit margins slightly above 10%. While most `Grave`'s trades are not profitable, their trades in their exclusive builder `Titan`'s blocks have a median profit margin of -274.0%. In Figure 9c, as a result of submitting higher revenue trades and sharing greater revenue proportion, exclusive searchers favor their exclusive builder with substantially higher total payments than the amount paid to other builders.

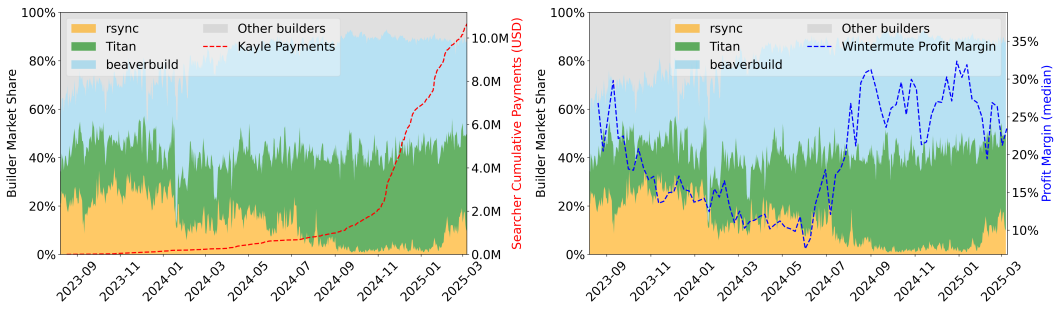


Figure 10 **a)** `Titan` market share and `Kayle`'s cumulative payments to `Titan`. The left y-axis shows the builder market share, and the right y-axis shows `Kayle`' cumulative payments to `Titan`. **b)** `rsync` market share and `Wintermute` profit margin. The left y-axis shows the builder market share, and the right y-axis shows `Wintermute`'s median profit margin.

Both `Wintermute` and `SCP` transfer nearly 90% of their arbitrage revenue directly to their integrated builder to gain an edge in the block building auction and seek a prioritized execution for their trades. `Graves` sends their trade exclusively to `Titan` with payments much higher than their revenue. While `Kayle` splits its trade volume almost evenly between `beaverbuild` and `Titan`, the above economics point to a much higher integration with `Titan`: trades landed in `Titan` blocks earn noticeably higher revenue but carry a markedly lower profit margin, and `Kayle`'s total payments to Titan are roughly double those paid to the rest builders combined. Similar patterns are also evident for `Senna` with `beaverbuild`.

`Thresh`, by contrast, appears only loosely affiliated with `beaverbuild` despite sending 60% of its volume there. The median margin `Thresh` keeps in `beaverbuild` blocks still exceeds 50% – in fact slightly higher than in other blocks – and its total payments to `beaverbuild` are only modestly larger than to builders, which is reasonable considering the success of `beaverbuild` as a builder [55]. This indicates that `Thresh` simply routes most of their trades to `beaverbuild` but is less likely to engage in an exclusive arrangement.

In summary, neutral searchers distribute trades across several builders and pay only the lowest necessary tips for block inclusion, thereby retaining the largest possible share of revenue. Exclusive searchers – likely bound by an exclusive orderflow deal with the builder – often direct their profitable trades to a single builder, willingly sharing a substantially larger fraction of their revenue and accepting slim or even negative PnL at the trade level.

7.3 Impact of Exclusive CEX-DEX Flow on Market Structures

Prior research has highlighted the strategic advantage builders gain by accessing high-value exclusive CEX-DEX flow [32, 34, 43, 57, 58]. Here, we extend this discussion by closely examining interactions between exclusive searcher-builder pairs and their impact on both the searcher and builder markets.

As previously described, `Kayle`, an exclusive searcher affiliated with builder `Titan`, quickly ascended to top-tier searcher status with the third-largest volume since June 2024. Simultaneously, as illustrated in Figure 10a, `Titan`'s market share grew by 15% till September 2024 with a substantial surge in tips from `Kayle`. To quantitatively assess this relationship, we conduct a correlation analysis between `Kayle`'s daily CEX-DEX volume share and `Titan`'s daily market share between June 1 to September 30, 2024.

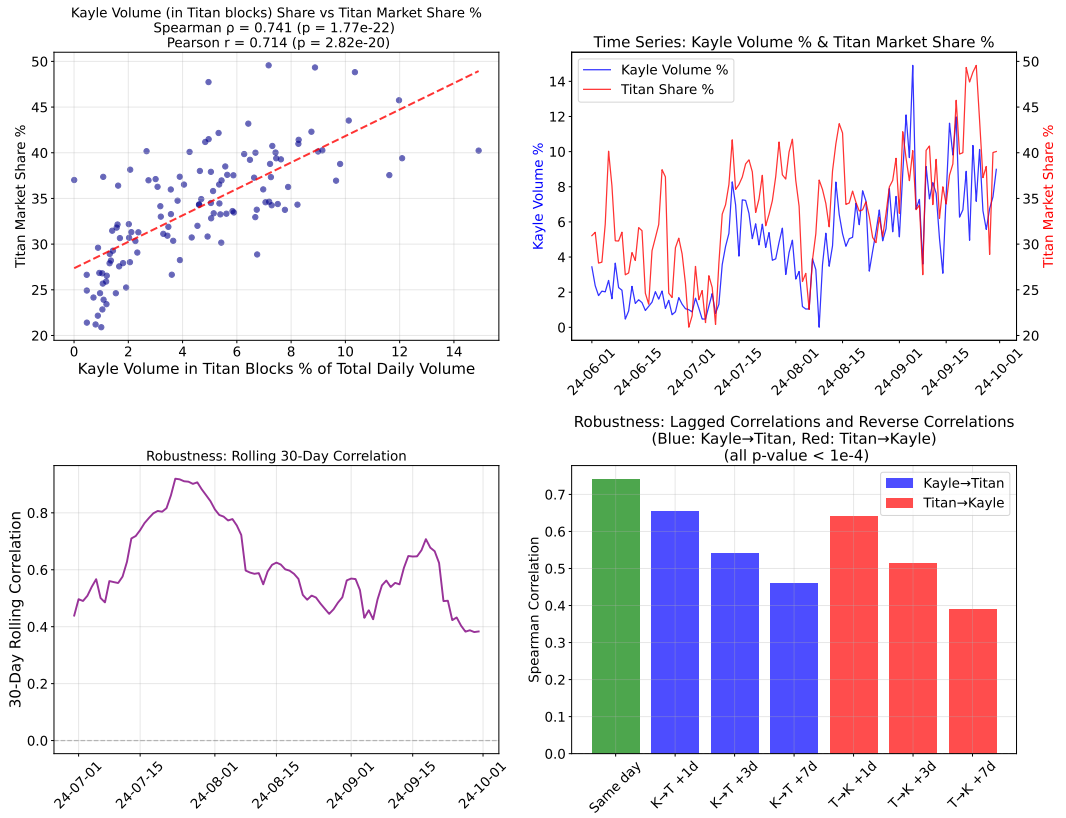


Figure 11 Correlation analysis of `Kayle`'s daily volume share in `Titan`'s blocks (as a percentage of total daily CEX-DEX volume) and `Titan`'s daily market share. Using percentage metrics effectively controls for overall fluctuations in daily total CEX-DEX volume.

We find a strong and robust mutual correlation between `Kayle`'s daily volume share in `Titan's` blocks and `Titan`'s daily market share during this period. Specifically, Figure 11 (top-left panel) shows a significant contemporaneous correlation (Spearman's $\rho = 0.74$, $p < 0.0001$), indicating that higher proportion of `Kayle`'s trade volume included in `Titan`'s blocks strongly coincide with higher share for `Titan` on the same day. The aligned trajectories shown in the time series comparison (top-right panel) further confirm their synchronized behavior. Robustness checks using rolling 30-day correlations confirm the consistency and significance of this relationship across the period analyzed (bottom-left panel).

We further investigate the lagged and reverse correlations at intervals of 1, 3, and 7 days (bottom-right panel). The results indicate a notable predictive component: higher daily volume share from `Kayle` to `Titan` positively correlates with increased market share for `Titan` in subsequent days. This effect is most pronounced at a 1-day lag (Spearman's $\rho = 0.655, p < 0.0001$), gradually decreasing over one week. One plausible explanation is that larger `Kayle` flow in one day brings larger tip revenue for Titan and enhances `Titan`'s bidding power in subsequent days. Moreover, we identify a reverse effect of similar magnitude, suggesting a strong feedback loop: higher market share for `Titan` subsequently increases `Kayle`'s market capture and flow directed back to `Titan`. Indeed, we observe a substantial surge in CEX-DEX volume share included in `Titan`'s blocks since June 2024.

This dynamic played a major role in accelerating centralization within the searcher market during this period and beyond. As shown in Figure 12, CEX-DEX volume became increasingly concentrated. This shift also had significant repercussions for competing builders

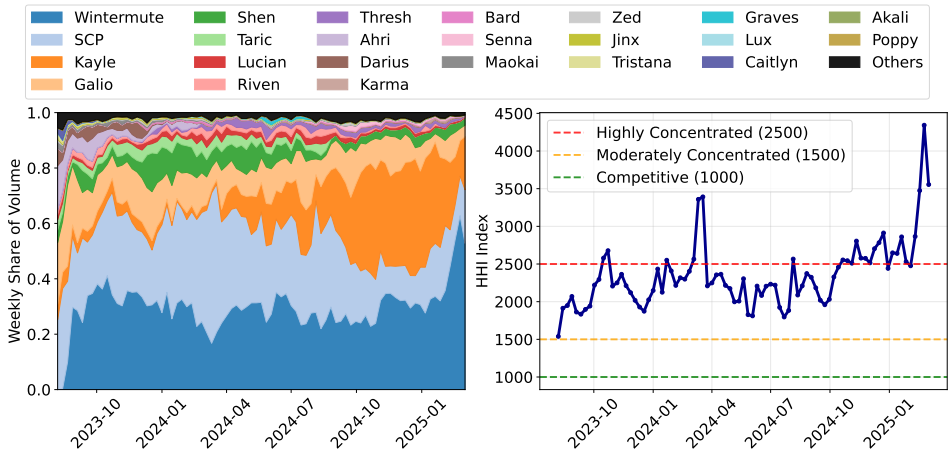


Figure 12 Weekly share of CEX-DEX volume by all the searchers, including Pattern 3 and other searchers, and HHI index of weekly CEX-DEX volume.

– particularly `rsync`. As a result of `Titan`'s expansion, in Figure 10b, we observe `rsync`'s market share fell below 5% by September 2024, effectively retreating from competition in the builder market. At the same time, `Wintermute`'s median profit margin quickly rebounded, indicating a strategic shift away from aggressively supporting `rsync` towards maximizing searcher profitability.

Interpretation and Implications. Combined with the analysis in Section 7.2, these findings provide strong quantitative evidence supporting a mutually reinforcing exclusivity partnership between `Kayle` and `Titan`. Indeed, [15] analyzed both winning and non-winning block contents submitted to the Agnostic Relay and confirmed that `Kayle`'s trades were exclusively seen in `Titan`'s blocks.

Searchers with exclusive builder partnerships, despite paying higher per-trade costs and retaining lower margins, likely gain prioritized inclusion, potentially capturing a greater share of arbitrage opportunities. Builders, in turn, leverage exclusive high-value flow to strengthen their competitiveness in block auctions, further empowering their searcher partners. While explicit off-chain rebates and agreements remain opaque, the observed patterns – higher payments, lower searcher margins, and increased market share – strongly indicate such arrangements. Collectively, these results underscore the significant centralizing pressure exerted by exclusive partnerships and vertical integration within the Ethereum searcher and builder markets.

8 Integrated Builder Profits Correction

Section 7.2 shows `SCP` and `Wintermute` only retain 10-15% of their arbitrage revenue and transfer nearly 90% to their integrated builders. However, `beaverbuild` and `rsync` are unlikely to pass this entire amount as their bid to the proposer; part of the searcher payments may be pocketed as builder surplus upon winning the block. In this section, we combine builders' on-chain profits with searcher-level PnL estimates to correct the oversight by prior research, where only builders' on-chain profit was mesaured [43, 58]. This reconciliation provides deeper insight into the profitability of integrated searcher-builder entities.

Table 2 Builder profitability summary.

Builder-Searcher	Total Blocks [#]	Total Bid Value [#]	Total Builder Profit [USD]	Builder Margin	Total Searcher PnL [USD]	Aggregated Profit [USD]	Agg. Profit Margin
beaverbuild - SCP	1,603,780	402.87M	66.68M	6.69%	11.05M	77.73M	7.92%
rsync - Wintermute	543,564	156.78M	9.65M	-2.24%	14.77M	24.42M	27.06%
Titan	1,206,806	266.80M	92.44M	5.85%	N/A	92.44M	5.85%

We collect the Ultra Sound bid adjustments data [46] and MEV-Boost payloads data [28], and analyze builder profitability from block 17866488 to block 21998438 spanning the period from August 8, 2023, to March 8, 2025. Table 2 summarizes the profitability of `beaverbuild`, `rsync`, and `Titan`, considering integrated searcher profits (if any). "Builder Margin" presents the cumulative mean of profit margin if only builder profit for each block is considered, while "Aggregated Profit Margin" presents the cumulative mean of profit margin if considering both builder profit and integrated searcher PnL for each block. Recall, given that the searcher PnL is an upper-bound estimate, the aggregated profit and margin are also upper-bound estimates.

Results show that `beaverbuild`'s builder profit is roughly 6 times their searcher profit, and the profit margin only increases by about 1% by incorporating their searcher profit. This confirms that while `SCP` transfers nearly 90% of their revenue to their builder, a sizable share never reaches the proposer and is instead retained by their builder.

Since `rsync` is less competitive as a builder, their builder profit is lower even with a slightly negative builder margin. However, if incorporating their searcher profit, we find higher profit retained at their searcher level than `SCP`, and the blocks they win are exceptionally lucrative with a 27.06% profit margin. This indicates that they compete selectively for the most profitable blocks and mainly profit as a searcher instead of as a builder.

Finally, `Titan`, despite not being vertically integrated with any searcher, earns the highest total profit. This is strong evidence that they have exclusive deals with high-value orderflow providers (e.g., searcher `Kayle`), and rebate part of their builder profit back to them.

We further refine builder subsidization measure in MEV-Boost auctions by including integrated searcher profits for `beaverbuild` and `rsync`. Prior studies measured subsidies with builders' on-chain profits only, suggesting builders subsidize blocks to maintain market presence, potentially offset by exclusive orderflow provider profits [43, 58, 52]. We define a block as "subsidized" only if both the builder's on-chain profit and aggregated profit are negative. If the builder profit is positive, we do not classify the block as subsidized, even if the aggregated profit is negative. As is shown in Table 3, after accounting for integrated searcher profits, the apparent subsidy burden for both builders shrinks by roughly 15% fewer blocks and slightly lower outflows. This suggests that the majority of block subsidizing behavior by `beaverbuild` and `rsync` is not directly driven by immediate coverage from their searcher profits in that block.

9 Discussion

Limitations of Our Approach. Although our heuristics expand the coverage of identified CEX-DEX arbitrage trades compared to prior methodologies [43, 34], certain trades may remain undetected, resulting in potentially underestimating searcher revenues and PnL. Given the resources available, our methodology represents the best feasible solution at present.

Table 3 Builder subsidy summary.

Builder	Subsidized Blocks Before Correction [#]	Subsidized Blocks After Correction [#]	Builder Subsidy Before Correction [USD]	Builder Subsidy After Correction [USD]
beaverbuild	498,388	426,335	-2.31M	-2.13M
rsync	179,925	153,713	-3.89M	-3.77M

Our revenue estimation method – based on empirically derived optimal execution horizons across historical trades for each searcher – introduces approximation errors by assuming uniform hedge execution timing across all trades for each searcher and neglecting the price impact of hedging, particularly relevant for low-liquidity tokens. Despite these limitations, our approach is grounded in microstructure literature and captures dominant execution patterns for each searcher, offering consistent upper-bound estimates that allow meaningful comparisons among searchers.

Finally, we do not account for potential off-chain rebates or Orderflow Auction refunds, which might underestimate actual searcher PnL and overstate builder profits, especially for exclusive searcher-builder partnerships.

Centralization in the searcher market. Previous studies have discussed the impact of vertical integration, exclusive orderflow, and builder market centralization, highlighting concerns such as censorship and proposer loss [43, 58, 34]. Our analysis pushes the centralization narrative one step upstream. Critically, we are the first to provide robust quantitative estimates of the value extracted through CEX-DEX arbitrages and shed empirical light on profit-sharing dynamics previously understood only qualitatively.

It appears that, just as arbitrages in traditional finance have been concentrated among a few specialist firms [48] – often because they purchase retail order flow (i.e., payment for order flow) or control clearing licenses [22, 36] – arbitrages at the TradFi and DeFi interface are similarly centralized by searcher-builder orderflow arrangements and vertical integration.

The two largest CEX-DEX searchers by volume and revenue – `SCP` and `Wintermute` – are each vertically integrated with leading block builders, `beaverbuild` and `rsync`, respectively. This vertical integration grants them a decisive strategic advantage in securing block inclusion for their arbitrage transactions. Further, we identify that the third-best searcher `Kayle` emerged as a significant market participant beginning around June 2024, likely following an exclusive arrangement with the second largest builder `Titan`. Collectively, these three searchers dominate approximately three-quarters of the total arbitrage volume and value extracted, reinforcing further centralization in both the builder and searcher markets.

In contrast, smaller searchers, despite maintaining higher margins per trade, have seen their overall volume and revenue share diminish over time. Given the already high entry barriers to CEX-DEX arbitrage, this further centralization significantly elevates barriers to entry, discouraging market participation by smaller entities and fostering economies of scale.

Recent research also highlights centralization concerns in DEX solver markets, where notably, `Wintermute` and `SCP` similarly dominate the solver markets on prominent platforms like CoWSwap, Uniswap X, and 1inchFusion [59]. Taken together, these observations suggest that the entire MEV supply chain – from solver to searcher to builder – is becoming increasingly consolidated around a few dominant market participants.

Mitigations and Challenges. One proposed mitigation discussed in prior works [39, 34, 31] involves shorter block times (e.g., EIP-7782 [6]) to reduce price movements between blocks and limit overall arbitrage opportunities. While this lowers the total value extractable from CEX-DEX arbitrages, it is less clear that shorter block times would meaningfully shift the market structure, given the sustained advantage held by dominant, integrated players.

Mechanisms designed to promote decentralization in the downstream builder market, such as Orderflow Auctions [13, 29] and BuilderNet [23], could enhance decentralization at the builder level. However, their impact on the upstream searcher market remains uncertain and warrants further study.

Lastly, mechanisms aimed at capturing, redistributing, or burning MEV at the application [60, 38] or protocol level [20, 21, 14] could reduce centralization pressures. Nevertheless, these approaches rely on accurate MEV oracles [58], which remain an open technical challenge.

10 Conclusion

In this work, we shed light on value extraction, profitability, and market dynamics associated with CEX-DEX arbitrage on Ethereum. Our findings highlight increasing centralization as three major searchers affiliated with top builders dominate CEX-DEX arbitrage opportunities, compressing smaller participants despite their higher per-trade margins. Exclusive searcher-builder arrangements further amplify these centralization pressures both downstream and upstream of the MEV supply chain, underscoring critical economic and strategic considerations for Ethereum's decentralization guarantees.

References

1 Coinmarketcap. URL: `https://coinmarketcap.com/`.
2 Dune analytics. URL: `https://dune.com/home`.
3 Etherscan.io. URL: `https://etherscan.io/`.
4 libmev. URL: `https://libmev.com/`.
5 Zeromev. URL: `https://zeromev.org/`.
6 Ben Adams and Dankrad Feist. Eip-7782: Reduce block latency. URL: `https://eips.ethereum.org/EIPS/eip-7782`.
7 Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3(2):5–39, 2000.
8 Data Always. Mev bid data explorer. URL: `https://dune.com/data_always/mev-bid-data-explorer?block_number_t72e94=22196045`.
9 Dune Analytics. dex.trades table on ethereum. URL: `https://docs.dune.com/data-catalog/evm/ethereum/curated-data/dex/dex-trades`.
10 Michael J. Barclay and Jerold B. Warner. Stealth trading and volatility: Which trades move prices? *Journal of Financial Economics*, 34(3):281–305, 1993.
11 Dimitris Bertsimas and Andrew W Lo. Optimal control of execution costs. *Journal of financial markets*, 1(1):1–50, 1998.
12 Binance. Spot trading fee rate. URL: `https://www.binance.com/fee/trading`.
13 MEV Blocker. Mev blocker. `https://mevblocker.io/`. accessed: 2025-05-14.
14 Jonah Burian, Davide Crapis, and Fahad Saleh. Mev capture and decentralization in execution tickets, 2024. `doi:10.48550/arXiv.2408.11255`.
15 Andrea Canidio and Vabuk Pahari. Becoming immutable: How ethereum is made, 2025. `doi:10.48550/arXiv.2506.04940`.
16 Colin Chan, Thomas Thiery, Julian Ma, and Barnabé Monnot. Empirical analysis of cross domain cex-dex arbitrage on ethereum. URL: `https://ethresear.ch/t/empirical-analysis-of-cross-domain-cex-dex-arbitrage-on-ethereum/17620`.

17 Eugene Chen, Alex Toberoff, Suraj Srinivasan, and Ankit Chiplunkar. A tale of two arbitrages. URL: https://frontier.tech/a-tale-of-two-arbitrages.

18 Ankit Chiplunkar and Stephane Gosselin. A new game in town. URL: http://frontier.tech/a-new-game-in-town.

19 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv preprint arXiv:1904.05234*, 2019. arXiv:1904.05234.

20 Francesco D'Amato. Committee-driven mev smoothing. URL: https://ethresear.ch/t/committee-driven-mev-smoothing/10408.

21 Justin Drake. Mev burn — a simple design. URL: https://ethresear.ch/t/mev-burn-a-simple-design/15590.

22 Darrell Duffie and Haoxiang Zhu. Does a central clearing counterparty reduce counterparty risk? *The Review of Asset Pricing Studies*, 1(1):74–95, 2011.

23 Flashbots. Buildernet. URL: http://buildernet.org/docs.

24 Flashbots. Dex trading bot table. URL: https://dune.com/queries/3375587.

25 Flashbots. Illuminating the order flow landscape. https://writings.flashbots.net/illuminate-the-order-flow. accessed: 2025-05-23.

26 Flashbots. Mempool dumpster. accessed: 2025-05-08. URL: https://github.com/flashbots/mempool-dumpster.

27 Flashbots. Mev-boost. https://docs.flashbots.net/flashbots-mev-boost/. accessed: 2025-05-20.

28 Flashbots. Mev-boost analytics - relay scan. https://www.relayscan.io/builder-profit?t=7d. accessed: 2025-05-14.

29 Flashbots. Mev-share. https://docs.flashbots.net/flashbots-protect/mev-share. accessed: 2025-05-14.

30 Elijah Fox, Mallesh Pai, and Max Resnick. Censorship resistance in on-chain auctions, 2023. doi:10.48550/arXiv.2301.13321.

31 Robin Fritsch and Andrea Canidio. Measuring arbitrage losses and profitability of amm liquidity, 2024. doi:10.48550/arXiv.2404.05803.

32 Tivas Gupta, Mallesh M Pai, and Max Resnick. The centralizing effects of private order flow on proposer-builder separation. In *5th Conference on Advances in Financial Technologies*, 2023.

33 Lioba Heimbach, Lucianna Kiffer, Christof Ferreira Torres, and Roger Wattenhofer. Ethereum's proposer-builder separation: Promises and realities. In *2023 ACM Internet Measurement Conference (IMC), Montreal, QC, Canada*, October 2023.

34 Lioba Heimbach, Vabuk Pahari, and Eric Schertenleib. Non-Atomic Arbitrage in Decentralized Finance . In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3866–3884, Los Alamitos, CA, USA, May 2024. IEEE Computer Society. doi:10.1109/SP54263.2024.00256.

35 hildobby. Atomic mev table. URL: https://dune.com/queries/3493305.

36 Edwin Hu and Dermot Murphy. Competition for retail order flow and market quality. *Available at SSRN 4070056*, 2022.

37 Robert A. Korajczyk and Dermot Murphy. Performance of algorithmic trading orders: Size, speed, and volume. *Review of Financial Studies*, 32(7):2829–2873, 2019.

38 Sorella Labs. Angstrom. URL: https://x.com/angstromxyz/status/1889725902851854520.

39 Jason Milionis, Ciamac C. Moallemi, and Tim Roughgarden. Automated market making and arbitrage profits in the presence of fees, 2023. arXiv:2305.14604.

40 Jason Milionis, Ciamac C Moallemi, Tim Roughgarden, and Anthony Lee Zhang. Automated market making and loss-versus-rebalancing. *arXiv preprint arXiv:2208.06046*, 2022.

41 Alex Nezlobin and Martin Tassy. Loss-versus-rebalancing under deterministic and generalized block-times, 2025. arXiv:2505.05113.

42 Alexandre Obadia, Alejo Salles, Lakshman Sankar, Tarun Chitra, Vaibhav Chellani, and Philip Daian. Unity is strength: A formalization of cross-domain maximal extractable value, 2021. arXiv:2112.01472.

43 Burak Öz, Danning Sui, Thomas Thiery, and Florian Matthes. Who wins ethereum block building auctions and why? In *6th Conference on Advances in Financial Technologies*, 2024.

44 Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. An empirical study of defi liquidations: Incentives, risks, and instabilities. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 336–350, 2021. doi:10.1145/3487552.3487811.

45 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 198–214. IEEE, 2022. doi:10.1109/SP46214.2022.9833734.

46 Ultra Sound Relay. Ultra sound relay bid adjustment. URL: https://github.com/ultrasoundmoney/docs/blob/main/bid_adjustment.md.

47 Christoph Rosenmayr and Mateusz Dominiak. Statistical arbitrage on amms and block building on ethereum, 2024. URL: https://greenfield.xyz/2024/09/10/statistical-arbitrage-on-amms-and-block-building-on-ethereum-part-1/.

48 Andrei Shleifer and Robert W. Vishny. The limits of arbitrage. *The Journal of Finance*, 52(1):35–55, 1997.

49 Danning Sui and Fei Wu. Cex-dex transactions 202308-202503. URL: https://dune.com/queries/4931834.

50 Tardis.dev. Available token price data on binance. URL: https://api.tardis.dev/v1/exchanges/binance.

51 Tardis.dev. Binance spot historical data. URL: https://docs.tardis.dev/historical-data-details/binance.

52 Titan and Frontier Resarch. Builder dominance and searcher dependence. URL: https://frontier.tech/builder-dominance-and-searcher-dependence.

53 Anton Wahrstätter, Jens Ernstberger, Aviv Yaish, Liyi Zhou, Kaihua Qin, Taro Tsuchiya, Sebastian Steinhorst, Davor Svetinovic, Nicolas Christin, Mikolaj Barczentewicz, et al. Blockchain censorship. In *Proceedings of the ACM Web Conference 2024*, pages 1632–1643, 2024.

54 Anton Wahrstätter, Liyi Zhou, Kaihua Qin, Davor Svetinovic, and Arthur Gervais. Time to bribe: Measuring block construction market. *arXiv preprint arXiv:2305.16468*, 2023.

55 Toni Wahrstätter. mevboost.pics. URL: https://mevboost.pics/.

56 Fei Wu, Thomas Thiery, Stefanos Leonardos, and Carmine Ventre. From competition to centralization: The oligopoly in ethereum block building auctions. *arXiv preprint arXiv:2412.18074*, 2024. doi:10.48550/arXiv.2412.18074.

57 Fei Wu, Thomas Thiery, Stefanos Leonardos, and Carmine Ventre. Strategic bidding wars in on-chain auctions. In *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 503–511, 2024. doi:10.1109/ICBC59979.2024.10634354.

58 Sen Yang, Kartik Nayak, and Fan Zhang. Decentralization of ethereum's builder market. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1512–1530. IEEE, May 2025. doi:10.1109/sp61157.2025.00157.

59 Yuki Yuminaga, Dex Chen, and Danning Sui. Execution welfare across solver-based dexes, 2025. doi:10.48550/arXiv.2503.00738.

60 Mengqian Zhang, Sen Yang, and Fan Zhang. Rediswap: Mev redistribution mechanism for cfmms, 2024. doi:10.48550/arXiv.2410.18434.

61 Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 428–445. IEEE, 2021.

62 Burak Öz, Christof Ferreira Torres, Christoph Schlegel, Bruno Mazorra, Jonas Gebele, Filip Rezabek, and Florian Matthes. Cross-chain arbitrage: The next frontier of mev in decentralized finance, 2025. arXiv:2501.17335.

Unravelling the Probabilistic Forest: Arbitrage in Prediction Markets

Oriol Saguillo
IMDEA Networks, Madrid, Spain

Vahid Ghafouri
Oxford Internet Institute, UK

Lucianna Kiffer
IMDEA Networks, Madrid, Spain

Guillermo Suarez-Tangil
IMDEA Networks, Madrid, Spain

Abstract

Polymarket is a prediction market platform where users can speculate on future events by trading shares tied to specific outcomes, known as *conditions*. Each market on Polymarket is associated with a set of one or more such conditions. To ensure proper market resolution, the condition set must be *exhaustive* – collectively accounting for all possible outcomes – and *mutually exclusive* – only one condition may resolve as true. Thus, the collective prices (probabilities) of all related outcomes (whether in a condition or market) should be \$1, representing a combined probability of 1 of any outcome. Despite this design, Polymarket exhibits cases where dependent assets are mispriced, allowing for purchasing (or selling) a certain outcome for less than (or more than) \$1, guaranteeing profit. This phenomenon, known as arbitrage, could enable sophisticated participants to exploit such inconsistencies.

In this paper, we conduct an empirical arbitrage analysis on Polymarket data to answer three key questions: (Q1) What conditions give rise to arbitrage? (Q2) Does arbitrage actually occur on Polymarket?, and (Q3) Has anyone exploited these opportunities? A major challenge in analyzing arbitrage between related markets lies in the scalability of comparisons across a large number of markets and conditions, with a naive analysis requiring $O(2^{n+m})$ comparisons. To overcome this, we employ a heuristic-driven reduction strategy based on timeliness, topical similarity, and combinatorial relationships, further validated by expert input.

Our study reveals two distinct forms of arbitrage on Polymarket: *Market Rebalancing Arbitrage*, which occurs within a single market or condition (intra-market), and *Combinatorial Arbitrage*, which spans across multiple markets (inter-market). We use on-chain historical order book data to analyze when these types of arbitrage opportunities have existed, and when they have been executed by users. We find a realized estimate of 40 million USD of profit extracted across both types of arbitrage during our measurement period.

2012 ACM Subject Classification Security and privacy → Economics of security and privacy

Keywords and phrases Prediction Markets, Maximal Extractable Value, Large Language Models

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.27

Related Version *Full Version*: `https://arxiv.org/abs/2508.03474` [32]

Funding This work was supported by a Flashbots Research Proposal FRP-51.

Acknowledgements Additionally, we thank the anonymous AFT reviewers for the helpful feedback.

1 Introduction

Forecasting future events has long been a central problem in economics and finance, where accurate predictions can have significant informational and monetary value. Traditional approaches have often relied on expert judgment or econometric models, each with their own limitations in adaptability and responsiveness to new information [33, 5].

© Oriol Saguillo, Vahid Ghafouri, Lucianna Kiffer, and Guillermo Suarez-Tangil;
licensed under Creative Commons License CC-BY 4.0

Prediction markets [39] offer a novel approach to generating forecasts. The general idea is to pose a question openly to the public along with its possible outcomes. Participants place bets when they believe that the probability assigned to an outcome is inaccurate and may earn money over time if their predictions are correct. An example of such a market was introduced by Tradesports.com [13], which listed a security that would pay $100 if the head of DARPA was ousted by the end of August 2003. The novelty of this approach lies in the speed with which the probabilities were updated as new information about the event became available. The future can also be viewed as an incomplete information game, where each individual holds a private valuation of an outcome based on their personal perception of reality. By aggregating these valuations across participants, the system can generate a more accurate probability of an event occurring. Recent prediction markets for the 2024 U.S. election have demonstrated greater accuracy in forecasting outcomes compared to traditional media forecasting tools [9].

Polymarket [27], a prediction market protocol on the Polygon blockchain [17], has emerged as the leading partially-decentralized protocol for making prediction markets a reality, largely due to its high levels of user engagement during the US elections. For instance, during the 2024 elections, Polymarket saw explosive growth, with over $3.7 billion in total trading volume and tens of thousands of active users placing bets on election outcomes [23]. (See Figure 3 for an overview of data on liquidity locked in the markets over time.) At its peak, markets on Polymarket were resolving millions of dollars in open interest, with major news outlets and online communities citing market odds as real-time indicators of public sentiment. This surge in activity is positioning Polymarket as one of the most popular forecasting ecosystems, bridging retail speculation with informational efficiency.

Arbitrage opportunities arise when two identical assets are valued differently due to a pricing mismatch between sellers. Arbitrageurs are sophisticated individuals or agents who exploit these discrepancies by purchasing the asset at the lower price and selling it to a buyer willing to pay a higher price. In traditional finance, arbitrage is a well-studied mechanism for enforcing price consistency and improving market efficiency. In prediction markets like Polymarket, arbitrage plays a similar role, acting as a corrective force that aligns asset prices with their underlying probabilistic truth values. Unlike centralized markets, however, arbitrage on-chain introduces unique complexities: it requires cross-market, fast execution, and the ability to parse fragmented and often noisy information across multiple interrelated conditions. This makes the identification and exploitation of arbitrage on platforms like Polymarket a sophisticated technical challenge, especially at scale.

1.1 Our Contributions

To understand how arbitrageurs may be taking advantage of pricing mismatches, we set out to answer the following research questions:

- Q1. What conditions give rise to arbitrage?
- Q2. Does arbitrage actually occur on Polymarket?
- Q3. Has anyone exploited these opportunities?

To answer these questions, we precisely characterize the types of arbitrage that arise in outcome-dependent condition spaces. We then design a methodology that combines heuristic-driven reduction with semantic analysis to efficiently identify arbitrage opportunities across Polymarket. Our approach leverages temporal proximity, categorization into primary topics using textual embeddings (generated by *Linq-Embed-Mistral*), and large language models (LLMs) to extract combinatorial relationships and logical dependencies from market condition descriptions. This enables us to reduce the arbitrage search space and systematically triage market arbitrage. To perform our analysis, we collect historical bid data from Polymarket.

We use this bid data to characterize arbitrage opportunities within conditions in a single market, and across dependent markets. Our data collection efforts span markets that were resolved over the period of one year, from April 1, 2024, to April 1, 2025.

1.2 Related Work

This work presents the first large-scale analysis of arbitrage on Polymarket, one of the most widely-used prediction markets. A central contribution of our study is the investigation of combinatorial relationships between conditions to infer and characterize dependencies that may give rise to arbitrage opportunities. To contextualize our approach, we survey prior research on Maximal Extractable Value (MEV) detection and highlight recent advances in leveraging LLMs to analyze and interpret complex cognitive and decision-making processes.

1.2.1 MEV Detection

Maximal Extractable Value (MEV) [6] refers to the game-theoretic problem of transaction manipulation within a block, including inserting, reordering, or censoring transactions. MEV has become a regular feature of decentralized systems, where it is used to extract profit, often to the detriment of end users. This phenomenon has been explored across a range of contexts, including in [37], [8], [43], and [21]. In this paper, we focus on a specific class of MEV strategies: arbitrage, particularly as it applies to a new category of decentralized applications, prediction markets. Several mechanisms have been proposed to mitigate the negative effects of MEV, as surveyed in [41]. Among these is the creation of more optimized MEV extraction environments, such as the Proposer-Builder Separation (PBS) paradigm. The goal of PBS is to reduce the influence of validators over transaction ordering and block content. Empirical studies, including [14] and [20], have examined the implementation and impact of PBS in production.

Currently, arbitrage is generally considered a positive-sum form of MEV, as it promotes price alignment across decentralized protocols. Nonetheless, there is ongoing debate in the community about whether arbitrageurs capture a disproportionate share of value from this rebalancing process [19]. A key clarification for this article is our focus on non-atomic arbitrage. Unlike atomic arbitrage – where buy and sell operations are executed simultaneously and failure is impossible – non-atomic arbitrage introduces execution risk, as one leg of the trade may succeed while the other fails. Empirical measurements of non-atomic arbitrage in blockchain systems can be found in [15] and [31].

1.2.2 LLMs

Recent studies have demonstrated the utility of LLMs in various social and semantic annotation tasks, making them cost-effective alternatives to human annotators. Tasks such as *stance detection* [45], *sentiment analysis* [44], *toxicity detection* [44], detecting sociopolitical affiliation of texts [10], etc. have been put to large-scale applications by previous works and have proven successful even with light-weight open-source LLMs such as "*Mistral-7B-Instruct-v0.2*" [45], Llama-3.2-8B-Instruct [40], and DeepSeek [12].

Moreover, previous literature has extensively leveraged LLMs to analyze and understand complex cognitive processes, including the intricate structures of reasoning and the underlying logical dependencies within various tasks, often through techniques such as Chain-of-Thought (CoT) prompting [36, 35]. These methods guide LLMs to break down complex problems into intermediate steps, making the reasoning process more explicit and verifiable [42].

Our LLM annotation task of detecting logical dependencies between human-generated markets falls in a hybrid domain between social annotation tasks and the analysis of logical dependencies. The perfect execution of all the mentioned tasks, including ours, relies not only on the power of the chosen LLM but also on the quality of a well-curated prompt, namely *prompt engineering* (see Section 2.2).

2 Background

We present the background necessary to understand prediction markets and how LLMs can aid in understanding when markets are dependent, a first step needed to assess arbitrage.

2.1 Polymarket

Polymarket is a prediction market platform that allows users to speculate on the outcomes of future events by trading shares tied to specific outcomes. Polymarket is built on top of the Polygon blockchain [17], providing some decentralized properties to the platform. Each **condition** poses a question about a future event, such as "Will team A defeat team B in the big game?". Users can buy shares (or *tokens*) in "YES" (the condition will become true) or "NO" (the condition will not become true) outcomes, with share prices fluctuating based on market demand and reflecting the collective belief about the likelihood of each outcome.

A **market** is thus a future event associated with one or more conditions. For example, consider the question, "Who will win the Team A vs. Team B match?". In this case, the market would include three conditions: (1) Team A wins, (2) the match ends in a tie, or (3) Team B wins. Each condition is represented by a binary token indicating whether the condition is true or false. To ensure proper resolution, the set of conditions must be *exhaustive*, collectively covering all possible outcomes of the event, and *independent*, where only one condition can resolve to true. When a market contains multiple conditions, they share a market ID and are labelled as NegRisk (neg risk market).[1]

2.1.1 Market Creation

The creation of markets [29] and their associated conditions is controlled by Polymarket and must be conducted via their Discord server or by tagging the official Polymarket account on Twitter (@polymarket). To suggest a new market, users provide (i) a title for the market, (ii) the designated resolution source,[2], and (iii) evidence that there is demand for trading in this market. Markets are then registered on-chain, and the corresponding tokens are created for each condition and outcome. Tokens representing a condition follow the ERC-1155 standard under the Gnosis Conditional Tokens Framework [11].

2.1.2 Buying Positions

Positions (tokens) in Polymarket are bought/sold based on a hybrid-decentralized Central Limit Order Book (CLOB) system [28]. Users make a bid to buy/sell some token and submit these directly to the Polymarket API. Polymarket then matches bids, and those matched bids are executed on-chain. The on-chain element ensures that users own the tokens corresponding to their positions in each market, adding an element of decentralization while the matching is done entirely in a centralized manner.

[1] As users can hedge against specific risks by placing bets that the event won't happen.
[2] One or more sources that should be referred to for information on the actual outcome of the event.

Users place limit orders to buy or sell "YES" or "NO" outcome shares at specified prices. The simplest type of match occurs when two users submit a sell and buy bid for a token at the same price. In this case, the token and USDC values are directly traded, and the conditional token contract emits a **OrderFilled** event logging the trade on-chain.[3] Another type of match is made when the prices of opposing orders sum to \$1.00. For example, a "YES" bid at \$0.60 matches with a "NO" bid at \$0.40. Upon matching, \$1.00 is converted into one "YES" and one "NO" share, each allocated to the respective buyer. On-chain, this is done by users locking USDC tokens[4] with the Polymarket exchange contract, while one new conditional token is minted corresponding to one "YES" token and one "NO" token of the condition. The generation of the new token emits a **PositionSplit** event and two **OrderFilled** events for each side of the buys. Similarly, two sells of opposing orders can sum to \$1.00, resulting in the two tokens being burned and each user withdrawing the corresponding USDC from the exchange contract. This action emits a **OrdersMerged** event capturing the tokens being burned, and an **OrderFilled** event for each side of the sell.

Lastly, a user can buy both sides of a condition by locking in 1 USDC and generating both one "YES" and one "NO" token, which they may trade later. This emits a **PositionSplit** event. Similarly, a user may sell both positions for a condition at a cumulative price of 1 USDC. This withdraws the USDC from the exchange and burns the two tokens representing it, emitting a **OrdersMerged** event.

2.1.3 Market Resolution

Polymarket utilizes UMA's Optimistic Oracle to determine market outcomes [34]. When a market concludes, a resolution is proposed. If undisputed during a designated challenge period, it is accepted as final. However, if challenged, the resolution is escalated, here UMA token holders vote to determine the outcome. In the end, each condition resolves to "True" or "False", and a single condition in each market resolves to "True". This resolution is registered on-chain, and users can then claim their corresponding USDC. This system can lead to discrepancies between the oracle's resolution and the actual event, especially in complex scenarios.[5] Additionally, the concentration of voting power among large UMA token holders can influence resolutions, as observed in [30], and the oracle is susceptible to potential governance attacks as discussed in [7].

2.2 LLMs & Prompt Engineering

Prompt engineering refers to the process of crafting input queries (called *prompts*) to guide an LLM toward producing useful or accurate outputs. Since these models do not inherently "understand" in the human sense, the way a question is phrased can significantly impact the quality of the response. Prompt engineering has emerged as a practical technique to control and refine LLM behavior without retraining the model. For instance, adding a few examples to a prompt (called *few-shot prompting*) or specifying desired formats can help steer the model more effectively.

[3] We use events emitted by the conditional token contract to log all buy/sells that take place on Polymarket when looking at on-chain data from contract `0x4D97DCd97eC945f40cF65F87097ACe5EA0476045`.
[4] The token representation of the US Dollar.
[5] For instance, in the 2024 FIDE World Blitz Championship market [22], both Carlsen and Nepomniachtchi were declared winners of different sections, but only one outcome could be selected as "True" due to the framework's constraints.

In this paper, we leverage prompt engineering practices to extract and interpret logical dependencies within the Polymarket markets to detect potential cases of arbitrage. Our prompt engineering involves explaining the task coherently, defining a set of rules for the task, and restricting the output to our specific desired JSON format while providing a sample desired response to the model (see Listing 1).

3 Definitions

We now introduce some notation to define the set of possible outcomes for a given market, and for pairs of markets. With these, we can precisely define what it means for the resolution of markets to be dependent such that there can exist arbitrage opportunities (i.e., when the resolution of some conditions implies the resolution of some other conditions, either within the same market or across markets). We then define two types of arbitrage: 1. **Market Rebalancing Arbitrage**, which occurs within a single market, and 2. **Combinatorial Arbitrage**, which occurs between multiple markets.

3.1 Market Dependence Taxonomy

3.1.1 Single Market

For a given market, which is designed to forecast the outcome of a real-world event, we define the set of all possible *resolutions* of the market as the set of all possible combinations of True/False labels assigned to each condition associated with that market.

▶ **Definition 1** (Single Market Outcome). *Let $M = \{C_1, C_2, \ldots, C_n\}$ be a market modelling an event E, where each C_i is a boolean variable representing a possible outcome of E (i.e., the n conditions of the market) and so will be resolved as either* `True` *or* `False`*. The set of vectors $V = \{v_i\}$ represents all possible unique resolutions of M with each $v_i =< c_1, c_2, ..., c_n >$ where $c_j \in \{0, 1\}$. Market conditions are* ***exhaustive****, such that $|V| = n$ and $\forall v \in V$, the following holds*

$$\sum_{c_i \in v} c_i = 1$$

We define a market as *exhaustive* as it includes all mutually exclusive conditions required to fully determine the outcome of an event. In such a market, exactly one condition must be true at resolution time, ensuring the outcome space is complete. As such, conditions within a market are dependent by definition: if one resolves to true, all others must resolve to false.

Consider the following example: let M_1 denote a prediction market for an election in New York with the following three mutually defined conditions:

1. The Republican candidate wins in New York.
2. The Democratic candidate wins in New York.
3. A third-party candidate wins in New York.

In this market, the set of conditions is *exhaustive*, as it covers all possible outcomes of the election, and *exclusive*, since at most one of the conditions can be true simultaneously.

3.1.2 Multiple Markets

Depending on how markets are created, it is possible to have two markets with a subset of conditions whose resolutions are semantically dependent. A common example in betting markets is to define both the outcome of an event and the margin by which that outcome

occurs across two semantically related markets. To illustrate this, consider two markets: **Market X**, which contains a set of conditions C representing the possible outcomes of an event (e.g., which team wins), and **Market Y**, which includes a different set of conditions C' that expresses the *margin* or *degree* of the outcome (e.g., the score difference).

For example, consider a football match. Market X might include conditions indicating whether Team A or Team B wins, while Market Y could contain more granular conditions, such as specific scorelines or a minimum winning margin for Team A. Suppose Market Y includes a condition stating that "*Team A wins by at least 2 goals*". Then any context which has a resolution of market Y with "*Team A wins by at least 2 goals*" as True must also have a resolution of market X with Team A also winning. Thus, the state space of possible resolutions of the two markets combined is more limited.

▶ **Definition 2** (Market Outcome for Two Markets). *Let M_1 and M_2 be two markets with condition sets $\{C_1, \ldots, C_n\}$ and $\{C'_1, \ldots, C'_m\}$ respectively. We define the set of possible unique resolutions for each market as V_1 and V_2 respectively, and the combined set of possible **joint** unique resolutions as $V_1 \times V_2 = \{v_i\}$ with each $v_i =< c_1, \ldots, c_n, c'_1, \ldots, c'_m >$ representing a possible resolution of both markets with subsets $< c_1, \ldots, c_n > \in V_1$ and $< c'_1, \ldots, c'_m > \in V_2$.*

While each market is exhaustive, i.e., $|V_1| = n$ and $|V_2| = m$, the set of possible joint resolutions have two cases:

(1) *If $|V_1 \times V_2| = n \cdot m$, the two markets are **independent** of each other.*

(2) *If $|V_1 \times V_2| < n \cdot m$, the two markets are **dependent**, and $\exists S \subset M_1$ and $S' \subset M_2$ where S and S' are **dependent subsets** s.t. $\forall v \in V_1 \times V_2$ the following holds:*

$$\sum_{c_i \in S} c_i = \sum_{c'_j \in S'} c'_j$$

In other words, two markets being independent means any resolution of one market leaves all possible resolutions of the other market as possibilities. When two markets are dependent, however, there is one or more conditions in one market where if any of those conditions resolve to true (or all resolve to false), there is one or more conditions in the other market where one of them must resolve to true (or respectively all to false).

In this work, we focus on single market and two market dependencies as an initial study of arbitrage on prediction markets built atop blockchain protocols. One can imagine extending the above definition to a set of n markets $\{M_1, M_2, \ldots, M_n\}$, where $M = M_1 \times M_2 \times \cdots \times M_n$ is the set of possible unique resolutions of the join n markets. If $|M| = \Pi_{i \in [1,n]} |M_i|$, the outcomes of all markets are independent of each other. If $|M| < \Pi_{i \in [1,n]} |M_i|$, there exists some dependency between some subset of the markets. Characterizing these dependencies quickly becomes more complicated and is out of the scope of this work.

3.2 Arbitrage Labels Taxonomy

In this section, we present a taxonomy of the two primary forms of arbitrage that can arise in order-book-based prediction markets.

3.2.1 Market Rebalancing Arbitrage Label

In prediction markets, the price of a "YES" token is interpreted as the market-implied probability of that outcome occurring. For events with multiple mutually exclusive outcomes – each represented as a distinct condition – the sum of the "YES" token prices should, in

theory, equal 1. When this condition is violated (e.g., if the total is less than 1), arbitrage opportunities arise. Traders can take long positions on all outcomes (or short positions if the sum exceeds 1), securing a risk-free profit when the market resolves. We refer to this as **Market Rebalancing Arbitrage**, where arbitrageurs restore market consistency by adjusting positions or submitting orders until probabilities realign with logical constraints.

▶ **Definition 3** (Market Rebalancing Arbitrage)**.** *Let M be a market with a set of conditions $\{C_1, \ldots, C_n\}$ where $\mathbf{val}(Y_i, t)$ is the price of the Yes position for condition C_i at time t. We say that a **Long Market Rebalancing Arbitrage** opportunity exists at time t if:*

$$\sum_i \mathbf{val}(Y_i, t) < 1,$$

*and a **Short Market Rebalancing Arbitrage** opportunity exists at time t if:*

$$\sum_i \mathbf{val}(Y_i, t) > 1.$$

In a Long Market Rebalancing Arbitrage, the total cost of acquiring one unit of each of the "YES" tokens is less than 1. Since one of the conditions must resolve to true (i.e., the final token value will be 1), this position allows for a *guaranteed* profit of $1 - \sum_i \mathbf{val}(Y_i, t)$.

In a Short Market Rebalancing Arbitrage, since the total cost of acquiring one unit of each "YES" token is more than 1, this implies the "NO" tokens are undervalued. One could buy one unit of all "NO" positions, and when the market resolves, the *guaranteed* profit is

$$n - \sum_i \mathbf{val}(N_i, t) = \sum_i \mathbf{val}(Y_i, t) - 1$$

where N_i is the "NO" position, as the sum value of one token of all "NO" resolutions is $n - 1$.

An alternative shorting strategy is for an arbitrageur to buy a unit of both positions for each condition (i.e., create a **Split** for each condition for 1USDC) and then sell the "YES" position right away, immediately taking advantage of the overvaluation and gaining the profit of $\sum_i \mathbf{val}(Y_i, t) - 1$.[6]

With both long and short Market Rebalancing Arbitrage, the profit gained is the difference between the sum probabilities of all the "YES" conditions and 1 (i.e., $|\mathbf{val}(Y_i, t) - 1|$). We note that due to the order book nature of Polymarket, each of the above trades is non-atomic; thus, there is always some risk in attempting the arbitrage.

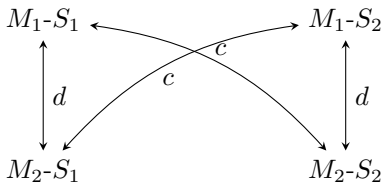
3.2.2 Combinatorial Arbitrage

Given two dependent markets as defined above, we can express the precise market conditions that allow for arbitrage and the resulting arbitrageur strategy. A Combinatorial Arbitrage opportunity arises between two markets when it is possible to construct a portfolio of bets across conditions in both markets such that at least one bet is guaranteed to win.

[6] We can use the same definition to define the much simpler **arbitrage in a single condition** if we think of the "YES" and "NO" positions as effectively two conditions: If they sum to < 1, an arbitrageur buys both positions; If they sum to > 1, an arbitrageur `SPLIT`s the condition and sells both positions.

	S_1	S_2
M_1	Democrat wins	Republican wins
M_2	Democrat margin	Republican margin

(a) Markets M_1 and M_2.



(b) Relationship between the markets.

Figure 1 Markets and state relationships. This figure illustrates two dependent prediction markets: M_1, which declares the winner of a state election, and M_2, which specifies the winning margin. For each market, we define two mutually exclusive and exhaustive states, S_1 and S_2. An assignment of a state in one market can imply the corresponding outcome in the other. In the relationship diagram, edges labelled d indicate that the connected states are *dependent* , and edges labelled c indicate the states are (share opposite token valueations – YES in one corresponds to NO in the other) *complementary* (both must evaluate to TRUE or both to FALSE).

▶ **Definition 4** (Combinatorial Arbitrage). *Let M_1 and M_2 be dependent markets with some dependent subsets $S \subset M_1$ and $S' \subset M_2$. We say that a* ***Combinatorial Arbitrage*** *opportunity exists at time t if either of the following holds:*

(1) *If $\sum_{c\in S} \mathbf{val}(T_c, t) < \sum_{c'\in S'} \mathbf{val}(T_{c'}, t)$, then an arbitrage opportunity exists holding "YES" positions for conditions in S and "YES" positions for conditions in the complement of S'.*

(2) *If $\sum_{c\in S} \mathbf{val}(T_c, t) > \sum_{c'\in S'} \mathbf{val}(T_{c'}, t)$, then an arbitrage opportunity exists holding "YES" positions for conditions in the complement of S and "YES" positions for conditions in S'.*

Note that holding all "YES" positions for the complement of a set $S \subset M$, $\overline{S} = M - S$, is equivalent to holding all "NO" positions for the subset S. Figure 1 shows the dependency graph for an example pair of markets. As with Market Rebalancing Arbitrage, for holding a "YES" position for some condition C, one can either buy a unit of the "YES" token or buy 1 USDC of the condition in a Split and sell the "NO". With Combinatorial Arbitrage, however, we consider arbitrage strategies only in holding the "YES" positions of complementary subsets across two markets (e.g., "YES" of $M_1 - S_1$ and $M_2 - S2$). The profit of the Combinatorial Arbitrage is the difference in the market values of the dependent conditions, i.e., $|\sum_{c\in S_1} \mathbf{val}(T_c, t) - \sum_{c'\in S_1} \mathbf{val}(T_{c'}, t)|$[7].

3.3 Arbitrage Analysis

Given the above definitions, we perform the following arbitrage analysis on Polymarket data (cf. Section 4 for our data collection methodology).

(i) In Section 5, we employ an instance of the LLM *DeepSeek-R1-Distill-Qwen-32B* [4] together with market description data to capture the state space of possible market resolutions to then infer semantic dependency between pairs of markets and their conditions.

[7] We don't consider strategies holding the "NO" position as multiple "NO"s can be true at once (by design), so there is not a symmetric profit between the positions of the two markets.

(ii) In Section 6, we analyze historical orderbook data to detect when Market Rebalancing Arbitrage opportunities existed for each market, and when Combinatorial Arbitrage opportunities have existed for pairs of dependent markets in our measurement window.

(iii) In Section 7, we further examine the orderbook data to determine whether any participants exploited these arbitrage opportunities, and explore the behavior of key actors involved.

4 Data Collection

For this study, we analyzed Polymarket data spanning markets that resolved over the period of one year, from 1st April 2024 to 1st April 2025. Our analysis required both the textual descriptions of each market queried from the Polymarket API [24] and the on-chain historical record of executed bids.

4.1 Market Descriptors

We retrieved market metadata directly from the Polymarket API using the official Python client [24].[8] In this section, we present only the relevant fields needed to understand the market and condition structures, as well as their connections with the underlying smart contracts. In Appendix A in the full version [32], we provide a full overview of all fields.

To detect arbitrage opportunities, it is essential that dependent markets share the same end date and describe the same underlying event in different ways. However, we observed inconsistencies in the `end_date_iso` field among markets with the same market ID, which should not occur, as all associated conditions are expected to resolve simultaneously. To address this issue, we computed the most frequent end date among conditions within the same market ID. In cases where multiple dates shared the highest frequency, we selected the latest date as the canonical `end_date_iso`.

Our data set consists of two types of markets: 8659 single-condition markets, and 1578 multiple-condition markets (i.e., NegRisk markets) made up of 8559 conditions, totalling 17218 conditions across all markets.

4.1.1 Topic Analysis

We group markets into primary topics using the topic categories listed on the Polymarket website [27]: `["Politics", "Economy", "Technology", "Crypto", "Twitter", "Culture", "Sports"]`. We first use *Linq-Embed-Mistral* model [1] to generate vector embeddings for both market questions and topics. The model was the best-performing *open-source* text embedder [2] at the time of conducting the experiments. We then compute the cosine similarity between each question's embedding and all topics' embeddings, and assign the question to the topic with the highest similarity. Figure 2 plots the number of markets per topic per end date for our measurement period. We see that the top categories are Politics and Sports, with the U.S. election falling in our measurement period, which clearly illustrates a rise in Polymarket activity.

To validate the classification performance, we randomly sampled 100 instances from the dataset and manually labeled them which yielded an accuracy of 92%. In several cases, markets framed as "*If politician X says Y*" were classified under *Politics*, which aligns with

[8] A tutorial on accessing this data is available in [38].

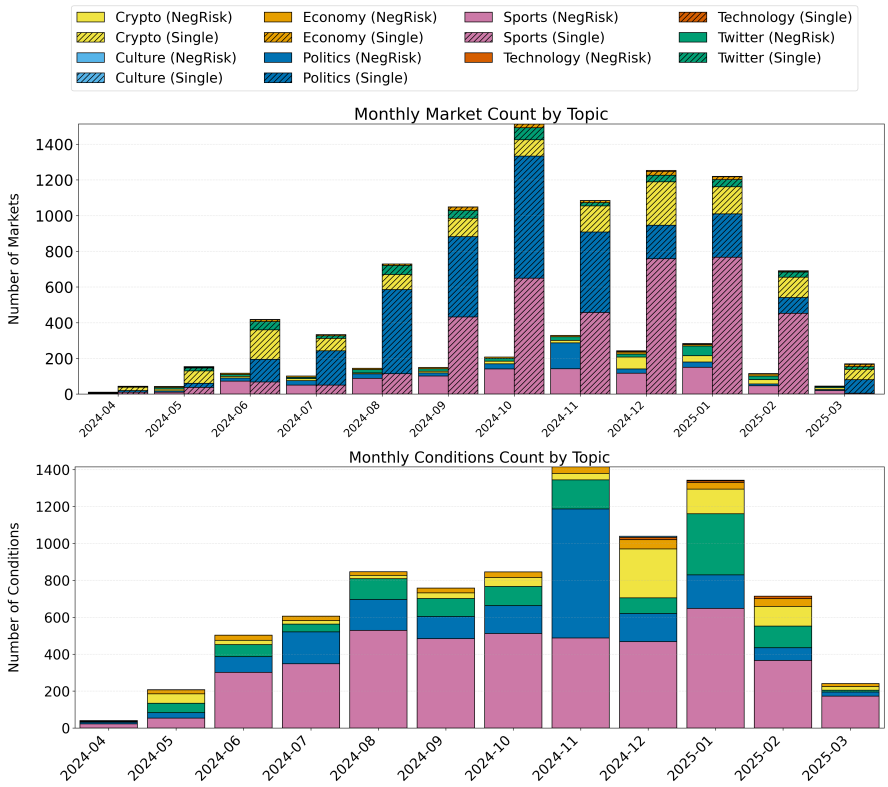


Figure 2 Top shows the total number of markets by topic and end-date, differentiating between the single-condition markets and the neg risk markets. Bottom shows the same value but for total conditions in the neg risk markets. Overall, politics and sports dominate in popularity.

our interpretation of the market context. However, certain markets revealed limitations in the labeling scheme; for example, markets such as "*If the weather will be X*" were sometimes categorized under *Twitter*, despite lacking a clearly appropriate label. Since the primary goal of this labeling was to reduce the search space, we aligned the categories with those used by Polymarket.

In Section 6, when analyzing dependencies between pairs of markets, we consider only pairs of markets within a given topic and end date to limit our search space to markets more likely to relate to the same event (e.g., markets about a match between team A and team B should both resolve on the day of the match when the outcome is known. This is due to the controlled/centralized nature of market creation).

4.2 Historical Bid Data

To get Polygon on-chain data for Polymarket, we use the Alchemy public node API [3]. While users submit bids directly to Polymarket, those bids which are matched are registered on-chain by Polymarket operators. All conditional tokens (a "YES" or "NO" token for each market condition) are an instance of an ERC-1155 Conditional Token, which is managed by the Polymarket Conditional Token Contract at address `0x4D97DCd97eC945f40cF65F87097ACe5EA0476045`. While there are several contracts that make up the logic of Polymarket, all trades and liquidity operations are eventually registered as an `EVENT` by this contract. In particular, we care about the `OrderFilled`, `PositionSplit`, and `PositionsMerge` events.

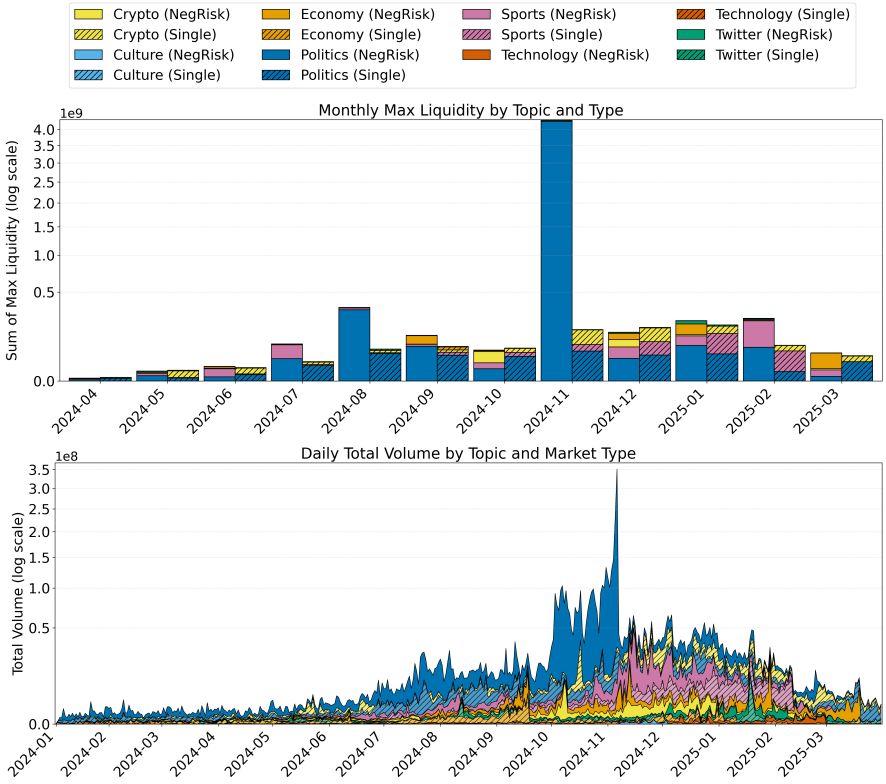


Figure 3 (Top) Total liquidity per market by end date and topic. (Bottom) Total volume of executed bids by market type over time. We see the U.S. election markets were the primary driver of activity in Polymarket at the time.

Section 2 describes the various bids that take place, which can be a simple match of a buy and sell order, or a match of two buys for each side of a condition, such that new tokens are created. Ultimately, the `OrderFilled` event captures any time a token is traded for USDC and who is the buyer/seller (the seller can be the Polymarket exchange contract if new tokens are minted). Whenever new tokens are minted (or conversely destroyed), the `PositionSplit` event registers the USDC being locked (or conversely `PositionsMerge` and withdrawn) and by whom (again, this can be the Polymarket exchange).

We query the transaction traces for the Polymarket Conditional Token Contract for blocks from 1st January 2024 to 1st April 2025 and filter for the three events and only conditional tokens for the markets in our measurement period. Figure 3 shows the total locked volume in markets ending in each month, with the U.S. election surpassing all other markets, and the total volume over time by market cap.

5 Markets Analysis: Detecting Market Dependencies

We begin by developing a methodology that leverages LLMs to automatically detect semantic dependencies between markets as defined in Section 3.1 and represented in Figure 4. While our goal is to determine whether conditions in two separate markets are dependent, we begin with the single market case, where we know all conditions are dependent, to validate the capabilities of the LLM for reasoning.

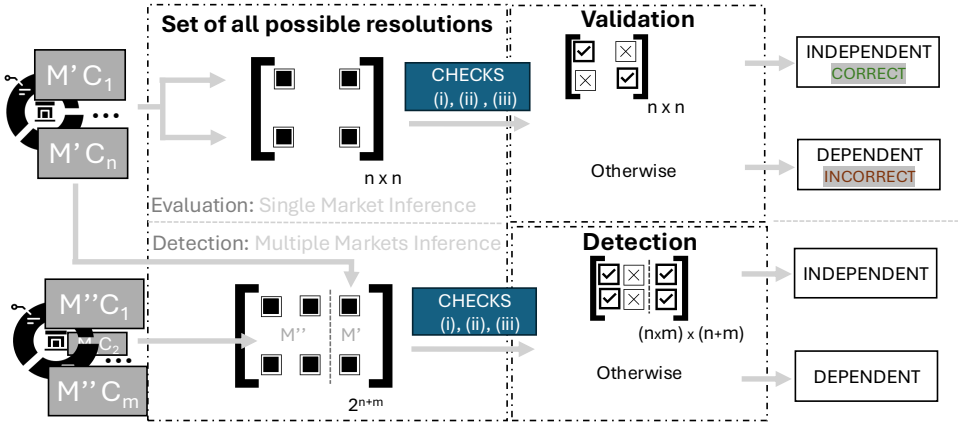


Figure 4 Our approach to detect market dependencies in a nutshell.

5.1 Single Market Inference

Given a collection of conditions, our goal is to output the set of all possible resolutions of the markets. Our approach uses the reasoning capabilities of an LLM to evaluate the logical consistency between a given assertion and a set of statements, using structured inference. We abstract away the prediction market context of this task, and instead focus on *semantic* dependency between statements.

Our first approach iterates exhaustively through the conditions. We take the set of conditions of a market (structured as questions in the question variable), and assert to the LLM that one condition is true and ask it whether the remaining conditions can be true. An example of a market M is defined by the following condition questions: "Will team A win the Turtle Bowl?", "Will team B win the Turtle Bowl?", and "Will team A and team B tie in the Turtle Bowl?". In this setup, the general structure for the LLM prompt is as follows: the `assertion_statement` corresponds to the specific condition being evaluated (e.g., "Team A will win the Turtle Bowl"), while the remaining condition questions are assigned indices to define the mapping between column indices and condition indices.

Again, for conditions in the same market, the LLM should return that the remaining questions must resolve to *False*. While this approach works, it would involve n calls to the LLM for a market with n conditions. We thus compose queries such that the space of all possible outcomes is computed in a single call, and the LLM returns a JSON representation of the output space (see Appendix B in the full version [32] for the full prompt).

We run our prompt for all NegRisk markets on election day 2024-11-05 with assigned_topic as Politics and test the following conditions: (i) The LLM returns a valid JSON, as sometimes the LLM gets stuck in a logical loop. (ii) The returned JSON is correct, meaning there are n unique vectors, and (iii) each vector has exactly one true condition. **We get that out of the 128 markets tested, only 4 do not return a valid JSON, with 101 (81.45%) fulfilling all conditions.**

Looking at the failed prompts, we observe that the LLM cannot handle too many conditions at a time – a known limitation of LLMs failing to handle large prompts well [18, 16]. This would be further problematic when we check the dependency between two markets, as we will concatenate conditions between markets (further increasing the prompt size). We thus pre-process markets with more than 4 conditions and reduce them to 4 or fewer conditions with the most total traded volume (in *YES*, and *NO* tokens), and a fifth condition to catch

all other outcomes. In Appendix C in the full version [32], we show that over 90% of all liquidity in a market resides in the top 4 conditions. Note that the fifth condition is a logical "OR" of all remaining conditions, thus it preserves logical dependencies.

5.2 Multiple Markets Inference

We extend the LLM analysis to pairs of markets. Recall that due to the centralized nature of how markets are created, markets relating to the same event should share an end date. We thus check pairwise dependencies across markets within the same topic group in the same day. Given a pair of markets, we take the union of all conditions (in their reduced format to max 5 conditions), and pass them to the LLM prompt as one set. We perform the following consistency check: (i) the LLM returns a correct JSON, (ii) for each vector of assignments of conditions, there is exactly one true value in each set and (iii) the set of vectors is of size at most $n+m$ for pairs of reduced markets of size n and m. Conditions (ii) and (iii) check that the set of vectors returned by the LLM are a valid assignment for a market (i.e., exactly one condition in a market must resolve to "True").[9]

Non-US Elections. Of the 2267 pairs of markets checked outside the primary U.S. election group ("Politics" group with end date November 5th, 2024), 30 did not return any JSON, and 203 failed the other two checks. Of the remainder, 2033 pairs were classified as independent, and 1 pair in this set was classified as dependent. Table 3 in Appendix D in the full version [32] shows the details of this pair. While the resolution of some conditions impacts the state space of outcomes for others, it does not strictly satisfy our Combinatorial Arbitragedefinition of Section 3.1. Such dependencies are left for future work.

US Elections. Taking the Politics group with end date November 5th, 2024, we have 128 NegRisk and 177 Single markets. We run all 46360 pairs of markets, we get that 353 do not return a JSON, and 4374 returned an output which does not satisfy (ii) or (iii). In the end, we get a total of **40057 independent markets and 1576 markets characterized as dependent**. Of the pairs characterized as dependent, 129 are between two Single markets, 1353 are between a NegRisk and a Single market, and 94 are between two NegRisk markets. We run the vector of assignments through a checker that verifies whether there exist subsets in the two markets that satisfy Definition 3 [10]. We get 4 NegRisk-NegRisk, 94 Single-Sigle, and 276 NegRisk-Sigle pairs. **We manually check these 374 pairs and get 11 NegRisk-NegRisk and 2 NegRisk-Single pairs, which satisfy our Combinatorial Arbitrage definition.** The majority of false positives corresponded to pairs of markets with some weaker notion of dependency (e.g., a market for who wins a certain swing state, and another for who wins the election) or the LLMs conflating U.S. election outcomes (e.g., the popular vs. electoral college votes, Senate vs. House elections, etc.). Eight markets contributed to 1469 invalid pairs, largely due to their own inherent ambiguity, they are listed in the Appendix E in the full version [32].

[9] We see a pattern in the pairs that fail this check, where there is a loop in the LLM's reasoning, resulting in it eventually returning the exhaustive set of "True"/"False" assignments of a vector of size 2^{n+m}.

[10] This check involves computing all possible subsets in a pair, an exponential task, but tractable for 5 conditions

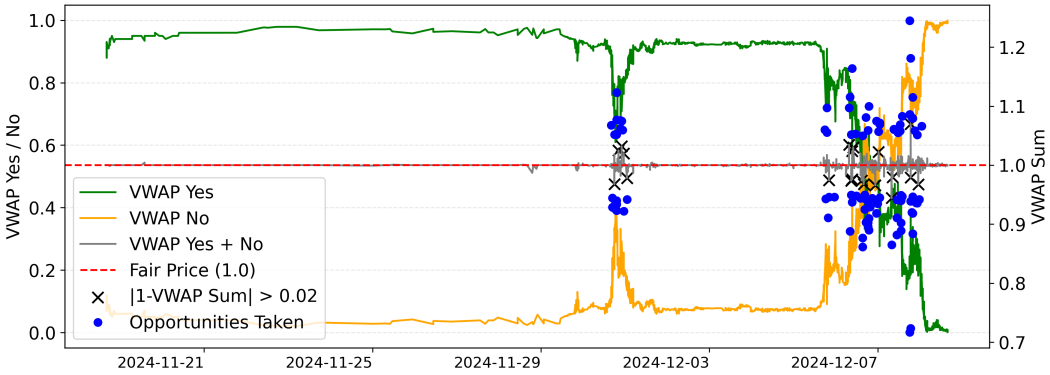


Figure 5 A view of the market behavior of the "Will Assad remain President of Syria through 2024¿' condition. We plot the VWAP price of each position, when we detect an arbitrage opportunity ($|1 - \text{VWAP Sum}| > 0.02$), and the events where an arbitrageur profited from the opportunity (computed in Section 7. Because we calculate token prices from an average of executed bids, we underestimate the margin of profit an arbitrageur is able to realize.

6 Markets Analysis: Detecting Arbitrage Opportunities

We now use the history of executed bids to explore when arbitrage opportunities existed in each market. We take all executed bids for each position (this includes both the USDC amount and the token amount) and calculate a weighted average price for each position, weighted by the token amount. We compute this average over some time frame T. We note that arbitrage happens during periods of volatility, so the larger the time frame, the less volatility we capture; we take the average over one block, carrying forward the last known price for up to 5K blocks (2.5 hours) if a token isn't traded; Otherwise, we set the price to 0 (a token that stops being traded effectively has no value). Additionally, we want to capture arbitrage when the outcomes of markets are not yet known (i.e., there is sufficient uncertainty such that the market is liquid – there is a market to buy/sell the different outcomes), so we only look at times when no position (no token) is worth more than \$0.95 (that is, it has a predicted outcome greater than 95% probability). Lastly, we limit our analysis to opportunities with a profit of at least \$0.05 on the dollar to focus on the higher-reward opportunities given the risk.[11]

We show an example condition in Figure 5 with the price calculations over time for each position and whether an arbitrage opportunity existed ($|1 - \text{VWAP Sum}| > 0.02$). We see how market uncertainty creates opportunity for arbitrage, and show that players actually capitalized on these opportunities (and at a larger profit margin than our averaged estimates).

6.1 Arbitrage Within Single Conditions

We begin by exploring arbitrage within a single condition, the "YES" and "NO" positions of a single outcome. In total, our data set consists of 17.2K conditions (8.56K from NegRisk and 8.66K from Single markets), of these 7,051 conditions have at least one arbitrage opportunity within our parameters (2,628 NegRisk, 4,423 Single), with most conditions having only a few opportunities (see Fig. 6). All arbitrage opportunities observed are *long*, meaning the sum of

[11] Since placing multiple orders in an order book is non-atomic (only a subset of the attempts may succeed), there is some inherent risk to attempting arbitrage.

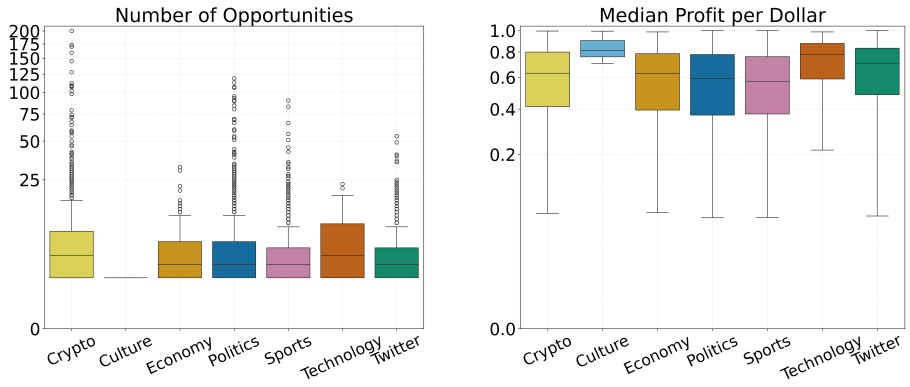


Figure 6 Arbitrage opportunities detected within a single condition. In Appendix H.1 in the full version [32], we show NegRisk and Single Markets separately. We see that most conditions have only a few opportunities, with Crypto having the biggest outliers. Most surprisingly, the median profit on the dollar for these opportunities is much higher than our 2 cents bound, and highlights large market inefficiency.

the price of "YES" and "NO" is less than 1. We see the median price of the sum of conditions (i.e., profit per Dollar) is around \$0.60 for all topics of markets, showing remarkable market inefficiency.

As taking advantage of one arbitrage opportunity may impact the prices afterwards, we bound the possible arbitrage profit by considering the most profitable arbitrage opportunity for each condition. We define the maximum profit of an opportunity as the price for both "YES" and "NO" tokens times the total amount of tokens that exist (i.e., the maximum one could purchase from the market at that time). Figure 7 top shows the sum of this value across all conditions, differentiating between the NegRisk and Single condition markets. We see that most potential profit comes from markets related to politics, particularly those related to the U.S. 2024 presidential election. Assuming one could take advantage of arbitrage at 1% of the available tokens, this is still millions in potential profit.

6.2 Arbitrage Within Markets

Next, we consider arbitrage that is possible between the conditions of a NegRisk market. Of the 1578 NegRisk markets, 662 had at least one arbitrage opportunity. In Appendix H.2 in the full version [32], we show the distribution of several characteristics of these opportunities. In general, we see many more opportunities per market, averaging around 100 per market, with quite high outliers, particularly in Sports. Within a market we also see both short (sum of "YES" is more than \$1) and long (less than \$1) arbitrage, though the average maximum profit on the dollar is more for the long arbitrage, but with short having significant outliers.

We again look at the maximum profit possible for each market from a single opportunity. Since each condition in a market may have a different volume, we take the minimum volume across all conditions that have a probability more than 2% (when performing arbitrage, it may be low risk to ignore low probability events). Figure 8 shows the sum of these opportunities for *long* and *short* positions separately. While the average maximum opportunity for both short and long arbitrage across topics is approximately the same (40 cents on the Dollar), we see in the cumulative of these opportunities that Sports dominates across all months except for the U.S. election period. In Appendix H.2 in the full version [32], we again consider a modest arbitrageur with a \$100 budget; there is twice as much profit in long arbitrage, with Sports dominating both, suggesting arbitrage opportunities are generally common within Sports markets.

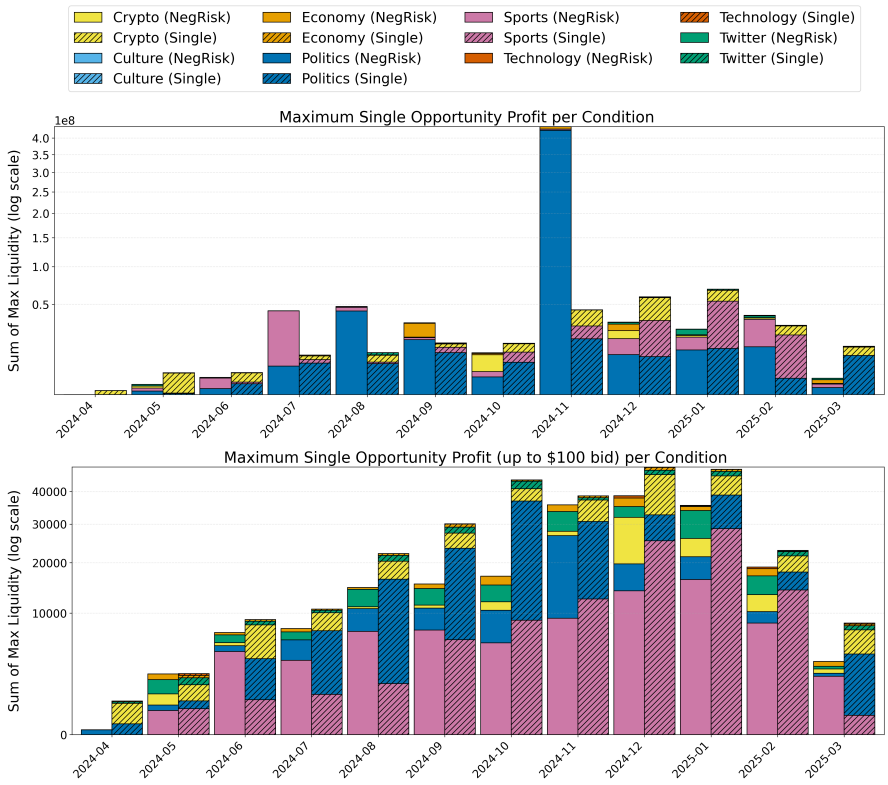


Figure 7 We explore the total arbitrage possible if an arbitrageur were to take advantage of the single most profitable opportunity in each condition at the maximum liquidity (top), and up to just $100 of liquidity (bottom). The latter shows us that the existence of arbitrage is quite split among markets, and many more opportunities exist in Sports markets. Without capping profit (top), we see that the very lucrative opportunities are predominantly in Politics, with higher profits per condition (mostly in NegRisk markets). In Appendix H.1 in the full version [32], we compare results from price averages over 100 blocks and we find that we capture less arbitrage generally, and particularly less in the Politics markets of November and Sports markets of July – both of which were highly exploited cf. Fig. 10.

6.3 Arbitrage Across Markets

Lastly, in this section, we look at possible arbitrage opportunities across the 13 dependent pairs of markets during the 2024 U.S. presidential election. For the pairs of markets, we again focus on arbitrage relating to the price of the "YES" token of all conditions. In Figure 9, we show the distribution of profit per USDC and maximum profit for each opportunity given the liquidity at the time. The pairs are shown in order of number of opportunities, with Pair 8 having none, and the median among the rest being 8 opportunities (pairs 2, 1, and 4 have 72, 176, and 6630 opportunities, respectively). The markets of Pair 4 are on who will win the popular vote, and if the popular vote winner will the presidency (recall the description of each pair is in Appendix F in the full version [32]). We note that though we were able to observe arbitrage opportunities, they are largely during lower liquidity moments, and at lower profit than the previous sections (the average max profit is around $100, suggesting total token volume in the market of less than 2K).

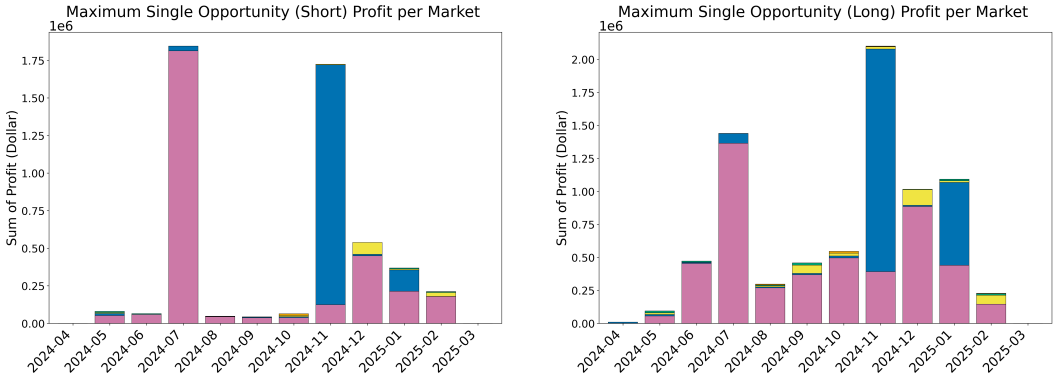


Figure 8 Here we explore the total arbitrage possible across each market if an arbitrageur were to take advantage of the single most profitable opportunity. Overall, Sports consistently has more profit, particularly in Long opportunities, besides a couple of outliers in Politics, suggesting Sports markets are often overvalued.

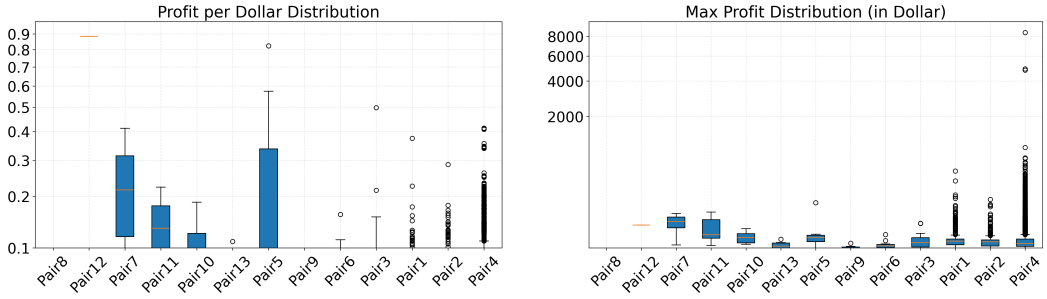


Figure 9 Distribution of profit per Dollar (left) and profit assuming maximum liquidity (right) for the 11 pairs of dependent markets from the U.S. election in order of number of arbitrage opportunities.

7 Markets Analysis: Uncovering Arbitrageurs

7.1 Bids Processing and Window Size

Having explored the space of arbitrage opportunities, we next detect whether any actors took advantage of these opportunities. For each user (single Polygon address), we take the history of all bids – buys, sells, splits accompanied with sells(the latter can be used the compute the price for which the user held the opposing position, e.g., a user that creates 1 "YES"/"NO" position and then sells the "YES" for \$0.70 effective holds a "NO" at price \$0.30.). In Appendix I.1 in the full version [32], we summarize the set of 86 million bids. To make the data more manageable but still capture the majority of arbitrage value, we filter bids below \$2.00. We then group all executed bids from a user and consider bids within a time window T as part of the same opportunity. We set T to 950 blocks (approx. 1 hour) to capture some delay in bids being matched (75% of bids fall within this window – see Appendix I.1 in the full version [32] for the distribution). We then calculate a rolling window of positions a bought and at what price. We calculate profit from the minimum amount of tokens held across all positions minus the price to acquire the positions. It is important to clarify that we do not take into account the fees because Polymarket currently does not charge per trade executed.

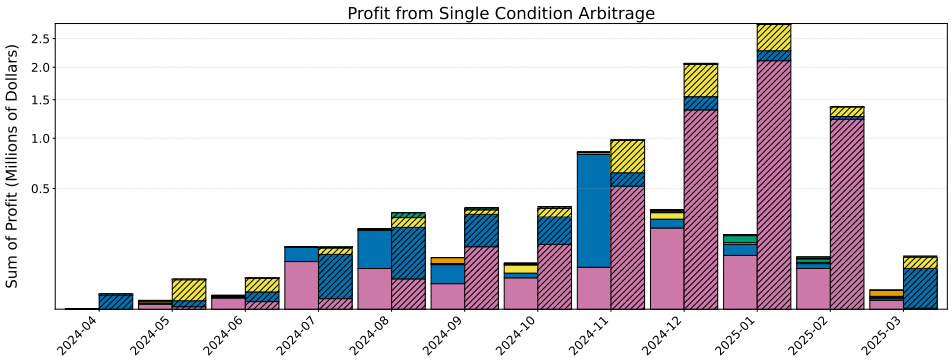


Figure 10 Total profit made by all users in single condition arbitrage. About 1% of our estimated U.S. election opportunities were exploited by users. Interestingly, Sports Single markets dominate in exploited opportunities, surpassing the election profit.

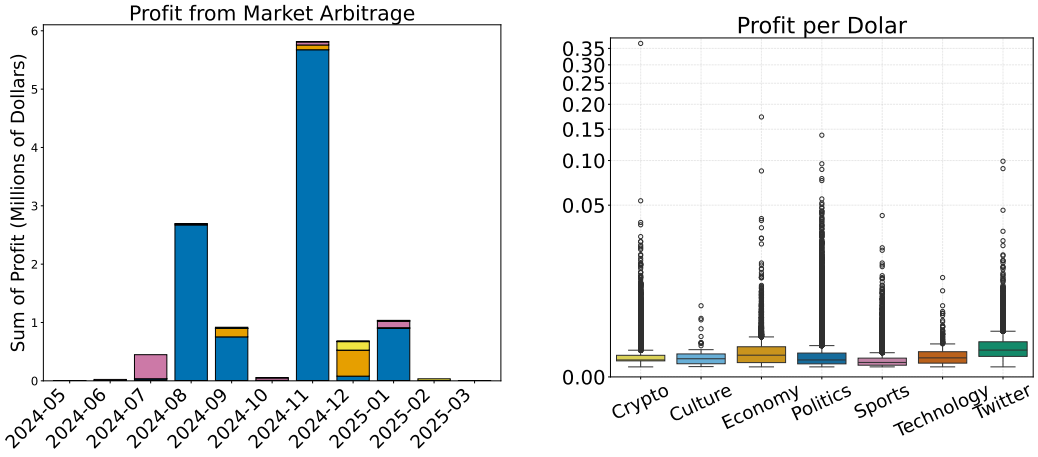


Figure 11 (Left) The total profit obtained through Rebalancing Arbitrage in NegRisk markets. We see that most profit is found in markets relating to Politics. Sports is surprisingly absent from this figure, likely due to a smaller scale of profits. (Right) The distribution of opportunities by potential yield per dollar. Most opportunities exhibit low returns, while a few outliers – primarily in Crypto, Politics, and Twitter markets – offer significantly higher yields.

7.2 Arbitrage Within Single Conditions

We first consider strategies within a single condition – recall from Section 7 that the largest profit was seen in this type of arbitrage. We look at the price the user acquired "YES" and "NO" tokens and find all instances when the sum of prices deviates from $1. Figure 10 shows the total arbitrage captured by all users – we see that these types of opportunities are largely captured. Of the two strategies, the total profit from *buying below one dollar* amounts to $5,899,287.427, whereas the total profit from *selling above one dollar* is $4,682,074.77.

An interesting phenomenon we observed was the presence of opportunities with extremely high market discounts. The most striking example was executed by the user `@Tutaaa91`, who simultaneously purchased both *"YES"/"NO"* tokens for less than $0.02 each, resulting in a single-trade profit of $58,983.36 (other two trades from this account also exhibited high returns). This phenomenon arises when prices are mismatched with real-world probabilities, creating an opportunity to exploit the discrepancy – an interesting area for future work.

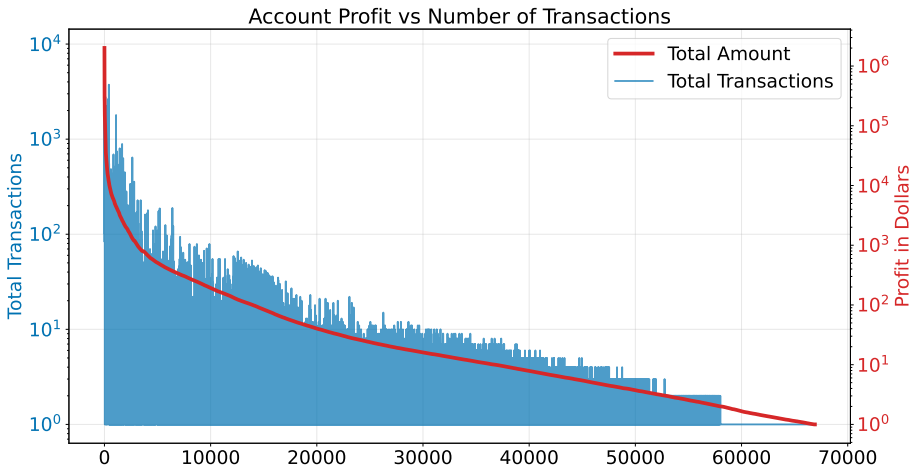


Figure 12 Total number of bids across accounts plotted alongside their aggregate profits in dollars. Both y-axes are represented on a logarithmic scale.

7.3 Arbitrage Within and Between Markets

Next, we look at strategies within a single market. Many markets include extraneous low-probability conditions that a user can safely ignore when performing arbitrage on the higher-probability events. To handle this, we use the approach from Section 6 to estimate the price of the missing positions. We include this missing price in characterizing if arbitrage exists (to not undercount the total probabilities). We calculate the total profit for each strategy as follows: buying "YES" \$11,092,286.31, selling "YES" \$612,188.83, selling "NO" \$4,264.33, and buying "NO" \$17,307,113.81. Compared with single condition arbitrage, selling becomes more difficult across multiple markets, and buying seems to be the predominant strategy. Interestingly, buying "NO" outperforms other strategies. In fact, Polymarket announced that some accounts are making a lot of profit just by buying "NO" positions (see related Polymarket tweets [26, 25]).

We plot the total profit from single market arbitrage and the distribution of profit on the Dollar in Figure 11. Surprisingly, Sports are largely absent from the plots – maybe a less explored venue for arbitrageurs. Additionally, Politics dominates in extracted arbitrage, with the U.S. election in November and predominantly two markets in August relating to the Democratic party pick and VP pick following Biden's drop from the election.

We also investigate arbitrage opportunities among the 13 dependent U.S. election pairs identified in Section 5. No executed arbitrage was detected in the 2 NegRisk-Single pairs. Among the remaining 11 pairs, we found evidence of value extraction in 5 cases. Recall that our check focuses on the sum of `YES` outcomes across dependent subsets of the pairs. The extracted total arbitrage across market pairs is as follows: pair 2, \$60,236.71; pair 4, \$18,472.31; pair 1, \$15,818.53; pair 3 \$629.16, (see Appendix F in the full version [32] for pair descriptions). Interestingly, one market on the GOP presidential win margin is in pairs 1 and 2, and one market on the balance of power among the presidency, House, and Senate, is in pairs 2 and 4.

Table 1 Top 10 accounts ranked by total amount and number of successful opportunities.

#	Account (trimmed)	Amount ($)	Transactions
1	0xd218e474776403a3301422...	2,009,631.76	4,049
2	0x63d43bbb87f85af03b8f2f...	1,273,058.68	2,215
3	0x9d84ce0306f8551e02efef...	1,092,616.17	4,294
4	0x44c1dfe43260c94ed4f1d0...	768,565.50	211
5	0x59ee6c6a56d7b00223f0c3...	749,795.99	3,468
6	0xd42f6a1634a3707e27cbae...	537,959.59	4,533
7	0x4a64afa45a44a01890c216...	476,766.58	3,341
8	0xb7d54bf1d0a362beb916d9...	468,391.71	2,287
9	0x53d2d3c78597a78402d4db...	424,505.34	200
10	0x3cf3e8d5427aed066a7a59...	383,569.94	2,720

7.4 Arbitrageur Strategies

Lastly, we briefly explore the strategies of top arbitrageurs. For all arbitrage we detect, we sum the total profit made by each account and the number of executed bids that went into this profit. We plot this in Figure 12. The combined amount extracted from all strategies presented totals $39,587,585.02, assuming an $\varepsilon = \$1$ profit per trade. We see some very big players with bot-like behaviour in the number of bids they participate in. The user with the highest profits generated a total of $2,009,631.76 through these strategies. The top 10 users by profit can be found in Table 1. A deeper study of these strategies is left for future work.

8 Concluding Discussion

Despite the relatively modest volume of arbitrage compared to other markets like decentralized exchanges, where transactions are atomic and risk-free, our findings contribute valuable insights into the dynamics of prediction markets. While our analysis of Polymarket data revealed a limited number of dependent markets, our methodology remains applicable to future prediction markets, particularly as platforms evolve towards greater decentralization. As arbitrageurs develop more specialized strategies, akin to those observed in decentralized finance (DeFi) automated market makers, we anticipate an increase in dependent markets emerging as part of these strategies. Some limitations in the reasoning of the LLM for certain types of markets (e.g., reasoning loops we encountered) underscore the need for future enhancements in capabilities to better identify and interpret such dependencies, as well as to be able to handle larger inputs (allowing for determining dependencies among larger sets of markets). Our focus was on identifying unequivocal arbitrage opportunities – situations where purchasing positions guarantees a profit. However, the LLM-generated outcome tables also provide insights into weaker dependencies. For instance, in scenarios where one market pertains to "*Team A wins the semifinal*" and another to "*Team A wins the final*," the outcome of the first influences the second, creating a temporal window for arbitrage based on logical dependencies. Studying strategies in this weaker space of dependency remains an interesting open problem.

References

1 Linq-embed-mistral. https://huggingface.co/Linq-AI-Research/Linq-Embed-Mistral.

2 MTEB leaderboard – Embedding benchmark. URL: https://huggingface.co/spaces/mteb/leaderboard.

3 Public node rpc endpoint for polygon pos. URL: https://www.alchemy.com/chain-connect/endpoints/public-node-polygon-pos.

4 Deepseek-r1-distill-qwen-32b. URL: https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-32B, 2025.

5 Jon Scott Armstrong. *Principles of forecasting: a handbook for researchers and practitioners*, volume 30. Springer Science & Business Media, 2001.

6 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE symposium on security and privacy (SP)*, pages 910–927. IEEE, 2020. doi:10.1109/SP40000.2020.00040.

7 Rainer Feichtinger, Robin Fritsch, Lioba Heimbach, Yann Vonlanthen, and Roger Wattenhofer. Sok: Attacks on daos. In *6th International Conference on Advances in Financial Technologies (AFT), Vienna, Austria*, 2024. arXiv:2406.15071.

8 Christof Ferreira Torres, Albin Mamuti, Ben Weintraub, Cristina Nita-Rotaru, and Shweta Shinde. Rolling in the shadows: Analyzing the extraction of mev across layer-2 rollups. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 2591–2605, 2024. doi:10.1145/3658644.3690259.

9 Walter Frick. Journalists wake up to the power of prediction markets. https://www.niemanlab.org/2022/12/journalists-wake-up-to-the-power-of-prediction-markets/, 2022.

10 Vahid Ghafouri, Vibhor Agarwal, Yong Zhang, Nishanth Sastry, Jose Such, and Guillermo Suarez-Tangil. Ai in the gray: Exploring moderation policies in dialogic large language models vs. human answers in controversial topics. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, CIKM '23, pages 556–565, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3583780.3614777.

11 Gnosis Ltd. *Conditional Tokens Documentation*. Gnosis Ltd., 2020. Version 1.0.3. URL: https://conditional-tokens.readthedocs.io/_/downloads/en/latest/pdf/.

12 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint*, 2025. arXiv:2501.12948.

13 Robin Hanson. The policy analysis market (and futuremap) archive, 2003. Accessed: 2025-05-12. URL: https://mason.gmu.edu/~rhanson/policyanalysismarket.html.

14 Lioba Heimbach, Lucianna Kiffer, Christof Ferreira Torres, and Roger Wattenhofer. Ethereum's proposer-builder separation: Promises and realities. In *Proceedings of the 2023 ACM on Internet Measurement Conference*, pages 406–420, 2023. doi:10.1145/3618257.3624824.

15 Lioba Heimbach, Vabuk Pahari, and Eric Schertenleib. Non-atomic arbitrage in decentralized finance. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3866–3884. IEEE, 2024. doi:10.1109/SP54263.2024.00256.

16 Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677, 2024. doi:10.18653/V1/2024.ACL-LONG.91.

17 Jaynti Kanani, Sandeep Nailwal, and Anurag Arjun. Matic network: Scalable and instant blockchain transactions, 2019. Accessed: 2025-05-29. URL: https://github.com/maticnetwork/whitepaper.

18 Genki Kusano, Kosuke Akimoto, and Kunihiro Takeoka. Are longer prompts always better? prompt selection in large language models for recommendation systems. *arXiv preprint*, 2024. doi:10.48550/arXiv.2412.14454.

19 Jason Milionis, Ciamac C Moallemi, and Tim Roughgarden. Automated market making and arbitrage profits in the presence of fees. In *International Conference on Financial Cryptography and Data Security*, pages 159–171. Springer, 2024. doi:10.1007/978-3-031-78676-1_9.

20 Burak Öz, Danning Sui, Thomas Thiery, and Florian Matthes. Who wins ethereum block building auctions and why? *arXiv preprint*, 2024. doi:10.48550/arXiv.2407.13931.

21 Burak Öz, Christof Ferreira Torres, Jonas Gebele, Filip Rezabek, Bruno Mazorra, and Florian Matthes. Pandora's box: Cross-chain arbitrages in the realm of blockchain interoperability. *arXiv preprint*, 2025. doi:10.48550/arXiv.2501.17335.

22 Polymarket. Fide world blitz chess champion 2024. https://polymarket.com/event/fide-world-blitz-champion?tid=1747638945505, 2024. Prediction market on the 2024 FIDE World Blitz Chess Championship. URL: https://polymarket.com/event/fide-world-blitz-champion?tid=1747638945505.

23 Polymarket. Presidential election winner– 2024 market. https://polymarket.com/event/presidential-election-winner-2024, 2024.

24 Polymarket. py-clob-client: Python client for polymarket clob. https://github.com/Polymarket/py-clob-client, 2024. Accessed: 2025-05-14.

25 Polymarket. Tweet on market dynamics and liquidity. https://x.com/Polymarket/status/1935417783757738350, 2024. Accessed: 2025-07-31.

26 Polymarket. Tweet on unusual trading activity. https://x.com/Polymarket/status/1911874780678009202, 2024. Accessed: 2025-07-31.

27 Polymarket. Polymarket: A decentralized information markets platform. https://polymarket.com, 2025. Accessed: 2025-04-16.

28 Polymarket Documentation. Clob introduction. https://docs.polymarket.com/developers/CLOB/introduction#clob-introduction, 2025. Accessed: 2025-05-18.

29 Polymarket Documentation Team. How are markets created?, 2025. Accessed: 2025-05-18. URL: https://docs.polymarket.com/polymarket-learn/markets/how-are-markets-created.

30 @primo_data. Uma voter analysis in polymarket disputes. https://dune.com/primo_data/uma-voter-polymarket-disputes, 2024. Dune Analytics dashboard analyzing UMA voter behavior in Polymarket dispute resolutions. URL: https://dune.com/primo_data/uma-voter-polymarket-disputes.

31 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 198–214. IEEE, 2022. doi:10.1109/SP46214.2022.9833734.

32 Oriol Saguillo, Vahid Ghafouri, Lucianna Kiffer, and Guillermo Suarez-Tangil. Unravelling the probabilistic forest: Arbitrage in prediction markets. *CoRR*, 2025. IMDEA Networks; Oxford Internet Institute. doi:10.48550/arXiv.2508.03474.

33 Philip E Tetlock. *Expert political judgment: How good is it? How can we know? – New edition*. Princeton University Press, 2017.

34 UMA Protocol. Uma dvm oracle whitepaper. https://github.com/UMAprotocol/whitepaper/blob/master/UMA-DVM-oracle-whitepaper.pdf, 2025. Accessed: 2025-04-16.

35 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. arXiv:2203.11171.

36 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.

37 Ben Weintraub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. A flash (bot) in the pan: measuring maximal extractable value in private pools. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 458–471, 2022. doi:10.1145/3517745.3561448.

38 Jeremy Whittaker. Accessing polymarket data in python, August 2024. URL: https://jeremywhittaker.com/index.php/2024/08/20/accessing-polymarket-data-in-python/. Accessed: 2025-05-14.

39 Justin Wolfers and Eric Zitzewitz. Prediction markets. *Journal of economic perspectives*, 18(2):107–126, 2004.

40 Mahshar Yahan and Dr. Mohammad Islam. Harnessing NLP for indigenous language education: Fine-tuning large language models for sentence transformation. In Manuel Mager, Abteen Ebrahimi, Robert Pugh, Shruti Rijhwani, Katharina Von Der Wense, Luis Chiruzzo, Rolando Coto-Solano, and Arturo Oncevay, editors, *Proceedings of the Fifth Workshop on NLP for Indigenous Languages of the Americas (AmericasNLP)*, pages 119–125, Albuquerque, New Mexico, May 2025. Association for Computational Linguistics. URL: https://aclanthology.org/2025.americasnlp-1.14/.

41 Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. Sok: Mev countermeasures. In *Proceedings of the Workshop on Decentralized Finance and Security*, pages 21–30, 2024. doi:10.1145/3689931.3694911.

42 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: deliberate problem solving with large language models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

43 Jianting Zhang and Aniket Kate. No fish is too big for flash boys! Frontrunning on DAG-based blockchains. *Cryptology ePrint Archive*, 2024.

44 Yiming Zhu, Zhizhuo Yin, Gareth Tyson, Ehsan-Ul Haq, Lik-Hang Lee, and Pan Hui. Apt-pipe: A prompt-tuning tool for social data annotation using chatgpt. In *Proceedings of the ACM Web Conference 2024*, WWW '24, pages 245–255, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3589334.3645642.

45 Yusuf Mücahit Çetinkaya, Vahid Ghafouri, Guillermo Suarez-Tangil, Jose Such, and Tuğrulcan Elmas. Cross-partisan interactions on twitter, 2025. doi:10.48550/arXiv.2504.09376.

Optimistic MEV in Ethereum Layer 2s: Why Blockspace Is Always in Demand

Ozan Solmaz
ETH Zürich, Switzerland

Lioba Heimbach
ETH Zürich, Switzerland

Yann Vonlanthen
ETH Zürich, Switzerland

Roger Wattenhofer
ETH Zürich, Switzerland

Abstract

Layer 2 rollups are rapidly absorbing DeFi activity, securing over $40 billion and accounting for nearly half of Ethereum's DEX volume by Q1 2025, yet their MEV dynamics remain understudied. We address this gap by defining and quantifying optimistic MEV, a form of speculative, on-chain MEV whose detection and execution logic reside largely on-chain in smart contracts. As a result of their speculative nature and lack of off-chain opportunity verification, optimistic MEV transactions frequently decide not to execute any trades.

In this work, we focus on cyclic arbitrage, which we find is predominantly executed as optimistic MEV on Layer 2s. Using our multi-stage identification pipeline on Arbitrum, Base, and Optimism, we show that in Q1 2025, transactions from cyclic arbitrage contracts account for over 50% of on-chain gas on Base and Optimism and 7% on Arbitrum, driven mainly by "interaction" probes (on-chain computations searching for arbitrage). This speculative probing indicates that cyclic arbitrage on Layer 2s is predominantly executed as optimistic MEV and contributes to generally keeping blocks on Base and Optimism persistently full. Despite consuming over half of on-chain gas, these optimistic MEV transactions pay less than one quarter of total gas fees. Cross-network comparison reveals divergent success rates, differing patterns of code reuse, and sensitivity to varying sequencer ordering and block production times. Finally, OLS regressions link optimistic MEV trade count to ETH volatility, retail trading activity, and DEX aggregator usage. Together, these findings show that optimistic MEV has become a major source of persistent spam-like transaction activity on Layer 2s, dominating blockspace with low-value probes and reshaping the composition of on-chain activity.

2012 ACM Subject Classification Computer systems organization → Distributed architectures; Applied computing → Electronic commerce; Networks → Network measurement

Keywords and phrases blockchain, MEV, Layer 2, Ethereum

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.28

Related Version *Full Version*: `https://arxiv.org/abs/2506.14768` [50]

Supplementary Material
Collection (Source Code): `https://github.com/kingbaldwin-iv/Optimistic-MEV` [43]

Acknowledgements We thank Burak Öz, Vabuk Pahari and the anonymous AFT reviewers for their helpful feedback. Our gratitude extends to Patrick Züst for his exploratory work in his Master thesis supervised by the last three authors and Quentin Kniep, which helped shape this work.

© Ozan Solmaz, Lioba Heimbach, Yann Vonlanthen, and Roger Wattenhofer;
licensed under Creative Commons License CC-BY 4.0

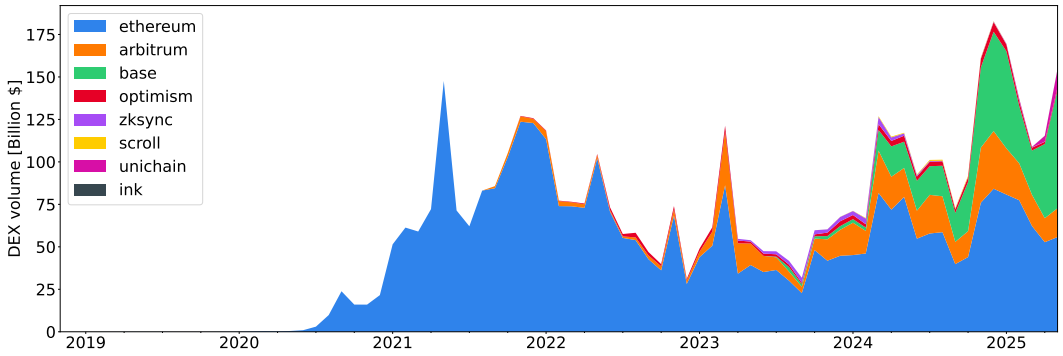


Figure 1 Monthly DEX transaction volume by network from November 2018 to May 2025. The plot highlights the significant rise of Layer 2 networks – such as Arbitrum, Base, Optimism, and others – in total DEX activity, with Ethereum Layer 1 gradually representing a smaller share of the overall volume.

1 Introduction

Ethereum, a decentralized and programmable blockchain, features robust smart contract functionality that enables trust-minimized applications and value transfer without reliance on traditional intermediaries. This programmability has cultivated a dynamic ecosystem of decentralized applications, particularly within the sphere of *decentralized finance (DeFi)*, which includes *decentralized exchanges (DEXes)* and *lending protocols*. As DeFi and other on-chain activities have grown in scale and value, they have contributed to significant network congestion on Ethereum Layer 1, driving transaction fees to levels that render smaller-scale operations economically unviable.

To address these critical scaling limitations, the Ethereum community has strategically embraced a rollup-centric development roadmap [9, 45, 30]. Layer 2 rollups scale Ethereum by moving transaction execution off-chain and periodically anchoring summarized results on-chain. This design preserves security while enabling higher throughput and lower costs [27, 33]. Adoption has been significant: as of April 2025, Layer 2 networks secure around $40 B in assets and account for a growing share of on-chain transaction volume, highlighting their central role in the Ethereum ecosystem.

As illustrated in Figure 1, a substantial share of DeFi activity now occurs on Layer 2 networks. To be precise, in the first quarter of 2025, roughly 47% of DEX volume in the Ethereum ecosystem was happening on Layer 2 networks. This figure is up from 35% in the first quarter of 2024. This migration of DeFi activity to Layer 2 networks has concomitantly fostered a distinct *Maximal Extractable Value (MEV)* landscape.

The shift in the MEV landscape has been accelerated by the advent of Layer 2 rollups and upgrades such as EIP-4844 Proto-Danksharding [21, 22], which have significantly reduced data availability costs for Layer 2 networks and, in turn, lowered transaction fees on these networks. These lower costs have allowed for MEV extraction strategies that are less viable on the Ethereum mainnet. Beyond lower fees, Layer 2s introduce operational characteristics that further enable MEV strategies previously infeasible on Layer 1. These include differing transaction ordering policies implemented by sequencers – ranging from simple *First-Come, First-Served (FCFS)* to auction-based mechanisms such as *Priority Gas Auctions (PGA)* – as well as variations in mempool privacy, and short inter-block times.

The confluence of these features reduces the financial risk and increases the execution uncertainty of MEV-seeking transactions. As a result, cyclic arbitrage bots can adopt an optimistic approach: they issue high-frequency, speculative transactions without knowing in advance whether an arbitrage opportunity exists, relying instead on rapid on-chain state reads and being sufficiently close in time to the opportunity-creating transaction to back-run it successfully. This differs from traditional cyclic arbitrage on Ethereum Layer 1, where generally bots precompute a guaranteed profit path and bundle all trades into a single cyclic transaction. On Layer 2, optimistic strategies involve repeatedly probing liquidity pools for small gains, despite not knowing if an opportunity exists, and accepting a high failure rate in exchange for marginal profits.

We refer to this class of strategies that rely on speculative execution, where MEV bots submit transactions dependent on on-chain computation to identify profitable opportunities, often resulting in failed attempts, as *optimistic MEV*. To the best of our knowledge, this is the first work to formalize and systematically study optimistic MEV on Layer 2s.

Our contributions. We summarize our main contributions below.

- **i.** We provide the first definition of *optimistic MEV*, describing it as a class of MEV extraction strategies that rely heavily on on-chain logic for opportunity identification and speculative execution.
- **ii.** We design and implement a multi-stage pipeline to identify and classify MEV transactions. This includes the construction of a cyclic arbitrage detector, an on-chain behavior classifier, and a revert/success labeling system to capture speculative dynamics.
- **iii.** We apply our methodology to present the first large-scale measurement of optimistic MEV on major Layer 2s, showing that cyclic arbitrage accounts for roughly 7% of gas usage on Arbitrum, 51% on Base, and 55% on Optimism in Q1 2025 and is predominantly optimistic in nature. In particular, out analysis reveals that a large share of this gas is consumed by transactions that fail to execute profitable trades, supporting the hypothesis that these bots engage in speculative on-chain probing.
- **iv.** We compare optimistic MEV activity across Arbitrum, Base, and Optimism and find substantial differences in strategy execution patterns, success rates, and gas usage concentration. We attribute these to network-specific factors such as transaction ordering policies and inter-block times.
- **v.** Using regression analysis, we examine how optimistic MEV activity correlates with market conditions and user trade behavior. Our findings suggest that volatility, trade volume, and aggregator usage significantly impact optimistic MEV prevalence.
- **vi.** We discuss the broader implications of optimistic MEV for the MEV ecosystem and Layer 2 design, highlighting its economic impact and the challenges it poses for existing mitigation approaches.

2 Related Work

Maximal Extractable Value. The study of MEV was commenced by Eskandari et al. [20] and Daian et al. [12], who first defined MEV and documented phenomena like front-running and PGAs on Ethereum. In subsequent work, Qin et al. [47] and Torres et al. [54], quantified various forms of MEV on Ethereum mainnet, including sandwich attacks (i.e., a type of front-running attack on DEXes), liquidations of positions on lending protocols, and cyclic arbitrage, providing a baseline understanding of its scale and impact. A subsequent line of work analyzes various and evolving aspects of the MEV landscape on the Ethereum Layer

1 [62, 61, 54, 59, 44, 48, 10, 49, 39, 60, 29, 64, 31]. While these works have established a foundation for understanding MEV on Ethereum Layer 1, the Layer 2 landscape remains comparatively underexplored. Our work builds on this foundation by focusing on the unique characteristics of Layer 2s, providing a classification and analysis of MEV transactions in these emerging environments.

A closely related line of work examines MEV in alternative Layer 1 networks, which operate under distinct architectural and economic conditions. Öz et al. [63] analyze MEV on FCFS blockchains such as Algorand, identifying latency optimization – rather than fee bidding – as the dominant extraction mechanism. Further, work done by Umbra Research [56] has provided valuable insights into the MEV on Solana. They point to the existence of optimistic MEV on Solana but provide no in-depth analysis of the phenomenon. In contrast, our work focuses on Ethereum Layer 2 networks, where optimistic MEV is both prevalent and, to the best of our knowledge, remains largely unstudied.

Maximal Extractable Value in Layer 2 Networks. Recent efforts have begun to examine MEV on Layer 2 networks. Torres et al. [55] conducted a comparative analysis of MEV across Ethereum and major Layer 2s (Arbitrum, Optimism, zkSync). Theoretical work has also addressed cross-domain MEV, with Obadia et al. [41] formalizing MEV across domains such as different Layer 2s. Complementing this, Gogol et al. [26] empirically analyzed non-atomic cross-rollup MEV and CEX-DEX arbitrage. Öz et al. [42] present the first systematic study of non-atomic cross-chain arbitrage strategies across nine blockchains, including several Layer 2s. Their work highlights the effects of these strategies on network congestion and the security implications of cross-chain MEV. Our work extends these Layer 2-focused studies by analyzing optimistic MEV, a form of MEV that involves on-chain detection of opportunities.

Decentralized Exchanges. Theoretical frameworks for routing on DEXes and profits of liquidity providers have provided important context for understanding MEV behavior. Angeris et al. [5] analyzed optimal routing and arbitrage in CFMMs. Milionis et al. [37, 38] modeled loss-versus-rebalancing (LVR) under Poisson-distributed interblock times; and Nezlobin et al. [40] extended this to deterministic block intervals, particularly relevant given the regular block production in many blockchain networks. Building on these theoretical foundations, our work provides empirical insights into gas usage, MEV bot behavior, and protocol-level design factors across Layer 2 networks, focusing on optimistic MEV: a speculative, high-frequency form of MEV.

3 Optimistic MEV

Next, we introduce optimistic MEV: a class of extraction strategies that (1) perform opportunity discovery and trade execution primarily through on-chain computation and (2) rely on speculative transaction submission without prior off-chain verification of profitability.

This behavior is most evident in *optimistic cyclic arbitrage*, the focus of this work. The MEV types under consideration are sandwich attacks, cyclic arbitrage (arbitrage between DEXes on the same chain), non-atomic arbitrage (e.g., CEX–DEX arbitrage), and liquidations of positions on lending protocols. Of these, cyclic arbitrage offers a practically viable case where opportunities can be identified and executed entirely on-chain in a single transaction through frequent probing. While liquidations are in principle detectable on-chain, they occur only when loans become undercollateralized and are likely too infrequent to support high-frequency probing [47]. Sandwich attacks require two separate transactions, and non-atomic

arbitrage depends on external price data from off-chain venues. For this reason, we focus our analysis of optimistic MEV on cyclic arbitrage, as this MEV strategy can be detected and executed fully on-chain.

Empirically, we observe this form of MEV most prominently on low-fee Layer 2 networks. Its defining signature is speculative execution: bots submit transactions without certainty that an arbitrage opportunity exists, using cheap blockspace to make such probing economically viable. In our analysis, these transactions frequently begin with top-level `STATICCALL` operations directed at DEX liquidity pools to read real-time reserves and prices. The contract then decides whether to execute a swap or terminate without trading based on these on-chain checks. This behavior, combined with the common use of minimal or repeated `calldata`, indicates that key parameters, i.e., the path of the arbitrage transaction and the swap amounts, are determined during execution rather than precomputed off-chain. The reliance on such runtime inspection also explains why optimistic MEV becomes infeasible when transaction fees are high: the preparatory on-chain calls consume significant gas. These observations motivate the following definition of optimistic MEV.

▶ **Definition 1 (Optimistic MEV).** Optimistic MEV *denotes MEV strategies in which the existence and parameters of a profitable opportunity are determined within the transaction itself through on-chain computation at execution time, rather than being verified off-chain prior to submission. This leads to speculative transactions where profitability is only resolved during execution, resulting in a significant share of transactions that terminate without performing a trade.*

Case Study. To concretely illustrate the observed Layer 2 MEV patterns, particularly the prevalent use of initial `STATICCALL` operations for on-chain reconnaissance, we present a comparative case study of two transactions interacting with the same contract `0xf5ff...a8c0`[1]: `0x1d97...18cb` (henceforth Tx_A) and `0xb678...9bd7` (henceforth Tx_B) on Base. Both transactions were initiated with the same function selector, namely `0x00003748`, as evidenced by the identical first 4 bytes in the calldata. This common entry point suggests a shared execution pathway, likely designed for conditional arbitrage based on real-time market conditions. Despite this identical initiation, the on-chain outcomes of Tx_A and Tx_B diverged significantly. Tx_A resulted in no token swaps. In stark contrast, Tx_B successfully executed a 2-swap cyclic arbitrage, cycling value through the path $WETH \to TYBG \to WETH$ and realizing a profit.

```
{
"calls": [
    {"input": "slot0()", "to": "UNIV3 ETH-USDC Pool", "type": "STATICCALL" },
    {"input": "slot0()", "to": "UNIV3 PLAY-USDC Pool", "type": "STATICCALL" },
    {"input": "slot0()", "to": "CL ETH-PLAY Pool", "type": "STATICCALL" }
    // ... further calls omitted for brevity
  ],
  "from": "0xdd57...a88",
  "to": "0xf5ff...8c0",
  "type": "CALL"
  }
```

Example 1 Condensed initial call sequence from Tx_A, illustrating `STATICCALL` probes to DEX pools. Input `slot0()` retrieves key state variables from Uniswap V3-compatible pools.

[1] We marked this contract as one of the top gas consuming cyclic arbitrage contracts on Base, which we will discuss in more detail in the later sections.

A detailed analysis of the execution trace for Tx_A reveals that the contract systematically probed multiple DEX liquidity pools via `STATICCALL` operations. These read-only calls occurred before any attempt to execute a trade, and the transaction subsequently terminated. This sequence of actions is strongly indicative of an on-chain reconnaissance strategy, where the contract assesses current pool states (e.g., token reserves, potential slippage, pool's fee structure, or tick data) to determine the viability of an arbitrage opportunity before committing to an execution path. A condensed representation of the initial call sequence within Tx_A is provided in Example 1.

4 Data Classification

To study optimistic MEV, we start by collecting relevant data (see Appendix C of the full version [50]) from the Ethereum Layer 1 as well as Arbitrum, Base, and Optimism (i.e., the biggest three Layer 2s in terms of DEX volume at the time of this writing). Then, we systematically identify cyclic arbitrage MEV activities on Layer 2 networks using a three-stage classification methodology. Recall that we focus on optimistic cyclic arbitrage in our work, as cyclic arbitrage opportunity detection can be fully resolved on-chain in a single transaction. In Section 5, we will then show that cyclic arbitrage on Layer 2 networks predominantly operates optimistically.

The implementation of our cyclic arbitrage classification pipeline, as well as the classification output (the full list of cyclic arbitrage contracts) are made openly accessible [43].

4.1 Cyclic Arbitrage Contract Detection

Building on the cyclic arbitrage detection heuristic of Wang et al. [59] and the toolkits behind EigenPhi and `libmev` [17, 35], we implement a cyclic arbitrage contract detection pipeline in Dune SQL, leveraging Dune Analytics' data tables and the Torino engine. Inspired by the replay logic of `mev-inspect-py` [23] and *Brontes* [51] we adapt our methodology to the Layer 2 circumstances.

We start by computing the set of transactions that, with high likelihood, perform cyclic arbitrage:

1. **Token Path & Balance Reconstruction.** We consider all transactions containing swap events by querying the swap logs from `dex.trades` [16]. Each transaction's swap events are ordered to recover the exact token path and the initiator's net token balance.
2. **Router & Aggregator Filter.** Transactions that directly interact with labeled routers (captured through the metadata of the `dex.addresses` [13] table) or aggregators (captured through the `dex_aggregator.trades` [14] table) are dropped.
3. **Cyclic Arbitrage & Profit Filter.** A transaction is kept only if it forms a sequence of at least two swaps, such that (i) the the token bought in the j-th swap is the same as the token sold in the $(j+1)$-th swap, (ii) the sequence begins and ends with the same token and (iii) yields a strictly non-negative balance change in every token, with at least one positive gain.

The first callee in any such profitable, cyclic arbitrage transaction is tagged as a *candidate contract*, indicating it likely belongs to a cyclic arbitrage bot. A more detailed and formal description of this classification process can be found in Appendix D of the full version [50].

Next, we inspect the candidate contracts once again to avoid misclassifications. The few misclassified contracts we find during this stage are mainly due to popular mislabeled routers or public utility contracts. The details of this step can be found in Appendix E of the full version [50]. Finally, the remaining contracts form our final set of cyclic arbitrage contracts $\mathcal{C}_{\text{bot}}$.

4.2 Transaction-Level Classification

We classify each transaction in our study window (obtained through the respective table `{{chain}}.transactions` [15]) according to three dimensions.

Transaction purpose (cyclicArb / other). The transaction is marked CYCLICARB when its first callee (the address in the `to` field) is a member of the validated set of cyclic arbitrage contracts $\mathcal{C}_{\text{bot}}$; otherwise, it is marked OTHER. This choice ties the label to the entity that originates the on-chain action rather than to the presence of any particular pattern inside the trace.

DEX involvement (trade / interaction / residual). We scan swaps in `dex.trades` and consider the full call trace (`{{chain}}.traces`). If the transaction itself emits at least one swap event (e.g. `Swap` in Uniswap V2/V3 [2] or `TokenExchange` in Curve [1] as a result of function calls such as Uniswap V2's `swapExactTokensForTokens`, Uniswap V3 pool's `swap` method [2], or a Curve pool's `exchange` function [1]), we label it TRADE. When no swap is emitted but the trace touches a recognized DEX pool contract, typically through read-only calls such as `getReserves` or `slot0`, the label assigned is INTERACTION. (We use the table `dex.raw_pools` to detect the existence of subcalls interacting with DEX-related contracts.) All remaining transactions, which never enter a DEX contract, are labelled RESIDUAL. The three options are mutually exclusive and collectively exhaustive.

Execution outcome (success / revert). Finally, this dimension records the final on-chain status of the transaction, based on its receipt. The transaction is marked as SUCCESS when execution was successful (`status`=1) and REVERT otherwise (`status`=0). A transaction classified as a TRADE cannot subsequently be classified as a REVERT, as only transactions that do not revert can perform trades.

These three categories give each transaction a concise profile, which we use to group transactions for the analysis that follows. For example, we write CYCLICARB – INTERACTION to mean the subset of transactions that invoke a cyclic arbitrage contract as the first callee and also probe a DEX contract.[2] OTHER – REVERT defines the set of all reverted transactions that have a callee outside of the set $\mathcal{C}_{\text{bot}}$. Comparing such sets of transactions over time and across roll-ups enables us to quantify the prevalence of optimistic MEV, success rates, and failure modes under different Layer 2 design choices, such as private mempools, sequencer ordering, low fees, and sub-second blocks.

5 Optimistic MEV Landscape

We commence our analysis of optimistic MEV by providing a broad overview of the landscape. Our analysis focuses on the period from August 2023 to May 2025 and looks at three Layer 2 networks (i.e., Arbitrum, Base, and Optimism) in comparison to the Ethereum Layer 1.

[2] One thing to note is that CYCLICARB – INTERACTION contains transactions from CYCLICARB – INTERACTION – SUCCESS and CYCLICARB – INTERACTION – REVERT.

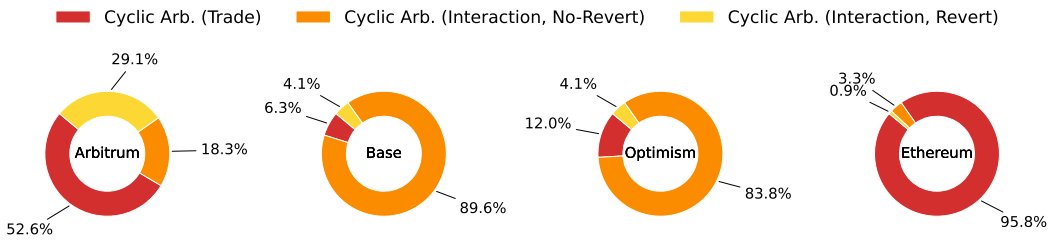


Figure 2 Distribution of cyclic arbitrage transaction outcomes across networks. Red indicates transactions from cyclic arbitrage contracts that execute trades, while the yellow and orange segments represent transactions that likely attempt arbitrage but do not execute any trade, either terminating without swaps (orange) or reverting entirely (yellow).

5.1 Execution Outcomes and Success Rates

Recall that the data classification identified cyclic arbitrage contracts. The following analysis shows that on Layer 2 networks, these contracts operate predominantly in an optimistic manner, exhibiting the speculative on-chain behavior characteristic of optimistic MEV, in contrast to their counterparts on Ethereum Layer 1.

Figure 2 illustrates the relative share of transactions from identified cyclic arbitrage contracts that execute swaps (red) compared to those that only probe DEX pools and terminate without a trade (orange) or revert (yellow), considering only the subset of transactions that interact with a DEX. The difference between networks is pronounced. On Base and Optimism, only 6.3% and 12% of such transactions, respectively, result in a successful trade, whereas on Arbitrum the share rises to 52.6% and on Ethereum Layer 1 to 95.8%. This distribution highlights a fundamental contrast: on Layer 1, cyclic arbitrage transactions appear largely precomputed and succeed overwhelmingly, while on Layer 2s, and particularly on OP-stack rollups, they display a speculative character with most attempts ending without executing a trade. This pattern of profitability being resolved only at execution and the prevalence of transactions that terminate without trading is exactly what defines optimistic MEV.

5.2 Prevalence Across Networks

We explore these differences further in Figure 3, showing the evolution of daily gas usage on Arbitrum, Base, and Optimism, broken down by activity from addresses performing CYCLICARB MEV and all OTHER activity. Each category is further subdivided into TRADE (transactions with executed token swaps), INTERACTION (on-chain probing or contract calls without token transfers), and RESIDUAL activity (see Section 4.2). Recall that the INTERACTION category includes both probing transactions that never execute a swap as well as reverted transactions. Notably, the share of gas consumed by CYCLICARB MEV (yellow, orange, and red), which on Layer 2 networks manifests predominantly as optimistic MEV, is very large on Base and Optimism in particular. The share of gas attributed to CYCLICARB - INTERACTION transactions rises sharply over time, indicating a surge in speculative, non-token-transfer, DEX contract probing activity associated with optimistic MEV. On Base, this category becomes a dominant contributor to gas usage by early 2025, accounting for 48% of total gas in the first quarter of the year. Overall, these optimistic MEV bots performing cyclic arbitrage are responsible for 51% of Base's gas usage during this period. Further, gas usage from an optimistic MEV bot on Base exceeds the entire block capacity of Ethereum

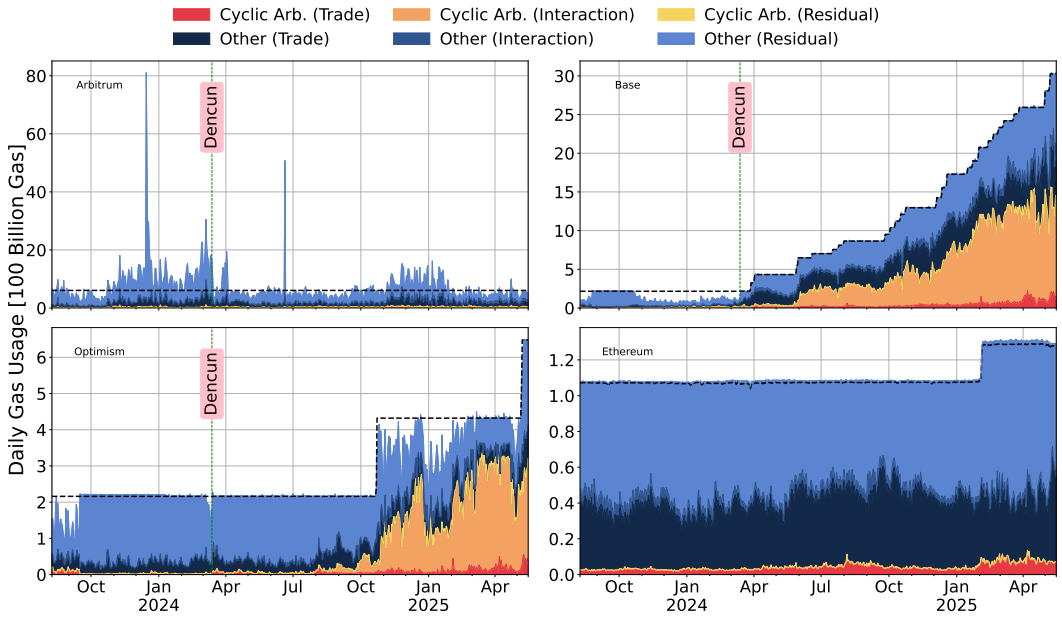


Figure 3 Daily gas usage by transaction category on Arbitrum, Base, Optimism, and Ethereum Layer 1. Each area shows gas consumed by cyclic arbitrage MEV bots (CYCLICARB) and all other activity (OTHER), subdivided into TRADE (transactions with executed swaps), INTERACTION (on-chain probing without swaps), and RESIDUAL segments (see Section 4.2). The vertical green dotted line marks the Dencun upgrade (EIP-4844), while the black dashed line indicates each network's target gas limit ("Gas Target" on Base, Optimism, and Ethereum; "Speed Limit" on Arbitrum).

Layer 1 by an order of magnitude. Optimism exhibits a similar trend, though with a slightly delayed onset. By the first quarter of 2025, optimistic MEV bots performing cyclic arbitrage account for 55% of gas usage on Optimism, with CYCLICARB - INTERACTION transactions specifically contributing 52%. In contrast, on Arbitrum and Ethereum Layer 1, CYCLICARB MEV consumes a much smaller fraction of available blockspace.

Turning to Figure 4, where we show the relative change in daily gas usage on Arbitrum, Base, Optimism, and Ethereum Layer 1, we focus on the relative usage of the respective categories. Observe that on Base and Optimism CYCLICARB - INTERACTION activity surges dramatically after Dencun, quickly outpacing executed arbitrages and consuming the majority of newly available blockspace in 2025. In contrast to the two other Layer 2s, Arbitrum shows no comparable increase in CYCLICARB - INTERACTION gas usage, suggesting differing strategic dynamics or protocol-level constraints (e.g., transaction ordering, fees, or shorter interblock times) that may discourage such speculative behavior. Note that Arbitrum displays the largest proportion of OTHER - INTERACTION gas consumption (see Appendix F of the full version [50]). This gas usage is driven by multiple contracts, the largest of which is the Uniswap V3 Liquidity Position Manager, but we observe no evidence of cyclic arbitrage from contracts in this category. Finally, the trade share from probing on the Ethereum Layer 1 remains flat, with trades by non-MEV actors dominant.

Importantly, Figures 3 and 4 display the categorization without taking the last factor (see SUCCESS / REVERT in Section 4.2) in consideration. Since optimistic MEV searchers often fail to capture arbitrage opportunities, one might expect their transactions to have a high revert rate. However, as shown in Appendix B of the full version [50], the gas consumed by reverted

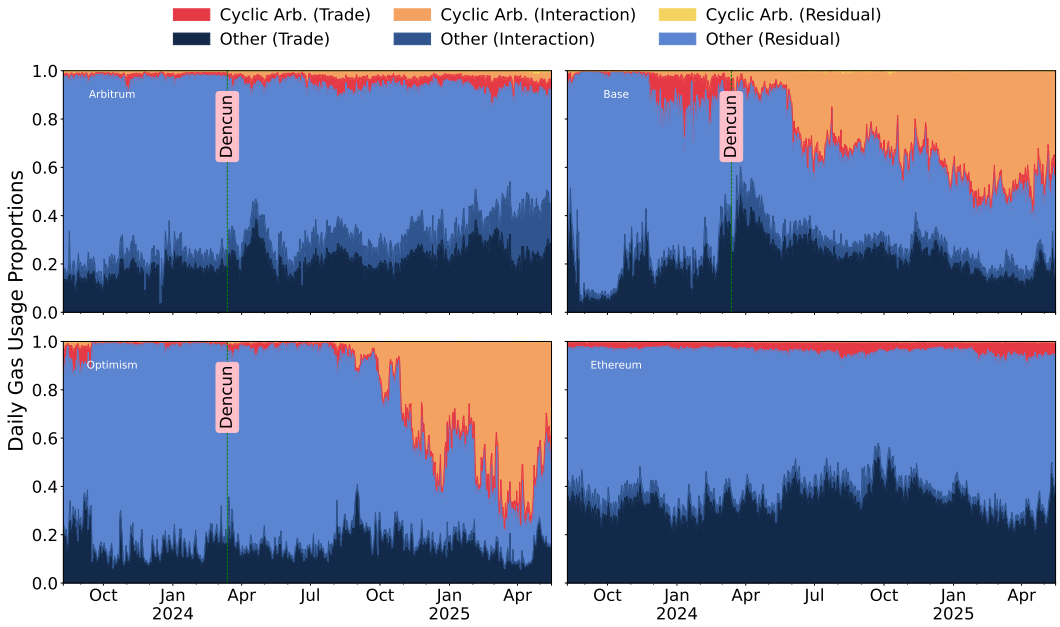


Figure 4 Relative daily gas usage by transaction category on Arbitrum, Base, Optimism, and Ethereum Layer 1. Each area shows gas consumed by cyclic arbitrage MEV bots (CYCLICARB) and all other activity (OTHER), subdivided into TRADE (transactions with executed swaps), INTERACTION (on-chain probing without swaps), and RESIDUAL segments (see Section 4.2). The vertical green dotted line marks the Dencun upgrade (EIP-4844).

optimistic MEV transactions is proportionally smaller on Base and Optimism than their share of total gas usage. Thus, on Base and Optimism failed MEV attempts by CYCLICARB bots primarily succeed but do not perform trades (i.e., CYCLICARB - INTERACTION - SUCCESS). By contrast, on Arbitrum and Ethereum Layer 1, a substantial share of MEV-related gas goes to reverted transactions, reflecting a higher rate of outright failures (i.e., CYCLICARB - INTERACTION - REVERT).

Next, we discuss possible factors driving the distinct behavior of optimistic MEV searchers on OP-Stack rollups versus Arbitrum and Ethereum Layer 1. First, Arbitrum's FCFS transaction ordering, especially before the Timeboost mechanism,[3] turns MEV extraction into a latency race, disincentivizing on-chain speculative probing. For example, with FCFS ordering, an optimistic MEV bot cannot specify a low-priority fee to sit at the end of the block and capture price differences after other transactions execute. Instead, it must carefully time its transaction submission, resulting in less control over its block position and over the number of possible preceding transactions that create exploitable price differences.

Second, CEX-DEX arbitrage profits are known to scale with the square root of the mean interblock time [37, 40]. Given that Arbitrum has shorter block times (approximately 250ms) compared to Base and Optimism (both 2s), the expected profits from CEX-DEX arbitrage are lower on Arbitrum. Conversely, the higher expected profits on Base and Optimism from CEX-DEX arbitrage may also elevate profitability in DEX-DEX arbitrage, particularly for pairs not actively traded on centralized exchanges. Once CEX-DEX arbitrage opportunities

[3] TimeBoost, introduced on Arbitrum on April 17, 2025, adds a sealed-bid second-price auction "express lane" that allows users to submit transactions directly to the sequencer for prioritized inclusion [7].

are closed, residual price discrepancies can remain between DEX pairs, creating additional cyclic arbitrage opportunities within the same Layer 2. Thus, lower block times might discourage excessive transaction spam.

Finally, block capacity and transaction fees also appear to shape the prevalence of optimistic MEV. In Figure 3, we see that immediately following the Dencun hardfork, Base increased its gas target by roughly an order of magnitude (black dashed line), and Optimism similarly raised its gas target. In contrast, Arbitrum's speed limit (the equivalent throughput cap) remained unchanged throughout our measurement window.

We conclude by highlighting that in contrast to all three Layer 2s, Ethereum Layer 1 shows almost no significant gas usage in the CYCLICARB – INTERACTION category. Instead, gas usage related to DEXes remains dominated by OTHER – TRADE activity. Activities resembling speculative probing, so prominent on Layer 2s, are largely negligible on Layer 1, reinforcing the fundamental distinction in MEV dynamics between the two environments. Overall, these findings support our core premise: optimistic MEV strategies are uniquely fostered by the low-cost, high-throughput conditions of Layer 2 architectures.

5.3 Impact of Dencun on Growth

To examine this further, Figure 5 shows the relative change in daily gas consumption for three categories (i.e., CYCLICARB, OTHER, and total activity (CYCLICARB+OTHER)) on Base (left) and Optimism (right), each normalized to 1 on the day of the Dencun upgrade. In both networks, cyclic arbitrage MEV bots exhibit the largest increase, indicating that the additional block capacity is largely consumed by optimistic MEV activity. Notably, on Optimism, there is no consistent rise in gas usage by other transactions despite the higher gas limit. This suggests optimistic MEV is the main beneficiary of the extra blockspace, with most new capacity used for on-chain computations that could be done off-chain.

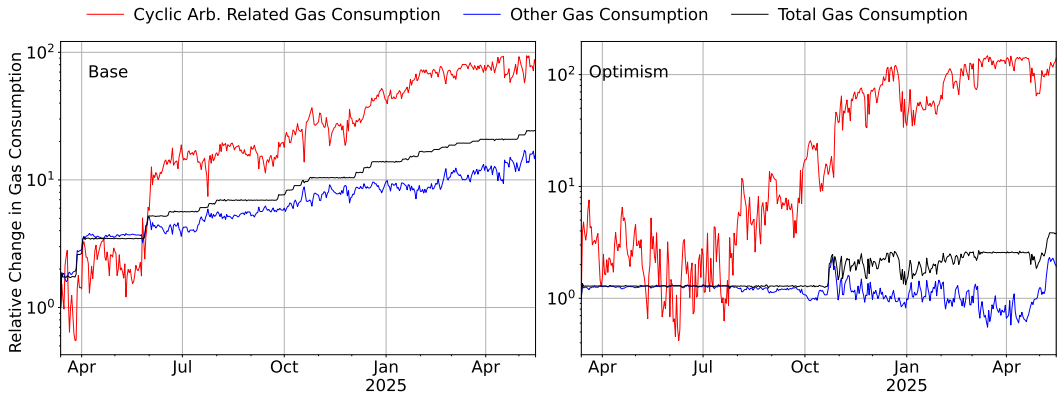


Figure 5 Relative change in daily gas consumption for three categories – cyclic arbitrage related, other, and total – on Base and Optimism, normalized to 1 on the day of Dencun hardfork (i.e., 13 March 2024). Each curve shows how gas usage in each category evolves post-Dencun, highlighting the divergent growth rates of optimistic MEV activity (CYCLICARB) versus all other on-chain activity.

5.4 Fee Dynamics

Turning to transaction fees, Figure 6 plots the daily median gas prices paid by optimistic MEV bots on each Layer 2. Notably, Arbitrum enforces a floor of 0.01 GWei on its base fee when demand falls below its speed limit [6]. In contrast, Base and Optimism follow the

EIP-1559 model, allowing fees to decline arbitrarily when demand is under the gas target. This difference explains why Base and Optimism blocks appear consistently full (see Figure 3) in comparison to target utilization, whereas Arbitrum's blocks often show unused headroom.

Median gas prices paid by optimistic MEV bot transactions (i.e., those labeled as CYCLICARB MEV) starting from Dencun, further underscore these distinctions: Optimism sees the lowest median price (0.0005 GWei), Base sits in the middle (0.0061 GWei), and Arbitrum commands the highest median (0.01 GWei). Thus, the median price paid by optimistic MEV bots on Optimism is a factor of 20 lower than on Arbitrum, while it is nearly a factor of 2 lower on Base than on Arbitrum.[4] Lower fees on Base and Optimism reduce the cost of speculative probing, making optimistic MEV more profitable, while Arbitrum's relatively higher fee floor dampens such behavior. Thus, a final factor driving optimistic MEV is the comparatively low fees on Base and Optimism. These reduced costs make speculative probing more profitable on those networks, whereas Arbitrum's higher fee floor suppresses such behavior.

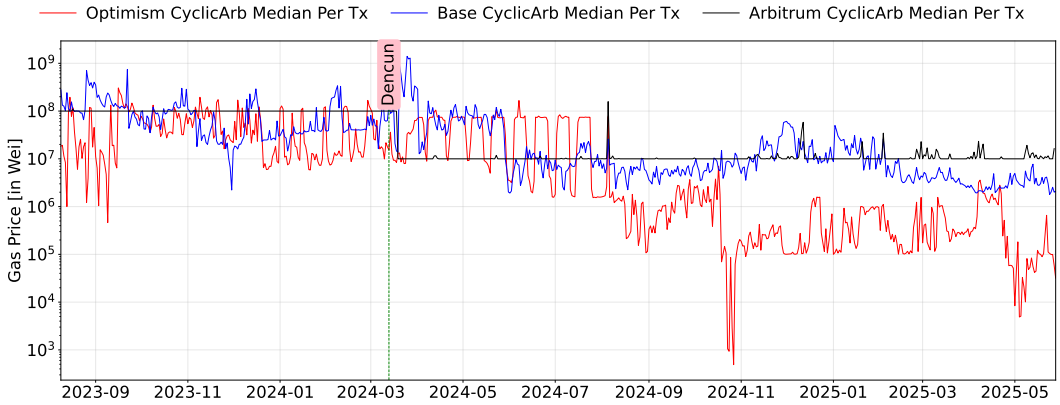


Figure 6 Median gas price paid by optimistic MEV bots (i.e., CYCLICARB MEV) on Optimism, Base, and Arbitrum. For Base and Optimism, the plot show `gas_price` which includes `base_fee` + `priority_fee` and for Arbitrum, it shows `effective_gas_price`.

Importantly, even though transactions from optimistic MEV searchers on Base and Optimism account for more than half the gas usage in the first quarter of 2025, they only account for 23% and 17% of the transaction fees paid, respectively. Thus, there is a disconnect between the gas used and the fees paid. Optimistic MEV transactions pay less for their transactions.

Next, we examine this fee discrepancy in more detail in Figure 7, which plots median and mean gas price per unit gas paid by optimistic MEV bots (i.e., those classified as CYCLICARB) versus all other transactions on Base and Optimism over time.

On Base, in Q1 2025, the average gas price per gas paid by optimistic MEV bots is 0.0209 GWei, compared with 0.0710 GWei for all other transactions – a factor of 3.5 difference. When we consider the median instead of the mean, the gap narrows: MEV bots pay a median of 0.0047 GWei, while the rest pay 0.0057 GWei. Notably, between April and September

[4] One reason the gap between Base and Arbitrum is smaller is that there is a short period where fees on Base are higher than those on Arbitrum. Between late November 2024 and early January 2025, we see the gas price paid by optimistic MEV bots on Base exceeding Arbitrum. The reason for this is the congestion in the network during those periods caused by the sniper bots (see Appendix B of the full version [50]).

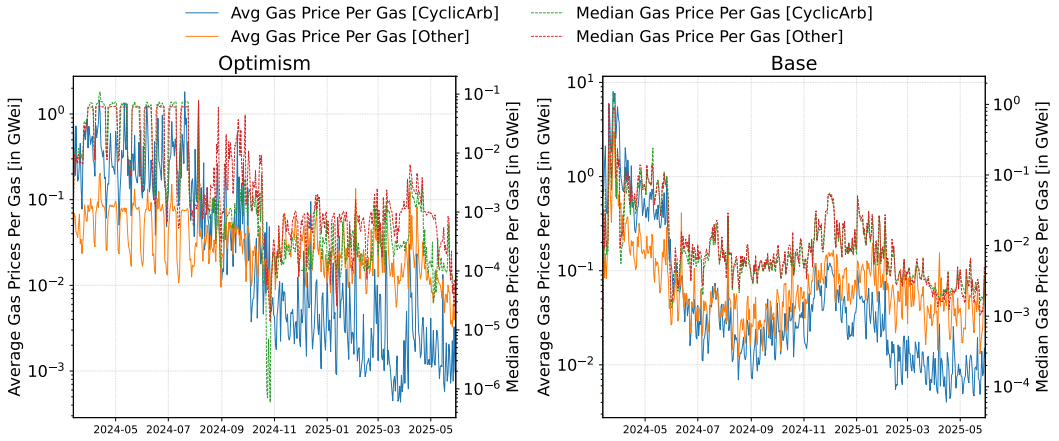


Figure 7 Gas prices per gas unit from optimistic MEV bots (i.e., CYCLICARB MEV) versus other actors on Base (right) and Optimism (left) starting from the day of Dencun hardfork (i.e., 13 March 2024). Solid lines show the daily average gas price per gas unit paid by CYCLICARB MEV bots (blue) and other actors (orange), while dashed lines show the corresponding median fees. Both average and median gas prices paid per gas unit by CYCLICARB MEV bots are lower than those of others.

2024, there was a period when MEV bot transactions actually paid higher average fees than other users. This shows that, although optimistic MEV bots tend to pay less overall, the difference is smaller and more nuanced than one might expect.

On Optimism, the pattern is similar. In Q1 2025, the average gas price per gas for MEV bots is 0.0042 GWei, versus 0.0252 GWei for other actors – a factor of 5 difference. Median gas prices are 0.0003 GWei for MEV bots and 0.0007 GWei for the rest. Again, the smaller median gap and occasional fee spikes by MEV bots in mid-2024 confirm that on-chain probing does not always seek the absolute cheapest gas. Nonetheless, the high prevalence of optimistic MEV correlates with periods of low gas prices after the Dencun hardfork.

5.5 Characteristics of the Largest Cyclic Arbitrage Bots

We now turn our attention to the behavior and structure of the most active CYCLICARB bots on each network. Specifically, we analyze the top 10 bots by gas usage on Base, Optimism, and Arbitrum, focusing on execution outcomes, code similarity, and operational architecture.

The statistics relevant to the top 10 bots (based on gas usage) are presented in the following tables for Base (Table 1a), Optimism (Table 1b), and Arbitrum (Table 1c). The primary metrics in the tables are defined as follows:

Swaps. The total number of individual swap operations executed as a result of calls to the given MEV bot contract. Notably, a single on-chain transaction may include multiple distinct swap operations.

Transactions With Trades. The total number of unique transactions that called the MEV bot contract and included one or more swap operations. It counts transactions rather than individual swaps.

Non-Reverted Transactions. The total number of transactions that called the MEV bot contract and were successfully executed and committed on-chain.

Reverted Transactions. The total number of transactions that called the MEV bot contract but failed during execution (i.e., were reverted), and therefore did not result in the intended state changes, although they still consumed gas.

Transaction per Unique Calldata. The ratio total number of transactions by the bot to the unique calldata entries used, where a value of 10 means that, on average, each calldata entry is reused across 10 transactions.
Median Calldata Length. The median length of the calldata used by the bot (in hexadecimal characters).
Cumulative MEV Bot Gas (%). The cumulative percentage of total transaction gas consumed by MEV bot contracts. The percentage is calculated relative to the total gas used by a defined set of contracts (i.e., all identified MEV bots within the dataset $\mathcal{C}'_{bots,chain}$) and is rounded to two decimal places.

Execution Outcomes and Bot Behavior Across Networks

A principal finding of our analysis concerns the execution outcomes of cyclic arbitrage attempts by MEV bots. Across all three Layer 2 networks, i.e., Base, Optimism, and Arbitrum, we observe that a large proportion of cyclic arbitrage attempts do not result in profitable outcomes as they do not even execute a swap. However, the way unsuccessful attempts are handled differs substantially between networks.

On OP-Stack-based networks (Base and Optimism), many unprofitable MEV transactions still conclude successfully from the perspective of the *Ethereum Virtual Machine (EVM)*; that is, they do not revert (on Optimism revert rate is 0.01%, on Base 0.005%), even though they fail to execute a profitable arbitrage. These transactions typically make no substantive state changes apart from incurring transaction fees. This behavior implies that the primary disincentive for speculative probing on these chains is economic (i.e., fees), not technical failure. Despite their technical success, our analysis shows that the actual success rate (defined as executing an arbitrage) is exceptionally low among the top contracts on Base (0.58%) and Optimism (1.49%). This indicates a high volume of ultimately unsuccessful, speculative execution.

In contrast, Arbitrum displays a different pattern. When top cyclic arbitrage bot transactions do not revert, they almost always yield successful arbitrage. There, the success rate of non-reverted transactions is 77%, while the overall success rate (i.e., including those transactions that revert) is 56%. This suggests a more selective or precise execution strategy. However, this high success rate for non-reverted transactions is accompanied by a significantly higher incidence of reverts, relative to Base and Optimism. Thus, Arbitrum-based bots appear to favor failing fast (via reverts) over speculative probing. These findings were further validated through cross-referencing with a curated dataset prepared by Entropy Advisors, an official partner of the Arbitrum Foundation [19, 18], strengthening our confidence in the observed execution dynamics across networks.

Further, our analysis of the largest cyclic arbitrage bots demonstrates again that they predominantly operate as optimistic MEV bots. To demonstrate this, we compute the number of transactions per unique calldata entry. On Base, the largest cyclic arbitrage bot reuses the same calldata more than 100 times on average, while on Optimism, one bot repeats identical calldata over 7 million times. Across both Base and Optimism, all major cyclic arbitrage contracts exhibit extensive calldata reuse, reflecting a speculative approach where little specific information such as exact paths or trade amounts is precomputed off-chain. This is indicative of optimistic MEV. Examining the calldata itself reinforces this: in many cases it is non-informative, as indicated by short median calldata lengths, or consists primarily of lists of DEXes to probe for potential opportunities.

When we turn to Arbitrum, the picture is somewhat different. While the majority of the largest cyclic arbitrage contracts exhibit extensive calldata reuse, several do not. In particular, the third, fourth, and fifth largest contracts show minimal reuse, resembling the pattern

Table 1 This table reports statistics for the top 10 cyclic arbitrage MEV bot contracts (ranked by cumulative gas usage) on Base (Table 1a), Optimism (Table 1b), and Arbitrum (Table 1c). For each contract, we report: the total number of individual swap operations executed (swaps), the number of transactions that included at least one swap (txs with trades), the number of transactions that executed without reverting (non-reverted txs), and the number of transactions that reverted during execution (reverted txs). The column txs per unique calldata gives the ratio of total transactions to the number of unique calldata entries used by the bot and median calldata length reports the median size of calldata (in hexadecimal characters) observed for the bot. The last column (cum. MEV bot gas (%)) indicates the cumulative share of gas consumed by each contract relative to all identified MEV bots on that chain.

(a) Base.

contract	swaps	txs with trades	non-reverted txs	reverted txs	txs per unique calldata	median calldata length	cum. MEV bot gas (%)
0xf5ff...a8c0	175,301	60,315	23,173,896	1	107.08	3830	6.71
0xbff6...73e9	411,376	152,046	24,379,515	3700	90.95	2977	11.76
0xaa87...fb84	128,100	65,074	14,680,490	535	16.65	2722	15.16
0xddfa...abd3	42,715	15,948	9,032,099	84	3.14	1255	17.90
0xdade...6084	148,186	51,864	5,338,322	28	145.06	6	19.42
0xe91c...96af	56,531	18,637	6,880,362	379	70.72	1330	20.50
0x4e85...0cea	125,575	45,586	1,765,057	25	44.43	2	21.56
0x2b24...446f	65,548	22,627	3,542,945	9	129.45	6	22.58
0x826f...b5f5	343,999	106,016	5,009,861	3	1.51	2465	23.58
0xbba9...3cf9	68,352	25,976	2,737,139	14	76.01	2	24.56

(b) Optimism.

contract	swaps	txs with trades	non-reverted txs	reverted txs	txs per unique calldata	median calldata length	cum. MEV bot gas (%)
0x8872...6324	676,753	156,634	11,966,203	0	498.11	6	33.06
0xabf4...017c	29,586	8251	7,979,548	0	7,979,548.00	10	41.94
0xcdcc...5c8f	247,308	68,040	6,382,847	0	3489.80	6	49.08
0xd3dc...f2ac	214,741	60,522	6,901,973	5	1674.42	6	55.96
0x4d43...987f	208,002	47,459	2,322,899	10	195.39	3213	62.34
0x3955...74c2	422,807	107,201	1,992,783	0	2.98	2	64.56
0x9d1b...f96f	37,920	9531	825,165	40	1650.41	6	66.62
0x0daf...a7dd	398,174	105,069	2,299,930	0	4.72	2	68.51
0x2642...39e6	73,322	26,656	1,387,723	242	775.83	2422	69.98
0xf261...c883	242,058	51,232	928,822	4004	100.14	6	71.44

(c) Arbitrum.

contract	swaps	txs with trades	non-reverted txs	reverted txs	txs per unique calldata	median calldata length	cum. MEV bot gas (%)
0x0000...98bd	9,871,348	3,338,292	3,338,353	1,241,384	17.34	110	10.42
0x60ca...294b	5,106,002	1,836,869	4,355,290	26,968	1.12	1207	18.68
0x6893...151c	4,769,221	1,706,662	1,708,167	1,087,474	1.00	572	24.28
0x9e52...5867	3,055,148	1,065,722	1,067,478	1,317,339	1.00	336	29.22
0xa9ff...d82f	774,311	347,885	348,021	51,732	1.05	2353	32.67
0x0000...2e43	3,045,007	1,125,161	1,125,162	338,202	14.52	104	35.34
0x0000...fb51	1,527,676	502,973	503,320	290,686	1.95	163	37.53
0xf238...e145	320,724	116,712	585,063	2796	2.39	266	39.28
0xe98b...d87c	1,145,798	560,741	516,379	413,451	30.23	52	40.87
0x84f1...b7cf	522,016	248,810	252,743	251,681	17.31	46	42.46

observed on Ethereum, where none of the major cyclic arbitrage bots have a transactions per unique calldata ratio higher than 1.006. Despite this, their transactions revert frequently, around 50%, suggesting they are not speculative in the sense of performing computation on-chain but are instead exposed to significant uncertainty about whether their transactions will land as intended. The remaining large cyclic arbitrage bots on Arbitrum, however, display frequent calldata reuse, consistent with the optimistic MEV pattern observed on Base and Optimism.

Concentration of Gas Usage

We also compared the concentration of MEV bot activity across networks in terms of gas usage. On Optimism, the top 10 MEV bots account for a disproportionately large share of gas consumed by atomic arbitrage, suggesting a more concentrated and potentially less competitive landscape. Arbitrum shows a slightly more distributed gas usage among top bots, while Base has the most diffuse distribution. These trends align with recent shifts in dominance across Layer 2s, with Base emerging as the most active MEV network, followed by Arbitrum and then Optimism.

Code Similarity and Cloning Behavior

To explore potential relationships among MEV-performing contracts, we conducted a bytecode similarity analysis inspired by prior work on Ethereum contract topology by Kiffer et al. [32]. This revealed meaningful differences in code reuse and cloning behavior across networks.

Contracts deployed on Optimism and Base show a moderate degree of bytecode similarity, suggesting some shared tooling or deployment patterns. In contrast, cyclic arbitrage contracts on Arbitrum appear more structurally distinct, with fewer instances of directly shared or closely related implementations. Among the three networks, Optimism exhibits the highest level of intra-network code similarity, pointing to a relatively concentrated MEV ecosystem with repeated use of identical or near-identical contracts – potentially indicating lower competitive diversity.

We also identified clear examples of contract cloning behavior, particularly on Base (see Appendix A of the full version [50]). For instance, contracts deployed at addresses `0x2b24...446f` and `0xdade...6084` share identical bytecode not only with each other but also with 48 additional contracts. All 50 of these contracts engage in cyclic arbitrage and are classified as MEV bots in our dataset [8]. Although none of these clones individually ranked among the top 10 gas consumers, their collective presence underscores the widespread use of standardized MEV bot implementations. This pattern may reflect coordinated deployments by a single actor or group.

Further analysis of these cloned contracts reveals another striking pattern: the transaction `calldata` passed to them is often minimal or non-informative (e.g., `0x0001`), suggesting that key arbitrage parameters are not set off-chain. As an example, `0xdade...6084` executed 51,864 transactions with trades of which 50,861 had `0x0001` as calldata. Instead, the contract logic itself likely handles opportunity identification, path selection, and trade sizing through real-time on-chain computation. This architecture relies heavily on internal heuristics and on-chain state queries (e.g., via `STATICCALL`s), contrasting with more traditional MEV bots where critical parameters are calculated off-chain and passed in via calldata.

6 Drivers of Optimistic MEV

To move beyond descriptive patterns and develop a quantitative understanding of the drivers of MEV activity on Layer 2 networks, we employ an OLS regression framework. The goal is to assess the statistical significance and direction of influence of various market and network-level metrics on daily MEV fluctuations. By systematically analyzing these relationships, we aim to identify the conditions under which MEV activity is amplified or suppressed across evolving Layer 2 ecosystems.

Our analysis focuses on two dependent variables: the daily change in the total number of transactions associated with CYCLICARB bots ($\Delta\texttt{CyclicArbTx}_t$) and the subset of those transactions that execute trades ($\Delta\texttt{CyclicArbTxWTrade}_t$), described in detail below:

ΔCyclicArbTx$_t$ (Change in Cyclic Arbitrage Transaction Count) The total number of transactions classified as CYCLICARB (see Section 4.2) within our dataset is measured by this variable.

ΔCyclicArbTxWTrade$_t$ (Change in Cyclic Arbitrage Trade Transaction Count) The daily change in the total number of transactions that are classified as CYCLICARB-TRADE.

This dual perspective allows us to distinguish between general CYCLICARB activity and the subset of transactions involving explicit on-chain value extraction through token trades, CYCLICARB-TRADE. Understanding these dynamics is crucial for assessing the efficiency of Layer 2 markets and informing the design of potential mitigation strategies.

The regression equation is then specified as:

$$\begin{aligned}\Delta y_t = \beta_0 &+ \beta_1 \cdot \Delta\text{Price}_t + \beta_2 \cdot \Delta\text{Volatility}_t + \beta_3 \cdot \Delta\text{RetailTxs}_t \\ &+ \beta_4 \cdot \Delta\text{RetailAggFrac}_t + \epsilon_t\end{aligned}$$

where $\Delta x_t = x_t - x_{t-1}$ denotes the first difference of variable x at time t, and ϵ_t is the error term. The independent variables are defined as follows:

ΔPrice$_t$ (Change in ETH Price) The daily change in the price of ETH (denominated in US$).

ΔVolatility$_t$ (Change in ETH Volatility) The daily change in intraday volatility of ETH. Intraday volatility is computed using the Garman-Klass estimator [25, 34], defined for a given day as:

$$\sigma_{\text{OHLC}} = \sqrt{0.5 \cdot (\ln H - \ln L)^2 - (2\ln 2 - 1) \cdot (\ln C - \ln O)^2}$$

where O, H, L, C are the Open, High, Low, and Close prices for ETH within the day, respectively.

ΔRetailTxs$_t$ (Change in Retail Trade Count) The daily change in the total number of on-chain trades – defined as transactions involving at least one swap – initiated by entities not classified as CYCLICARB bots in our curated dataset, i.e., OTHER-TRADE.

ΔRetailAggFrac$_t$ (Change in Retail Aggregator Usage) The daily change in the fraction of trades by non-cyclic arbitrage bot addresses that are routed through DEX aggregators:

$$\text{RetailAggFrac}_t = \frac{\text{RetailAggregatorTrades}_t}{\text{RetailTxs}_t}$$

A lower value indicates a greater share of direct-to-pool trades, which may reflect routing inefficiencies and create arbitrage opportunities [5, 53, 52]. However, not all direct trades are necessarily exploitable, some may avoid introducing new arbitrage opportunities and thus remain economically efficient, even without using an aggregator.

This metric excludes trades routed via standard Uniswap routers, which differ in function and behavior from third-party DEX aggregators [3, 36, 24, 57, 58].

We now turn to the empirical results of our regression analysis, presented in Table 2. The models examine how the previously defined variables relate to daily fluctuations in cyclicArb activity across Layer 2 networks. Specifically, we report results for two dependent variables: the daily change in overall cyclicArb-related transactions (`ΔCyclicArbTx`) and the daily change in cyclicArb-related transactions involving DEX trades (`ΔCyclicArbTxWTrade`).

Table 2 OLS regression results for Base, Optimism, and Arbitrum. The dependent variables are the daily change in cyclicArb-related transactions (ΔCyclicArbTx) and the daily change in cyclicArb-related transactions involving DEX trades (ΔCyclicArbTxWTrade). Independent variables include changes in ETH price, intraday volatility, the count of non-cyclicArb (other, see Section 4.2) trades, and the fraction of those routed via DEX aggregators. Robust standard errors are reported in parentheses. Significance levels: $^{*}p < 0.1$, $^{**}p < 0.05$, $^{***}p < 0.01$.

	ΔCyclicArbTx (1, Base)	ΔCyclicArbTxWTrade (2, Base)	ΔCyclicArbTx (1, Optimism)	ΔCyclicArbTxWTrade (2, Optimism)	ΔCyclicArbTx (1, Arbitrum)	ΔCyclicArbTxWTrade (2, Arbitrum)
const	2445.7691	173.5872	372.2486	44.9657	82.2414	32.3800
	(6290.7125)	(582.2378)	(1066.6453)	(152.2307)	(437.4133)	(177.3206)
ΔPrice	-12608.7057	-1881.3146**	-1156.6053	-454.8404	-25.9810	-357.1149
	(10629.4760)	(897.7109)	(1421.6869)	(298.1963)	(706.7666)	(264.1559)
ΔVolatility	34112.7533***	8919.1607***	1323.4766	1289.5335***	2265.7813***	299.0799
	(11474.9164)	(913.0034)	(1722.3345)	(375.0010)	(707.9527)	(307.2508)
ΔRetailTxs	23175.6525**	4172.5237***	2956.2109	3276.7001***	16024.0033***	8097.8011***
	(10917.7060)	(1245.3633)	(1837.3891)	(532.7932)	(904.0916)	(360.9759)
ΔRetailAggFrac	12230.8890**	1883.1927***	-1785.4280	-1264.8155***	-1524.6186***	-1319.3665***
	(4756.1076)	(442.2920)	(1253.4087)	(369.1399)	(475.3803)	(289.9207)
Obs	654	654	700	700	700	700
Adj. R^2	0.0736	0.3399	0.0240	0.5818	0.7143	0.7898
F-stat	6.8461	31.7334	2.7295	57.8104	208.3965	265.6485

A consistent pattern across all three networks is that the model explains daily changes in **cyclicArb-trade transaction count** (Model 2) substantially better than changes in the broader **cyclicArb transaction count** (Model 1), as indicated by higher adjusted R^2 values. This suggests that the chosen market variables are more directly associated with cyclicArb behaviors involving on-chain value extraction (i.e., swaps, trade), rather than speculative probing.

Arbitrum exhibits the best overall model fit, particularly for `ΔCyclicArbTx` category. This likely reflects a tighter correlation between cyclicArb-trade transaction count and total cyclicArb transaction count on that chain.

Comparative Analysis of Independent Variables

ΔPrice (Daily Change in ETH Price). The effect of ETH price fluctuations on cyclicArb activity appears weak and inconsistent across networks:

- **Model 1:** No statistically significant association between daily ETH price changes and overall cyclicArb transaction count on any network.
- **Model 2:** A statistically significant negative effect is observed on Base (-1881.31, $p < 0.05$), suggesting that a sharp decrease in ETH price may increase the number of cyclicArb-trade executions. The coefficients for Optimism and Arbitrum are also negative but fall short of significance thresholds.

These findings suggest a higher prevalence of arbitrage opportunities on days marked by ETH price declines. We propose two primary hypotheses for this observation. Firstly, diminished market liquidity during price downturns [11] may lead to increased slippage and suboptimal trade execution, thereby fostering greater price discrepancies [4]. Secondly, the

predominance of long leverage in DeFi protocols [28] could result in a higher frequency of liquidations on days with negative price movements. Such liquidations can trigger significant price swings, which in turn may create further price disparities [46].

ΔVolatility (Daily Change in ETH Volatility). Volatility, by contrast, plays a more significant role:

- **Model 1:** A strong and significant positive effect on Base (34112.75, $p < 0.01$) and Arbitrum (2265.78, $p < 0.01$), indicating that volatility increases overall CYCLICARB transaction activity. No significant effect is observed on Optimism.
- **Model 2:** A similarly strong positive effect is found for CYCLICARB-TRADE transactions on Base (8919.16, $p < 0.01$) and Optimism (1289.53, $p < 0.01$). On Arbitrum, however, the coefficient is small and not statistically significant.

The widely held expectation that increased volatility fuels MEV activity is supported by our results for CYCLICARB-TRADE transactions on Base and Optimism [37, 31], as well as for overall CYCLICARB counts on Base and Arbitrum. However, the absence of a significant volatility effect on the change of CYCLICARB-TRADE on Arbitrum deserves special attention. In a high-throughput setting with very short block times, an efficient CEX–DEX arbitrage layer can continually realign each DEX price to its CEX counterpart, effectively erasing intra-DEX cyclic arbitrage opportunities, regardless of volatility. Under such a "hierarchical arbitrage" regime, volatility may drive cross-venue trades but leave purely on-chain cycles unprofitable. This provides a possible explanation for why we observe little volatility-driven CYCLICARB trading on Arbitrum.

ΔRetailTxs (Change in Retail Trade Count). Retail activity (used here as a proxy for organic user flow) is consistently important for MEV trade behavior:

- **Model 1:** Significant positive effects on Base (23175.65, $p < 0.05$) and Arbitrum (16024.00, $p < 0.01$); not significant on Optimism.
- **Model 2:** Strong and significant on all three networks (i.e., Base, Optimism, and Arbitrum) all at $p < 0.01$.

These findings reinforce the idea that user-driven flow is the foundation for MEV extraction via arbitrage. This is because user transactions, such as trades on DEXs, are the primary actions that perturb market prices. These perturbations create transient price discrepancies across different venues or asset pairs. CYCLICARB bots then capitalize on these temporary imbalances by executing arbitrage trades, effectively profiting from the price impact of the initial user-driven activity. Thus, without the initial flow from users, the opportunities for this form of MEV extraction would be significantly diminished. On the other hand, the lack of significance for total CYCLICARB transaction count on Optimism suggests that probing or spam-like transactions may be more prevalent there, reducing the signal from genuine trade-driven activity.

ΔRetailAggFrac (Change in Aggregator Usage by Retail Users). This variable reveals the most striking network divergence:

- **Model 1:** Base shows a positive and significant effect (12230.89, $p < 0.05$); Arbitrum, a significant negative effect (-1524.62, $p < 0.01$); Optimism shows no significance.
- **Model 2:** Base again shows a significant positive effect (1883.19, $p < 0.01$), while both Optimism (-1264.82, $p < 0.01$) and Arbitrum (-1319.37, $p < 0.01$) show significant negative effects.

Increased aggregator usage is generally indicative of more efficient transaction routing, which we hypothesized would lead to fewer price discrepancies. However, the Base results present a counterintuitive finding, as higher aggregator utilization did not correspond with a reduction in naive arbitrage opportunities. Several hypotheses may explain this observation:

- Many tokens on Base may have only one active pool, making direct-to-pool trades effectively MEV-optimal (i.e., they do not open up an arbitrage opportunity) – undermining the aggregator efficiency signal.
- Aggregator usage may correlate with increased activity in major token pairs (e.g., ETH-stables), especially during volatile periods, making `RetailAggFrac` a latent proxy for volatility.

Thus, our use of `RetailAggFrac` as a proxy for the ratio between MEV-optimal Trades and Total Trades, where MEV-optimal trades are those that do not introduce new arbitrage opportunities, may be misaligned on Base due to its unique market structure.

On Optimism and Arbitrum, the expected pattern emerges: increased aggregator usage correlates with reduced CYCLICARB-TRADE activity, consistent with the role of aggregators in mitigating simple arbitrage opportunities by improving trade routing.

Summary of Insights

The regression results demonstrate that some drivers of CYCLICARB (such as user trade count) are robust across networks. In contrast, other factors like volatility and aggregator usage show strong network-specific effects, reflecting differences in market structure and protocol design. These findings underscore that CYCLICARB dynamics are not uniform across Layer 2s, but are shaped by their unique configurations, including block times, mempool behavior, and execution environments.

7 Discussion

Optimistic MEV fits into the broader MEV landscape as a new variant of an old problem that was thought to have been largely solved. Its optimistic execution model is novel because profitability is determined on-chain during transaction execution, but the externality it creates in the form of large volumes of low-value or failed transactions mirrors and even outpaces the spam problem that plagued early Ethereum MEV. On Ethereum, that earlier spam arose from bots flooding the mempool with precomputed transactions and engaging in priority gas auctions without knowing whether their transactions would land in the right place in the block to secure the opportunity, a dynamic that private orderflow and MEV auctions were introduced to mitigate.

This behavior has significant consequences. It generates large volumes of optimistic MEV transactions on Layer 2s, creating spam that consumes blockspace without delivering corresponding value to users. This spam constitutes a negative externality for the network and must still be stored by nodes. The resulting data burden increases costs for infrastructure providers and risks concentrating node operation among better-resourced actors. It also appears that optimistic MEV creates a workload that very often fills up blocks, which should be taken into account when designing blockchains.

For searchers, this environment reshapes the underlying economics. The focus shifts away from high-certainty execution toward balancing the cost of repeated speculative probes compared to the profit they generate. Capturing value becomes less of a latency race; instead,

the transactions are submitted before the opportunities even exist. Further, optimistic MEV requires paying far lower priority fees compared to traditional MEV on Ethereum Layer 1, which fundamentally changes how bots compete and extract value.

MEV auctions are the main mechanism used in practice to prevent the negative externalities of MEV by reducing spam. Optimistic MEV makes this harder to apply. Its speculative, high-frequency nature could produce a flood of bids and overwhelm an auction with noise, while the short block intervals on Layer 2s leave little room to run the process at all. Revert protection, as implemented by Unichain, offers an alternative mitigation but risks generating a similar kind of spam directed at the sequencer. In addition, the presence of centralized sequencers on Layer 2s creates incentive dynamics very different from Layer 1.

Addressing optimistic MEV will likely require rethinking how, or whether, MEV auctions and related mechanisms can work in rollup environments. Our results suggest that network design choices can influence the prevalence of optimistic MEV: the significantly lower rate of optimistic MEV we observe on Arbitrum compared to OP-stack rollups indicates that factors such as gas pricing or block intervals can meaningfully affect these strategies. This points toward protocol-level levers for reducing optimistic MEV as Layer 2 designs evolve.

8 Outlook and Conclusion

In this work, we investigate the significant and large-scale impact of *optimistic MEV* on Layer 2 networks, focusing on cyclic arbitrage, which on these networks is predominantly executed in this optimistic form and accounts for more than 50% of gas usage in OP-Stack ecosystems. Our findings reveal that a confluence of factors, namely low transaction fees, extended interblock times, and PGA ordering mechanisms prevalent on OP-Stack chains, contributes to this extensive MEV activity.

The ramifications of this activity are substantial, manifesting as a phenomenon akin to network spam. This spam-like behavior inundates the network with low-value transactions, leading to inefficient resource allocation, wasted chain space, and potential degradation of user experience. Critically, this also limits the network's capacity to scale for higher-value or user-driven activity, as blockspace is increasingly consumed by speculative probing.

Addressing this challenge is essential for the long-term sustainability of these networks. A shift in network dynamics is necessary, either through the emergence of organic demand willing to pay higher fees, thereby crowding out low-value MEV attempts, or through direct interventions such as raising minimum transaction costs. Arbitrum, where different design choices have demonstrated some efficacy in curbing similar behaviors, serves as a pertinent case study for the latter approach.

Further differentiating the network dynamics, our analysis uncovered that most unsuccessful MEV attempts do not revert on OP-Stack Layer 2s, a stark contrast to Arbitrum, where such reversions are common. This divergence is likely attributable to Arbitrum's FCFS transaction ordering, approximately tenfold shorter interblock times, and a distinct fee market. These elements collectively appear to incentivize more deterministic behavior from MEV bots on Arbitrum, as evidenced when comparing the strategies of top bots across platforms, unlike the more speculative attempts observed on Optimism and Base. This underscores how architectural and fee-market designs can significantly influence the strategies of MEV bots and overall network congestion.

References

1 Curve finance github repository. https://github.com/curvefi. Accessed: 2025-05-19.

2 Uniswap github repository. https://github.com/Uniswap. Accessed: 2025-05-19.

3 0x534154. Uniswap bug causes loss of eth. https://hackmd.io/@0x534154/Hy6_66gT5, 2022. HackMD post, accessed 2025-05-09.

4 Austin Adams, Benjamin Y Chan, Sarit Markovich, and Xin Wan. Don't let mev slip: The costs of swapping on the uniswap protocol, 2024. arXiv:2309.13648.

5 Guillermo Angeris, Tarun Chitra, Alex Evans, and Stephen Boyd. Optimal routing for constant function market makers, 2022. arXiv:2204.05238.

6 Arbitrum. Gas fees: Child chain gas pricing. URL: https://docs.arbitrum.io/how-arbitrum-works/gas-fees#child-chain-gas-pricing.

7 Arbitrum Docs. Timeboost: A gentle introduction. https://docs.arbitrum.io/how-arbitrum-works/timeboost/gentle-introduction, April 2025. Accessed: 2025-06-17.

8 BaseScan. Find similar contracts: Contract 0x22009730d6ad8a90574ea826603906dc81766fb1. https://basescan.org/find-similar-contracts?a=0x22009730d6ad8a90574ea826603906dc81766fb1&m=exact, 2025. Accessed: May 21, 2025.

9 Vitalik Buterin. Possible futures of the ethereum protocol, part 2: The surge, 2024. Accessed on March 26, 2025. URL: https://vitalik.eth.limo/general/2024/10/17/futures2.html.

10 Sam Chepal. The hidden world of mev. https://www.samchepal.com/the-hidden-world-of/, 2023. Accessed: 2025-05-25.

11 Tarun Chordia, Avanidhar Subrahmanyam, and Richard W. Roll. Market liquidity and trading activity. *SSRN Electronic Journal*, July 2000. doi:10.2139/ssrn.237674.

12 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges, 2019. arXiv:1904.05234.

13 Dune Analytics. Address labels. https://docs.dune.com/data-catalog/curated/labels/address-labels, 2025. Accessed: 2025-05-25.

14 Dune Analytics. Dex aggregator trades. https://docs.dune.com/data-catalog/curated/evm/DEX/dex-aggregator-trades, 2025. Accessed: 2025-05-25.

15 Dune Analytics. Ethereum raw transactions. https://docs.dune.com/data-catalog/evm/ethereum/raw/transactions, 2025. Accessed: 2025-05-25.

16 Dune Docs. DEX Trades. https://docs.dune.com/data-catalog/curated/evm/DEX/dex-trades. Accessed: 2025-05-18.

17 EigenPhi. Eigenphi, 2025. Accessed: 2025-05-23. URL: https://eigenphi.io.

18 Entropy Advisors. Arbitrum mev arbitrage dashboard. https://dune.com/entropy_advisors/arbitrum-mev-arbitrage, 2025. Accessed: May 22, 2025.

19 Entropy Advisors. Entropy advisors. https://entropyadvisors.com/, 2025. Accessed: May 22, 2025.

20 Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In *Financial Cryptography and Data Security (FC), Kota Kinabalu, Sabah, Malaysia*, 2020.

21 Ethereum Improvement Proposals. EIP-4844: Shard Blob Transactions. https://eips.ethereum.org/EIPS/eip-4844, 2025. Accessed: 2025-05-08.

22 Ethereum Improvement Proposals. EIP-4844: Shard Blob Transactions (GitHub Markdown). https://github.com/ethereum/EIPs/blob/master/EIPS/eip-4844.md, 2025. Accessed: 2025-05-08.

23 Flashbots. mev-inspect-py: A tool for inspecting MEV transactions. https://github.com/flashbots/mev-inspect-py, 2025. Accessed: 2025-05-18.

24 foldfinance. Tweet posted 2022-10-09 20:43 utc. Tweet. URL: https://x.com/foldfinance/status/1579211031855108096.

25 Mark B. Garman and Michael J. Klass. On the estimation of security price volatilities from historical data. *The Journal of Business*, 53(1):67–78, 1980. URL: http://www.jstor.org/stable/2352358.

26 Krzysztof Gogol, Johnnatan Messias, Deborah Miori, Claudio Tessone, and Benjamin Livshits. Cross-rollup mev: Non-atomic arbitrage across l2 blockchains, 2024. arXiv:2406.02172.

27 growthepie. growthepie: Layer 2 analytics and education. Accessed on March 27, 2025. URL: https://www.growthepie.xyz.

28 Lioba Heimbach and Wenqian Huang. Defi leverage. *Available at SSRN*, May 2023. doi:10.2139/ssrn.4459384.

29 Lioba Heimbach, Lucianna Kiffer, Christof Ferreira Torres, and Roger Wattenhofer. Ethereum's Proposer-Builder Separation: Promises and Realities. In *ACM Internet Measurement Conference (IMC), Montreal, QC, Canada*, 2023.

30 Lioba Heimbach and Jason Milionis. The early days of the ethereum blob fee market and lessons learnt, 2025. doi:10.48550/arXiv.2502.12966.

31 Lioba Heimbach, Vabuk Pahari, and Eric Schertenleib. Non-atomic arbitrage in decentralized finance, 2024. doi:10.48550/arXiv.2401.01622.

32 Lucianna Kiffer, Dave Levin, and Alan Mislove. Analyzing ethereum's contract topology. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18, pages 494–499, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3278532.3278575.

33 L2BEAT. Layer 2 scaling: Total value secured. Accessed on March 27, 2025. URL: https://l2beat.com/scaling/tvs.

34 Young Li. A practical model for prediction of intraday volatility. https://assets.bbhub.io/professional/sites/10/intraday_volatility-3.pdf, 2021. Bloomberg Enterprise Quants, accessed 28 Apr. 2025.

35 LibMEV. Libmev, 2025. Accessed: 2025-05-23. URL: https://libmev.com.

36 MevRefund. "@uniswap what you doing here? poor man just wants to plow 0.36 eth into some shitcoin ...". Tweet. URL: https://x.com/MevRefund/status/1643048619547971584.

37 Jason Milionis, Ciamac C. Moallemi, and Tim Roughgarden. Automated market making and arbitrage profits in the presence of fees, 2023. arXiv:2305.14604.

38 Jason Milionis, Ciamac C. Moallemi, Tim Roughgarden, and Anthony Lee Zhang. Automated market making and loss-versus-rebalancing, 2024. arXiv:2208.06046.

39 Misaka. Mev research during wormhole incident. https://misaka.substack.com/p/mev-research-during-wormhole-incident, 2022. Accessed: 2025-05-25.

40 Alex Nezlobin and Martin Tassy. Loss-versus-rebalancing under deterministic and generalized block-times, 2025. arXiv:2505.05113.

41 Alexandre Obadia, Alejo Salles, Lakshman Sankar, Tarun Chitra, Vaibhav Chellani, and Philip Daian. Unity is strength: A formalization of cross-domain maximal extractable value, 2021. arXiv:2112.01472.

42 Burak Öz, Christof Ferreira Torres, Jonas Gebele, Filip Rezabek, Bruno Mazorra, and Florian Matthes. Pandora's box: Cross-chain arbitrages in the realm of blockchain interoperability. *arXiv preprint arXiv:2501.17335*, 2025. doi:10.48550/arXiv.2501.17335.

43 Ozan Solmaz. Optimistic MEV. https://github.com/kingbaldwin-iv/Optimistic-MEV, 2025.

44 Paradigm. Ethereum is a dark forest. https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest, 2020. Accessed: 2025-05-25.

45 Polynya. Ethereum l1 scaling roadmap, 2023. Accessed on March 26, 2025. URL: https://polynya.mirror.xyz/epju72rsymfB-JK52_uYI7HuhJ-W_zM735NdP7alkAQ.

46 Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. An empirical study of defi liquidations: incentives, risks, and instabilities. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, pages 336–350. ACM, November 2021. doi:10.1145/3487552.3487811.

47 Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest?, 2021. arXiv:2101.05511.

48 Rekt News. Return to the dark forest. https://rekt.news/return-to-the-dark-forest, 2021. Accessed: 2025-05-25.

49 Samczsun. Escaping the dark forest. https://samczsun.com/escaping-the-dark-forest/, 2020. Accessed: 2025-05-25.

50 Ozan Solmaz, Lioba Heimbach, Yann Vonlanthen, and Roger Wattenhofer. Optimistic MEV in Ethereum Layer 2s: Why Blockspace Is Always in Demand, 2025. doi:10.48550/arXiv.2506.14768.

51 Sorella Labs. Brontes: A blockchain analysis tool. https://github.com/SorellaLabs/brontes, 2025. Accessed: 2025-05-18.

52 ThogardPvP. Tweet posted 2022-10-09 20:28 utc. Tweet. URL: https://x.com/ThogardPvP/status/1579207229059715072.

53 ThogardPvP. Tweet posted 2022-10-09 20:35 utc. Tweet. URL: https://x.com/ThogardPvP/status/1579208974812909568.

54 Christof Ferreira Torres, Ramiro Camino, and Radu State. Frontrunner Jones and the Raiders of the Dark Forest: An Empirical Study of Frontrunning on the Ethereum Blockchain. In *30th USENIX Security Symposium, Virtual Event*, 2021.

55 Christof Ferreira Torres, Albin Mamuti, Ben Weintraub, Cristina Nita-Rotaru, and Shweta Shinde. Rolling in the shadows: Analyzing the extraction of mev across layer-2 rollups, 2024. doi:10.48550/arXiv.2405.00138.

56 Umbra Research. MEV on Solana. https://www.umbraresearch.xyz/writings/mev-on-solana, 2023. Accessed: 2025-05-25.

57 Uniswap. Universal router repository. https://github.com/Uniswap/universal-router, 2022. GitHub repository, accessed 2025-05-09.

58 Uniswap Labs. Universal router – overview. URL: https://docs.uniswap.org/contracts/universal-router/overview.

59 Ye Wang, Yan Chen, Haotian Wu, Liyi Zhou, Shuiguang Deng, and Roger Wattenhofer. Cyclic arbitrage in decentralized exchanges, 2022. arXiv:2105.02784.

60 Ben Weintraub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. A flash (bot) in the pan: Measuring maximal extractable value in private pools. In *ACM Internet Measurement Conference (IMC), Nice, France*, 2022.

61 Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. On the Just-In-Time Discovery of Profit-Generating Transactions in DeFi Protocols. In *IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA*, 2021.

62 Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-Frequency Trading on Decentralized On-Chain Exchanges. In *IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA*, 2021.

63 Burak Öz, Jonas Gebele, Parshant Singh, Filip Rezabek, and Florian Matthes. Playing the mev game on a first-come-first-served blockchain, 2024. doi:10.48550/arXiv.2401.07992.

64 Burak Öz, Danning Sui, Thomas Thiery, and Florian Matthes. Who wins ethereum block building auctions and why?, 2024. doi:10.48550/arXiv.2407.13931.

Efficient Parallel Execution of Blockchain Transactions Leveraging Conflict Specifications

Parwat Singh Anjana*
Supra Research, Udaipur, India

Matin Amini
University of Southern California, Los Angeles, CA, USA

Rohit Kapoor
Supra Research, New Delhi, India

Rahul Parmar
Supra Research, Ahmedabad, India

Raghavendra Ramesh
Supra Research, Brisbane, Australia

Srivatsan Ravi*
Supra Research, Los Angeles, CA, USA
University of Southern California, Los Angeles, CA, USA

Joshua Tobkin
Supra Research, Taipei, Taiwan

Abstract

Parallel execution of smart contract transactions in large multicore architectures is critical for higher efficiency and improved throughput. The main bottleneck for maximizing the throughput of a node through parallel execution is transaction conflict resolution: when two transactions interact with the same data, like an account balance, their order matters. Imagine one transaction sends tokens from account A to account B, and another tries to send tokens from account B to account C. If the second transaction happens before the first one, the token balance in account B might be wrong, causing the entire system to break. Conflicts like these must be managed carefully, or you end up with an inconsistent, unusable blockchain state.

Traditional software transactional memory (STM) has been identified as a possible abstraction for the concurrent execution of transactions within a block, with Block-STM pioneering its application for efficient blockchain transaction processing on multicore validator nodes. This paper presents a parallel execution methodology that leverages conflict specification information of the transactions for block transactional memory (BTM) algorithms. Our experimental analysis, conducted over synthetic transactional workloads and real-world blocks, demonstrates that BTMs leveraging conflict specifications outperform their plain counterparts on both EVM and MoveVM. Our proposed BTM implementations achieve up to 1.75× speedup over sequential execution and outperform the state-of-the-art Parallel-EVM (PEVM) execution by up to 1.33× across synthetic workloads.

2012 ACM Subject Classification Computing methodologies → Parallel computing methodologies

Keywords and phrases Blockchain, Smart Contract, Parallel Execution, Conflict Specifications

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.29

Related Version *Full Version*: https://arxiv.org/abs/2503.03203

*Corresponding authors.

© Parwat Singh Anjana, Matin Amini, Rohit Kapoor, Rahul Parmar, Raghavendra Ramesh, Srivatsan Ravi, and Joshua Tobkin;
licensed under Creative Commons License CC-BY 4.0
7th Conference on Advances in Financial Technologies (AFT 2025).
Editors: Zeta Avarikioti and Nicolas Christin; Article No. 29; pp. 29:1–29:26

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Blockchains are in a race to reduce transaction latency, improving user experience and increasing throughput to meet anticipated demand. This optimization effort has examined every step of a blockchain transaction, including data dissemination, ordering, execution, and storage. In this paper, we investigate the state-of-the-art in optimizing the execution step of a transaction workflow.

Blockchains require all nodes to deterministically reach the same final state when executing a block of transactions. To leverage multicore architectures, modern blockchain designs seek to maximize parallel execution throughput while preserving deterministic consistency across all nodes. Traditional software transactional memory (STM) has been identified as a possible abstraction for the concurrent execution of transactions within a block. However, the execution order must follow a *preset order* of transactions in a block. The main bottleneck for maximizing the throughput of a node through block transactional memory (BTM) that respects the preset order is transaction *conflict* resolution: when two transactions, T_1 and T_2 in a block, interact with the same state, like an account balance, order in which the reads and writes on the state occurs matters for the correctness of the execution. Imagine that a transaction T_1 sends tokens from account A to account B, and another transaction T_2 tries to send tokens from account B to account C. If T_2 happens before the first one, the token balance in account B might be wrong, causing the entire system to break. Conflicts like these must be managed carefully, or you end up with an inconsistent, unusable blockchain state.

Ethereum maintain consistency by sequentially executing a block of transactions, underutilizing the multicore architectures of its nodes. In contrast, Solana [33] and Aptos [5] utilize parallel execution, though in different settings. Aptos employs the classical STM technique to execute transactions in parallel with BTM, without requiring any priori read-write access specifications with transactions. Aptos' Block-STM [17], represents the current state-of-the-art, applying classical STM [32] techniques to the execution of ordered blocks of transactions. STM techniques are typically speculative or optimistic, meaning that they attempt to execute as many transactions in parallel as possible, while detecting and resolving conflicts. If a transaction reads a value that later changes due to another transaction, it must be re-executed to maintain consistency. In contrast, Solana's parallel execution requires transactions to be tagged with the accounts read and written, though this is not a STM-based algorithm. We wondered whether the read-write access specifications could enhance a BTM algorithm's performance, and our findings indicated a positive correlation.

Firstly, we observe that some popular blockchains equip transactions a priori with *read-write sets*, i.e., the sets of accounts that the transaction *may* access. For instance, Solana leverages user-provided read-write sets as access specifications in its lock-profile-based iterative, parallel execution strategy [38]. Similarly, Sui [16] utilizes user-provided access specifications to execute transactions in a causally-ordered manner. Secondly, we further observe that, when such access specifications are not available a priori, it is possible to derive access specifications for public entry functions statically at the time of deployment of smart contracts using data-flow analysis on the smart contract code. This is a one-time computation, enabling efficient utilization of these specifications during transaction execution. These access specifications essentially serve as *conflict specifications* for any parallel execution technique on ordered blocks of transactions. As a complementary approach, we observe that it is possible to build conflict specifications efficiently at runtime by checking whether the accounts affected by a set of transactions (for instance, *payment transfers* in Aptos Move or *ETH* or *ERC-20* transfers in Ethereum [12]) are mutually disjoint.

This is where we see the challenge of improving BTM to further optimize execution by leveraging conflict specifications. Specifically, we seek to answer the following questions: Do conflict specifications improve transaction-execution throughput in existing BTM techniques? If so, what is the best BTM technique to optimally leverage these conflict specifications? By constructively answering these questions, we aim to advance the frontiers of parallel transaction execution, driving significant improvements in blockchain scalability and efficiency.

Contributions. Our key contributions are as follows:

1. Leveraging our insight that conflict specifications must be exploited for maximizing throughout in parallel execution, we detail two BTM algorithms, dBTM and oBTM, that can efficiently execute block transactions when *sound*, but possibly *incomplete* conflict specifications are available.
2. We present implementations of our algorithms for Ethereum VM (EVM) and Aptos MoveVM and conduct a detailed empirical analysis on real-world blocks, thoroughly analyzing the best and worst-case performance. Additionally, we design a workload generator for analyzing performance on large blocks and unconventional workloads.
3. We present a rigorous formalism and implementation for how conflict specifications are efficiently constructed for EVM and MoveVM. We then present a conflict analyzer integrated implementation (iBTM) that outperforms existing parallel execution approaches.
4. We evaluated the performance of iBTM against sequential and state-of-the-art Parallel-EVM (PEVM) [28] on both synthetic and historical workloads. In synthetic workload, the proposed iBTM achieve up to 1.75× speedup over sequential and 1.33× over PEVM and maintain consistent improvements, with an average speedup of 1.24× over PEVM. It achieves a maximum speedup of 2.43× over sequential execution on historical workloads.

The methodology is rigorously demonstrated for EVM and MoveVM, for which conflict specifications are algorithmically derived as part of our integrated implementation. However, we remark that the purpose of our modular presentation and detailed ablation studies is to demonstrate how easily our parallel execution algorithms can be applicable to other blockchain ecosystems, specifically Sui and Solana, which explicitly provide a priori transactional access sets that can be used to derive conflict specifications. By identifying dependencies upfront, we can avoid unnecessary rollbacks and retries, making our system much more efficient for prevalent blockchain infrastructures. This allows the system to automatically separate transactions that need special handling due to dependencies, making the entire execution process more streamlined and as we demonstrate, highly efficient. Finally, we present how this approach also unleashes the possibility of workload *adaptive* execution that can leverage a sequential or parallel execution algorithm, depending on the block's "conflict threshold".

Roadmap. The rest of the paper is organized as follows. Sections 2 and 3 presents related work and motivation on parallel execution of block transactions, respectively. Section 4 introduces the system model and the proposed BTM algorithms. Section 5 presents experimental results on parallel execution using conflict specifications. Section 6 formalize and implement conflict specification generation and evaluate the full integration in both real-world and synthetic workloads. For succinctness of presentation, we focus on the EVM in the paper and only summarize the MoveVM evaluation results. We conclude in Section 7.

2 Related Work

This section reviews smart contract execution models and existing parallel execution algorithms for block transactional execution.

Several models have been proposed for the execution of transactions in blockchains; one such model involves the *block proposer* executing transactions, generating a block containing state differences, and subsequently propagating this block across the network for validators to validate. Another approach entails the block proposer determining the transaction order in a block, after which all nodes reach consensus on this order before executing transactions in parallel. The former is known as the *Ethereum model* [12], while the latter is known as the *Aptos model* [5]. Another model facilitates parallel execution by incorporating read-write sets with transactions; this is commonly called the *Solana model* [33]. A fourth model exploits resource ownership to enable parallel execution, a methodology known as the *Sui model* [36].

These four execution models can be broadly categorized into two distinct classes that aim to optimize transaction execution. The first class is called the *read-write aware execution* which relies on transaction access hints provided by clients to facilitate parallel execution either through preprocessing in the form of a directed acyclic graph (DAG) or runtime techniques based on lock profiling. The second class comprises techniques that leverage run-time execution techniques, such as STM or locks, to optimistically execute transactions. We call this approach the *read-write oblivious execution*.

Read-write Aware Execution. Transactions on blockchains such as Solana [33] and Sui [36] upfront specify the accounts they access in read and write mode during execution. These access hints are used to enable parallel execution, either through static analysis or by employing a runtime scheduler that resolves read-write set conflicts and executes transactions in parallel. Alternatively, a DAG can be constructed from the read-write access sets to partition transactions into independent groups (iterations in Solana) for parallel execution. Specifically, Solana's [33] SeaLevel [37, 38], leverages read-write sets and lock profiling to execute transactions iteratively. Each iteration involves a locking phase to detect conflicts and an execution phase where non-conflicting transactions are executed in parallel, while conflicting transactions are deferred to subsequent iterations until all transactions in the block are processed. The iteration information is then included in the block by the block proposer to support parallel execution during validation at the validators.

In contrast, Sui [36] introduces an object-based state model to identify independent transactions. Objects are shared or exclusively owned, each with a unique identity and owner address. Dependency identification is simplified by determining whether multiple transactions access the same shared object. Based on this model, Sui enables parallel execution [15, 16] where transactions on owned objects are executed in parallel and can completely bypass consensus, while those on shared objects run sequentially through consensus to avoid conflicts. Other works, such as ParBlockchain [1] and Hyperledger Sawtooth [30], employ lock-based or conflict-analysis techniques, while DiPETrans [6] and the efficient scheduler for Sawtooth [25] utilize DAG-based read-write aware execution.

Read-write Oblivious Execution. In this class, the goal is to execute an ordered set of transactions in parallel as if they were executed sequentially and arrive at the same state. The key idea is that some transactions may not be conflicting, i.e, they do not read or write any common states; hence, can be executed in parallel, enabling execution acceleration that arrives at the correct sequential result. For the Ethereum model, Dickerson et al. [11] propose, the first pioneering work on parallel execution of block transactions. They proposed using pessimistic ScalaSTM for parallel execution at the block proposer and a happen-before graph for parallel execution at the validators. Later, Anjana et al. [2, 3] proposed an optimistic STM (OSTM) based multi-version timestamp ordering protocol for parallel execution at

the proposer, while a DAG-based efficient parallel execution at validators. Saraph and Herlihy [29] proposed a simple *bin-based two-phase* approach. In the first phase, the proposer uses locks and tries to execute transactions concurrently by rolling back those that lead to conflict(s). Aborted transactions are kept in a sequential bin and executed sequentially in the second phase. Later, OptSmart [4] proposed an approach that combines the idea of bin-based approach with the OSTM approach for efficient parallel execution.

In the Aptos model, differed execution of transaction based on preset serializable and multi-version concurrency control is proposed in the Block-STM [17]. Rather than speculatively executing block transactions in any order, they employ it on ordered-sets, called the *preset order*. Each validator uses Block-STM independently to execute a leader proposal of an ordered set of transactions in parallel to get the same state. This has currently been implemented on the Aptos blockchain [5] and is the most promising approach, as it does not require additional information to be attached to the transactions or in the block for parallel execution. It has been adopted for execution on the Polygon PoS Chain [26], where it is already live on the mainnet. The proposer uses Block-STM to execute transactions in parallel and includes a DAG in the block to allow deterministic and safe parallel execution at validators. Recently, SeiGiga [31] has also adopted Block-STM.

3 Motivation and Overview

This section motivates the need for parallel execution in blockchain systems and outlines the benefit of leveraging conflict specification for parallel execution.

Motivation. Naturally, the throughput of STM-based execution varies widely based on the level of transaction conflicts, aborts, and re-executions (triggered re-validations of the transaction's read state), presenting both a challenge and an opportunity to further optimize.

Consider the scenario shown in Figure 1, in which we have two transactions T_1 and T_2 running concurrently. We consider that T_1 must precede T_2 in the preset order (denoted $T_1 \to T_2$). Without *a priori* knowledge of read-write conflicts in the set of accounts (states)

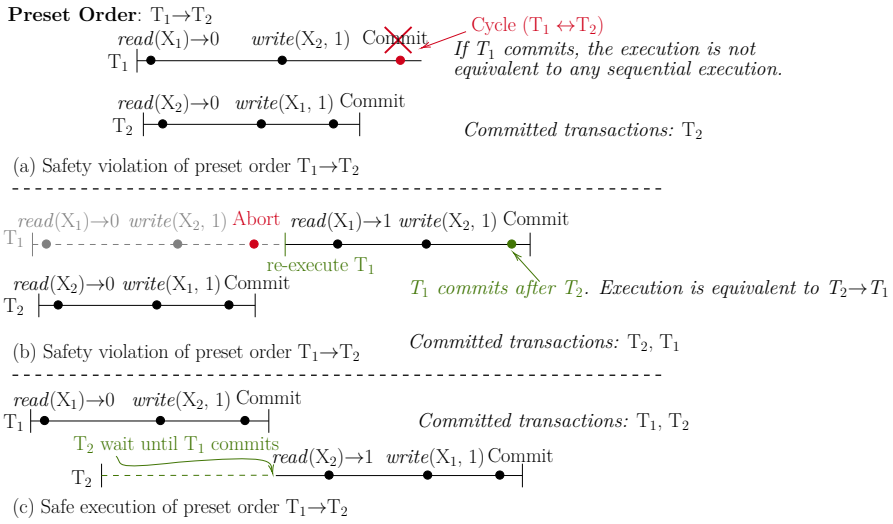


Figure 1 Safe execution in preset order: (a) and (b) show unsafe executions due to arbitrary or incorrect serialization; (c) shows a safe execution where T_2 waits for T_1 to commit, preserving the preset serialization order.

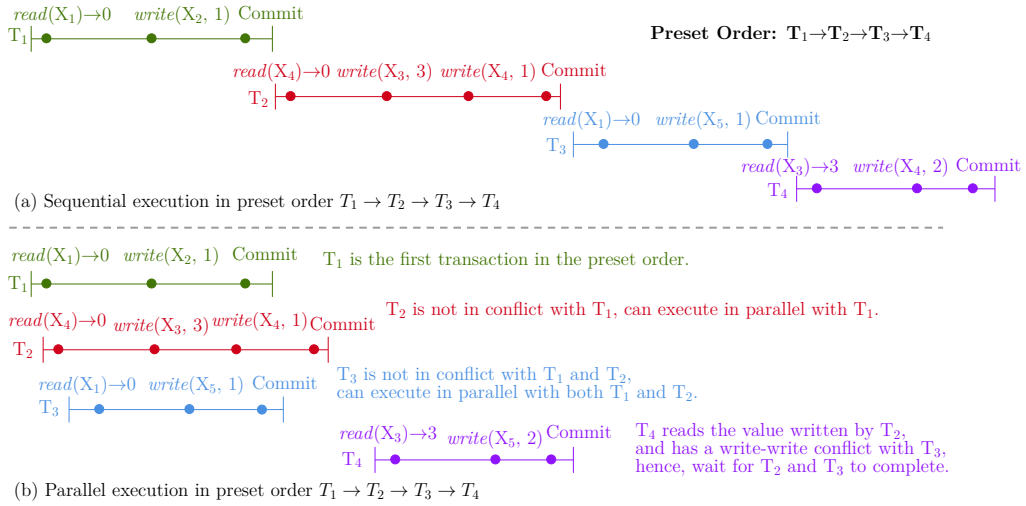


Figure 2 Leveraging conflict specifications a priori for parallel execution.

accessed, committing T_2 prior to committing T_1 may result in a safety violation. To illustrate this, consider the following execution: T_1 reads an account X_1 (value 0), following which T_2 reads X_2 (value 0), followed by a write of a new value 1 to X_1. Observe that if T_2 commits at this point in the execution and if T_1 writes a new value 1 to X_2 after the commit of T_2, then the resulting execution does not respect the preset order in any extension. Clearly, if T_1 commits, the resulting execution is not equivalent to any sequential execution, as shown in Figure 1a. Alternatively, if T_1 aborts and then re-starts, any read of X_2 will return the value 1 that is written by T_2, thus not respecting the preset order, as illustrated in Figure 1b. Consequently, the only possible way to avoid this, requires T_2 to wait until T_1 commits (see Figure 1c). Now consider a modification of this execution in which T_2 reads X_3 and writes to X_4. In this case, T_2 does not have a *read-from conflict* with T_1 allowing T_1 and T_2 to run in parallel with almost no synchronization. However, this is possible only if threads that execute these transactions are *aware* of the conflict prior to the execution.

After studying both read-write oblivious and aware models (taxonomized in Section 2), we conclude that the approach that leverage conflict specifications is the best way forward. It can be considered the "Goldilocks" approach which creates conflict specifications to overcome the limitations of parallel execution imposed by the preset order of transaction. This is what will allow us to scale with parallel execution efficiently and maximize throughput.

Overview. Consider the execution of four transactions T_1–T_4 depicted in Figure 2, as a running example. In Figure 2a, we observe a sequential execution, despite T_1 and T_2 accessing different states, limiting throughput. This is, for instance, the case in Ethereum [12] where transactions are executed sequentially. In contrast, Figure 2b illustrates parallel execution, where T_1, T_2 and T_3 execute in parallel and improve throughput. However, T_4 has a *reads-from* conflict with T_2 on X_3 and therefore must wait for T_2 to commit. Moreover, T_4 has a *write* conflict with T_3 and must either wait for T_3 to commit or employ data structures that allow tracking of both writes performed to X_5 (à la *multi-versioning* [18]).

As we constructively observe, the states accessed by these transactions can be efficiently derived a priori to build *sound*, but possibly incomplete specifications. Depending on how complete the derived specifications are, this information can be used to minimize runtime transactional conflicts and thus avoid unnecessary transaction aborts during parallel execution. However, as the results in this paper show, not only are there rigorous theoretical approaches

to deriving conflict specifications in smart contract ecosystems like Ethereum, fast techniques exist to identify a non-trivial number of transactional conflicts within a block, which provide significant speedup for parallel execution.

4 BTM Execution From Conflict Specifications

In this section, we formally present block transactional memory in the asynchronous shared memory model, detail our BTM algorithms that leverage conflict specification and satisfy preset serializability.

4.1 Model

The model of BTM is presented in the standard asynchronous shared memory model [22].

Transactions. A *transaction* is a sequence of *transactional operations*, reads and writes, performed on a set of virtual machine (VM) states. A BTM[1,. . . , n] *implementation* provides a set of concurrent *processes* with deterministic algorithms that implement reads and writes on accounts using a set of *shared memory locations* accessed by the n transactions with a preset order $T_1 \rightarrow, \ldots, T_n$. More precisely, for each transaction T_k, a BTM implementation must support the following operations: $read_k(X)$, where X is an object, that returns a value in a domain V or a special value $A_k \notin V$ (*abort*), $write_k(X, v)$, for a value $v \in V$, that returns *ok* or A_k, and $tryC_k$ that returns $C_k \notin V$ (*commit*) or A_k. The transaction T_k completes when any of its operations returns A_k or C_k.

Executions, Histories and Conflicts. A BTM *execution* is a sequence of *events* performed on shared memory states by an interleaving of transactions as prescribed by the implementation. To avoid introducing additional technical machinery that is not strictly necessary to follow the algorithmic exposition in this paper, we do not define executions using configuration semantics as is common in traditional shared memory systems [18, 22], although it would be straightforward to do so.

A BTM *history* is the subsequence of an execution consisting of the invocation and response events of operations of the transactions. For a transaction T_k, we denote all objects accessed by its read and write as $Rset(T_k)$ and $Wset(T_k)$, respectively. We denote all the operations of a transaction T_k as $Dset(T_k)$. The *read set* (resp., the *write set*) of a transaction T_k in an execution E, denoted $Rset_E(T_k)$ (resp. $Wset_E(T_k)$), is the set of objects that T_k attempts to read (resp. write) by issuing a read (resp. write) invocation in E. The *data set* of T_k is $Dset(T_k) = Rset(T_k) \cup Wset(T_k)$. T_k is called *read-only* if $Wset(T_k) = \emptyset$; *write-only* if $Rset(T_k) = \emptyset$ and *updating* if $Wset(T_k) \neq \emptyset$.

We say T_i and T_j *conflict* in an execution E if there exists a common state X in $Dset(T_i)$ and $Dset(T_j)$ such that X is contained within $Wset(T_i)$ or $Wset(T_j)$, or both. Furthermore, we say that T_i, T_j *read-from conflict* if $Wset(T_i) \cap Rset(T_j) \neq \emptyset$ and T_i appears before T_j in the preset order. Note that the definition of read-from conflict, unlike that of a conflict, relies on a preset order existing between the two transactions.

Let H be a sequential history, i.e., no two transactions are concurrent in H. For every operation $read_k(X)$ in H, we define the *latest written value* of X as follows: if T_k contains a $write_k(X, v)$ that precedes $read_k(X)$, then the latest written value of X is the value of the latest such write to X. Otherwise, the latest written value of X is the value of the argument of the latest $write_m(X, v)$ that precedes $read_k(X)$ and belongs to a committed transaction in H. This write is well-defined since H can be assumed to start with an initial transaction

writing to all states. We say that a sequential history S is *legal* if every read of a state returns the *latest written value* of this state in S. It means that sequential history S is legal if all its reads are legal.

Preset Serializability. Given a set of n transactions with a *preset* order $T_1 \to T_2 \to ... \to T_n$, we need a deterministic parallel execution protocol that efficiently executes block transactions using the serialization order and always leads to the same state, even when executed sequentially. We formalize this using the definition of *preset serializability.*

▶ **Definition 1** (Preset serializability). *Let H be a history of a Block-STM[1,. . . , n] implementation M. We say that H is* preset serializable *if H is equivalent to a legal sequential history S that is $H_1 \cdots H_i \cdot H_{i+1} \cdots H_n$ where H_i is the complete history of transaction T_i. We say that a Block-STM[1,. . . , n] implementation M is* preset serializable *if every history of M is preset serializable.*

4.2 Algorithm overview

The core idea behind our algorithm is that only independent transactions are executed in parallel, ensuring that no race conditions arise during execution. For any transaction T_k, if there exists a transaction $T_i \notin \text{cSet}(\text{T}_k)$ that precedes T_k in the preset serialization order, the execution of T_k is deferred until T_i completes (here $\text{cSet}(\text{T}_k)$ denotes the set of transactions that do not conflict with T_k). Additionally, each transaction is followed by validation to ensure that, if two dependent transactions are executed in parallel due to incorrect conflict specification, the one higher in the preset order can abort and re-execute. Thus, the output of the algorithm will be the same as that of a sequential execution.

Performance improvement over parallel execution techniques based on Block-STM is achieved by reducing the number of aborts and re-executions. This is made possible by leveraging the knowledge of the transaction conflicts: a transaction T_k is executed only after ensuring that all preceding transactions in the preset order are in its independence set, $\text{cSet}(\text{T}_k)$. This targeted execution strategy minimizes conflicts and improves throughput.

Problem Statement. In this section, we ask the following question: given a set of n transactions $T_1 \to, \ldots, \to T_n$ and $\text{cSet}(\text{T}_k)$ for all $k \in \{1, \ldots, n\}$, what are the most efficient algorithms for implementing Block-STM[1,. . . , n]. We implemented two different algorithms, leveraging conflict specification in DAG BTM (dBTM) and optimized BTM (oBTM), based on how the *scheduler* utilizes the transaction conflict specification for efficient parallel execution. As explained in Sections 1 and 2, this mechanism for implementing BTM is applicable directly to the read-write aware models like Sui and Solana in which the BTM is explicitly provided the transactional read-write sets. More importantly, as we demonstrate in Section 6, we can efficiently construct conflict specifications for read-write oblivious models like Ethereum EVM and Aptos MoveVM, even if under-approximate (i.e., incomplete), and still reap the benefits of our proposed methodologies for implementing BTM.

Leveraging Conflict Specification in DAG BTM (dBTM)

This approach utilizes the conflict specifications (assumed to be correct or overapproximated) and the preset order to create a DAG, which serves as a partial order and is provided as input for the scheduler-less execution algorithm. In the DAG, transactions are represented as vertices, whereas conflicts among transactions are denoted as directed edges. The *indegree* field is added with each vertex (transaction) to track dependencies with prior transactions in the preset order; a transaction becomes eligible for execution when its indegree is zero. A

transaction is independent if its cSet contains all preceding transactions in the preset order, resulting in an indegree of zero. During execution, the non-zero indegree transactions wait for the preceding transactions to commit and clear dependencies. At the time of commit, the committing transaction decreases the indegree of all dependent transactions.

We now present the the preset-serializable dBTM (Algorithm 1), which leverages conflict specifications to construct a transaction DAG representing a preset-serializable partial order.

Algorithm 1 dBTM[1,. . . , n]: It is a DAG-based scheduler to execute independent transactions in parallel. The *indegree* field of each transaction vertex in the DAG tracks dependencies on prior transactions. Consider a transaction T_k being executed by a process p_k.

Input: T: list of transactions in the block B_i; S: pre-state− state before execution of block B_i; *cSet*: specifications for transactions in T.
Shared Locations: *indegree*$[T_k]$: the number of transactions in preset order T_i is dependent on; *dependents*$[k]$: the set of transactions dependent upon T_k; a version list $\langle v_j \rangle$ for each state X_j.

```
1:  Fun read_k(X_j):
2:      if X_j ∉ Wset(T_k) then
            // Read the latest version of X_j
               created by a T_i that precedes T_k.
3:          [ov_j, i] := read_lvp(T_k, X_j)
4:          Rset(T_k) := Rset(T_k) ∪ {X_j, [ov_j, i]}
5:          return ov_j
6:      else
            // X_j is in Wset(T_k)
7:          [ov_j, ⊥ ] := Wset(T_k).locate(X_j)
8:          return ov_j
9:  Fun read_lvp(T_k, X_j):
10:     [ov, i] := [0, 0];
        // Read the largest version of X_j
           created by a T_i that precedes T_k.
11:     forall [ov_j, i] ∈ X_j do
12:         if k > i then
13:             [ov, i] := [ov_j, i]
14:     return [ov_j, i]
15: Fun write_k(X_j, v):
16:     nv_j := v
17:     if X_j ∉ Wset(T_k) then
18:         Wset(T_k) := Wset(T_k) ∪ {X_j, [nv_j, k]}
19:     else
            // X_j is in Wset(T_k), update its
               current value to v.
20:         Wset(T_k) := Wset(T_k).update(X_j, [nv_j, k])
21:     return ok
22: Fun tryC_k ():
        // Ensure commit order
23:     if indegree[k] ≠ 0 then
24:         wait until indegree[k] = 0;
        // Write back to shared memory
25:     forall X_j ∈ Wset(T_k) do
26:         Write(X_j, [nv_j, k])
        // Clear dependencies
27:     forall T_i ∈ dependents(i) do
28:         indegree[i] ← indegree[i] − 1;
29:     return C_k
30: Fun gen_dag(cSet):
31:     block_size ← size_of(B_i);
32:     for k ∈ (0, block_size - 1)) do
33:         dependencies[k] := ∅
34:         dependents[k] := ∅
35:         indegree[k] ← 0;
        // Compute complement sets
36:     for k ∈ (0, block_size - 1) do
37:         cSetComp(T_k) ← {j | j < k and j ∉
              cSet(T_k)};
        // Use cSetComp to build dependencies
38:     for k ∈ (0, block_size - 1) do
39:         foreach j ∈ cSetComp(T_k) do
40:             dependencies[k].add(j)
41:             dependents[j].add(k)
42:         indegree[k] := |dependencies[k]|
43:     return (dependents, indegree)
```

Implementation state. The *DAG* is implemented using two primary data structures: 1. *indegree*, where *indegree[k]* denotes the number of preceding transactions in the preset order that transaction T_k depends on; and 2. *dependents*, which maps each T_k to a set of its dependent successors. For each *VM state* X_i, the algorithm maintains a memory location v_i that stores a set of tuples $([v_1, k], [v_2, k'], \ldots)$. Each tuple $[v, k]$ represents the value v of X_i and k is the transaction T_k that wrote that value.

Read implementation. The $read_k(X_j)$ function returns the value of state X_j as visible to transaction T_k. It first checks whether X_j is already present in $Wset(T_k)$ at line 2. If not, it reads the largest version written by a transaction T_i in shared memory such that

$T_i \to T_k$, using the helper function *read_lvp()* defined in lines 9–14. This value, along with its source, is then recorded in $Rset(T_k)$ at line 4. If X_j has already been locally written by T_k, it is returned directly from the $Wset(T_k)$ at line 7. The helper function $read_lvp(T_k, X_j)$ (lines 9–14) is invoked by $read_k$ to identify the latest version of X_j in the multi-version data structure created by a transaction that precedes T_k in the DAG. It iterates over all versions of X_j in shared memory and selects the one with the highest index $i < k$. This mechanism ensures multi-version consistency by enforcing serialization semantics across transactions.

Write implementation. The $write_k(X_j, v)$ function (lines 15 to 21) stages a write to VM state X_j with value v in the local context of T_k. If this is the first write to X_j by T_k, an entry is created in $Wset(T_k)$. If X_j has been written previously, the existing entry is updated with the new value. This operation does not alter shared memory but updates the local write set.

Commit implementation. The $tryC_k()$ function, lines 22 to 29, performs the commit logic for transaction T_k. It first waits until all dependencies are satisfied, that is, all preceding transactions in the DAG on which T_k depends are committed and $indegree[k] := 0$ (line 24). Then updates the shared memory with writes from $Wset(T_k)$ at line 25. At line 27, it clears the dependencies of all successor transactions, specifically, for each transaction $T_i > T_k$ where $T_k \notin cSet(i)$, it decrements $indegree[i]$ by 1 (line 28). Finally, it returns C_k at line 29.

DAG Generation. The *gen_dag(cSet)* function, lines 30 to 43, constructs a DAG from the specification *cSet*, which maps each transaction T_k to the set of preceding transactions (in preset order) that T_k is independent of. It initializes the per-transaction data structures ($dependencies := \emptyset$, $dependents := \emptyset$, and $indegree \leftarrow 0$) in lines 32–35. Then, computes the complement conflict set $cSetComp(T_k)$ for each transaction $T_k \in T$ (line 36), which includes all preceding transactions of T_k that are not in $cSet(T_k)$. These are the transactions with which T_k conflicts, and hence it must be conservatively dependent on them in the DAG. Using *cSetComp*, the function constructs the dependency edges (lines 38–42). For every $T_k \in T$, the function sets $indegree[k]$ to the cardinality of the dependency set of T_k. Finally, the function returns references to the *dependents* and *indegree* structures at line 43, which are used as input to DAG-based scheduler for execution.

Proof of Correctness. We prove preset serializability for the Algorithm 1 by adapting the proof structure from traditional STM literature for the block transactional model [18, 19]. The proof is provided in the extended version on arXiv due to space limitations.

Leveraging Conflict Specification in Optimized BTM (oBTM)

The oBTM algorithm optimizes the Block-STM scheduler by integrating independence information (i.e., *cSet*) associated with transactions, which is assumed to be complete and provided as input. The scheduler is modified to allow transactions that are independent of all preceding transactions in the preset order to execute without validation. A greater number of such independent transactions improves the execution efficiency. This method builds on the existing scheduler of Block-STM [17] for MoveVM and PEVM [28] developed by the RISE chain [27] for EVM. In the absence of conflict specifications, the algorithm defaults to optimistic execution as in PEVM and Block-STM.

The design of oBTM, shown in Algorithm 2, leverages the conflict specifications to enable efficient optimistic execution by minimizing the validation overhead of vanilla PEVM and Block-STM. Consider a transaction T_k being executed by a process p_k.

Algorithm 2 oBTM[1,. . . , n]: The approach optimizes the Block-STM scheduler to allow transactions that are independent of all previous transactions to execute without validation. Consider a transaction T_k being executed by a process p_k.

Input: T: list of transactions in the block B_i; S: pre-state− state before execution of block B_i; $cSet$: specifications for transactions in T.
Shared Locations: $cSet(T_k)$: set of transactions T_i such that $i < k$ and T_k is independent of T_i.

```
1:  Fun read_k(X_j):
2:    if X_j ∉ Wset(T_k) then
        // Read the largest version of X_j created by a tx T_i preceding T_k.
3:      [ov_j, i] := read_lvp(T_k, X_j)
4:      if [ov_j, i].is_estimate then
5:        if ¬ add_dependency(T_k, T_i) then
6:          retry(T_k)
          // If T_k read an estimated value written by T_i
7:        return A_k
8:      Rset(T_k) := Rset(T_k) ∪ {X_j, [ov_j, i]}
9:      return ov_j
10:   else
11:     [ov_j, ⊥] := Wset(T_k).locate(X_j)
12:     return ov_j
13: Fun write_k(X_j, v):
14:   ov_j := v
15:   if X_j ∉ Wset(T_k) then
16:     Wset(T_k) := Wset(T_k) ∪ {X_j, [nv_j, k]}
17:   else
        // X_j is in Wset(T_k), update current value to v.
18:     Wset(T_k) := Wset(T_k).update(X_j, [nv_j, k])
19:   return ok
20: Fun tryC_k ():
21:   if ∃ T_i ∉ cSet(T_k) : i < k then
22:     forall X_j ∈ Wset(T_k) do
          // Update Wset(T_k) in shared memory
23:       write(X_j, [nv_j, k])
        // Read set validation
24:     if ∃ X_j ∈ Rset(T_k): [ov_j, k] ≠ read_lvp(T_k, X_j) then
          // On validation failure, versions of each X_j ∈ Wset(T_k) are marked as estimated in shared memory.
25:       forall X_j ∈ Wset(T_k) do
26:         mark_estimate(nv_j, k)
27:       return A_k
28:     return C_k
29:   else
        // T_k is an independent transaction.
30:     forall X_j ∈ Wset(T_k) do
31:       write(X_j, [nv_j, k])
        // Skip validation for T_k and commit.
32:     return C_k
33: Fun add_dependency(id, blocking_id):
34:   if txn_status[blocking_id] == Executed then
35:     return false
36:   return true
```

Implementation state. For each *state* X_i, the algorithm maintains a memory location v_i storing tuples $([v_1, k], [v_2, k'], \ldots)$, where $[v, k]$ represents a value v written to X_i by T_k.

Read implementation. The $read_k(X_j)$ function returns the most recent committed value of X_j visible to transaction T_k. If $X_j \notin Wset(T_k)$, the latest version written by a predecessor T_i is read from shared memory at line 3 using function *read_lvp()* (as discussed in dBTM algorithm). If the version is an estimate and T_i has not yet committed, *add_dependency* is invoked to check if T_k must wait (line 5); if so, T_k retries at line 6. Otherwise, the version is added to $Rset(T_k)$ for subsequent validation at line 8. If $X_j \in Wset(T_k)$, meaning it is already in transaction's write set, the value is returned directly from $Wset(T_k)$ at line 11.

Write implementation. The $write_k(X_j, v)$ performs a write that assigns a value v to state X_j by transaction T_k. It first checks whether $X_j \notin Wset(T_k)$, if so, a new entry $[v, k]$ is inserted into $Wset(T_k)$ (line 16); otherwise, the existing entry is updated with the new value v (line 18). This operation does not update in shared memory; instead, it maintains a local view of writes until commit (line 19).

Commit implementation. The $tryC_k()$ function attempts to commit transaction T_k. If T_k depends on any predecessor transaction(s) in preset, it first updates its $Wset(T_k)$ to shared memory (a multi-version data structure) at line 23 and then validates its read set $Rset(T_k)$

against the latest values committed to shared memory at line 24. If validation fails, the versions written by T_k in the multi-version structure are marked as estimated at line 26, and T_k aborts by returning A_k at line 27. If validation succeeds, or if T_k is an independent transaction (its $Wset(T_k)$ is committed to shared memory and validation is skipped), the function commits T_k by returning C_k at line 28 and line 32, respectively.

5 Implementation and Evaluation of BTM from Conflict Specifications

In this section, we analyze the execution latency and throughput of dBTM and oBTM, compared to baseline sequential execution and state-of-the-art parallel execution techniques.

5.1 Implementation

EVM Implementation. We compare dBTM and oBTM with PEVM [28], a version of Block-STM [17] for the EVM developed by the RISE chain [27]. Experiments are conducted on REVM [8] version `12.1.0`, an EVM implementation written in Rust. We used the core data structures provided by PEVM version `0.1.0` (commit hash `f0bdb21`) and implemented our algorithms on top of them. Specifically, we used a hash map to store the cSet, which allowed us to store both the list of dependent transactions and the indegree of each transaction in the DAG. A custom parallel-queue is implemented to manage and enable efficient parallel execution across multiple threads.

We analyzed the performance on both synthetic and historical blocks from Ethereum's mainnet. In the EVM, each transaction pays a gas fee to the *Coinbase* account, which belongs to the block proposer. As a result, every transaction updates the Coinbase account, leading to a 100% write-write conflict with all preceding transactions. To address this, we defer the fee transfer; that is, the gas fees are locally collected per transaction and credited to the Coinbase account only at the end, after all transactions in the block have been executed.

MoveVM Implementation. We analyzed the performance of our dBTM and oBTM approaches against the baseline Block-STM [17], which runs DiemVM (an earlier version of MoveVM) using the test setup from [7]. The evaluation includes two primary workloads: (i) peer-to-peer (P2P) transfers, as provided in the original test setup [7], and (ii) synthetic workloads (batch transfers and a generic mixed workload) we designed to mimic diverse execution patterns. Due to space constraints and for presentation succinctness, we defer the detailed analysis of BTM on MoveVM to the arXiv version.

Experimental Setup. We ran experiments on a single-socket AMD machine consisting of 32 cores (64 vCPUs), 128 GB of RAM, and running Ubuntu 18.04 with 100 GB of SSD. The experiments are carried out in an execution setting of in-memory using REVM and MoveVM, and abstract away the latency of persistent storage access. We ran experiments 52 times, with the first 2 runs designated as warm-up. Each data point in the plots represents an average throughput (tps) and latency (ms), where each execution is repeated 50 times.

5.2 EVM Analysis

Figure 3 to Figure 6 show the relationship between varying threads or varying conflicts ($\alpha-$tail distribution factor) within the block, represented on the X-axis, on two important performance metrics: *throughput (tps)*, represented by histograms on the primary Y-axis (Y1), and *execution latency (ms)*, as line graphs on the secondary Y-axis (Y2).

Construction and Testing of Synthetic Workloads

We conducted experiments on synthetic workloads to analyze throughput and latency under various execution conditions. A synthetic workload is a computational workload that does not depend on user input but instead simulates transaction types and concurrency patterns.

We analyzed Ethereum's Mainnet blocks to generate synthetic workloads and found that certain accounts (or smart contracts) are predominantly accessed, indicating that historical blocks exhibit a tail distribution in address access frequency. Based on this observation, we designed synthetic workloads that reflect varying characteristics of transaction conflicts in blocks. Following the tail distribution, we evaluated performance across synthetic workloads, including worst-case and best-case scenarios, enabling a comprehensive assessment of system behavior under varying levels of conflict within a block. For this, we added another metric on the X-axis, the parameter α, which represents the heaviness of the tail in the Pareto distribution [24]. The value of α determines the degree of conflict, as shown in subfigure (b); smaller values of α correspond to fewer conflicts.

1. ERC-20 Workload ($\mathbf{W}_{erc20}$): This workload simulates multiple ERC-20 smart contracts, with each contract representing a distinct cluster. Transfer transactions occur among addresses within each cluster, each of which has its own ERC-20 token. The number of transactions per cluster is determined using a Pareto distribution [24], while the sender and receiver addresses are selected uniformly within each cluster. This approach captures the distribution of transaction loads, resulting in certain clusters receiving higher transactions. The conflict specifications are derived using the sender, receiver, and contract addresses accessed by transactions in the block. For testing, we generate 10k ERC-20 transfers across 10k contract addresses (clusters), with each cluster mapped to a unique externally owned account (EOA) address. Since each contract has a single EOA, transactions are effectively self-transfers. However, an EOA may initiate multiple transactions within a cluster, potentially causing conflicts, with a tail factor of $\alpha = 0.1$.

As shown in Figure 3a, increasing thread count improves performance. In fact, dBTM achieves peak throughput at 24 threads with 411k tps, compared to 178k tps and 171k tps for oBTM and PEVM, with an average throughput of 299k, 157k and 143k tps, respectively. In particular, dBTM achieves a 4× speedup over sequential and a speedup of 2.4× over PEVM. Latency trends show that optimistic approaches initially reduce execution time, but later increase it due to higher abort rates. In contrast, dBTM minimizes aborts by leveraging conflict specifications, reducing both abort and (re-)validation costs. Similarly, in Figure 3b, with 16 threads, increasing block conflicts leads to a steady performance decline, which is more pronounced in optimistic approaches due to higher abort rates and increased waiting in dBTM. At $\alpha = 5$ (maximum conflicts), all approaches incur a higher overhead and perform worse than sequential execution.

2. Mixed Workload ($\mathbf{W}_{m}$): This workload combines 50% native ETH transfers and 50% ERC-20 transfers, ensuring a balanced representation of both types of transactions. For ETH transfers, sender and receiver addresses are selected using a Pareto distribution [24]. The number of ERC-20 contracts is set to 25% of the block size, with workload generation similar to W_{erc20}. As a result, 2.5k contracts (clusters) comprise a block of 10k transactions.

As shown in Figure 4, throughput and latency remain consistent with W_{erc20} workload. However, due to 50% ETH transfers (microtransactions) in the block, all approaches show a noticeable increase in throughput. The average tps is 356k for dBTM, 238k for oBTM, 229k for PEVM, and 157k for sequential. This translates to speedups of 2.27×, 1.52×, and 1.46× for dBTM, oBTM, and PEVM, respectively, over sequential execution. The maximum tps of dBTM increased from 411k to 454k.

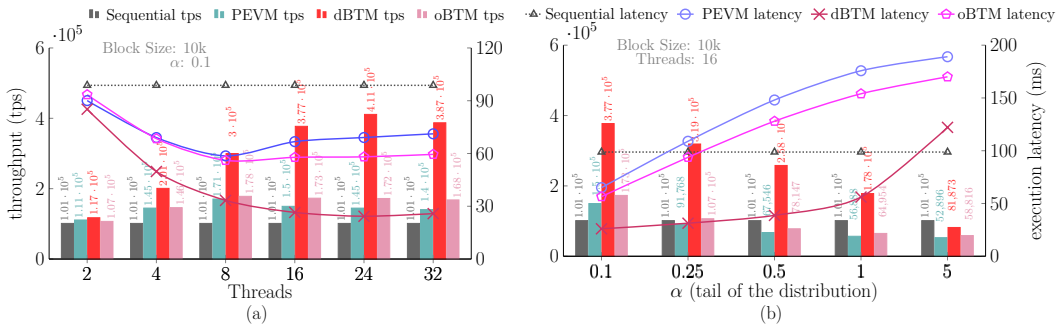


Figure 3 ERC-20 transfer workload (W_{erc20}).

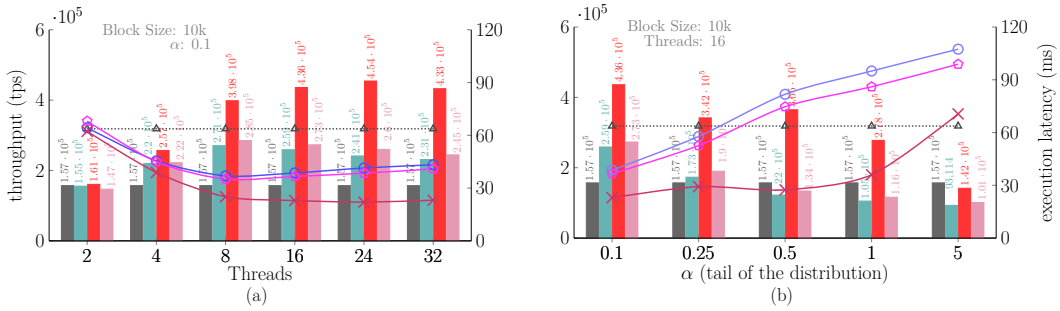


Figure 4 Mixed workload (W_m).

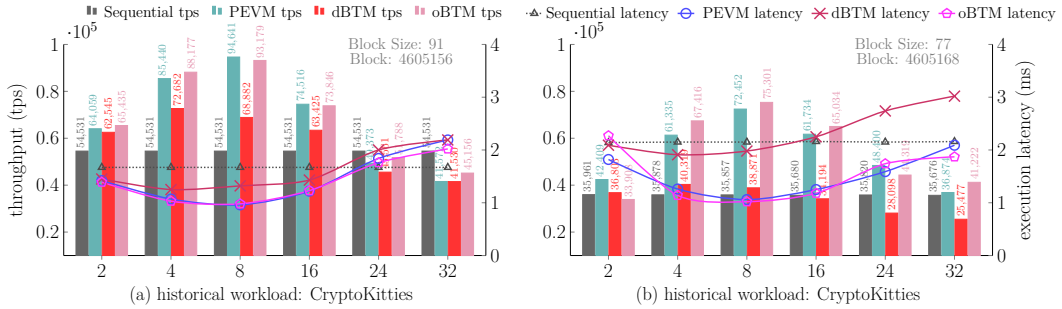
Testing on Real-World Ethereum Transactions

We selected blocks from different historical periods based on major events that may have impacted Ethereum's concurrency and network congestion. Each of these blocks allows us to analyze performance in different historical periods, providing insight into how major events, such as popular dApp launches and significant protocol upgrades, affect transaction throughput and network latency. Consequently, this also helps us understand the limitations of parallel execution approaches under different network conditions, such as the gradual increase in conflicts in the block and the change in resource requirements over time.

We trace the states accessed by transactions within a historical block using `callTracer` and `prestateTracer` APIs [9], which provide a complete view of the block's pre-state, the state required for executing the current block. To derive conflict specifications, the pre-state file is parsed to identify all EOAs and smart contract addresses accessed within the block. These addresses form the basis for constructing the conflict specification and serve as the "ground truth". Note that to derive specifications for a transaction T_k, all transactions (their pre-state EOAs and smart contract addresses) preceding it in preset order are considered.

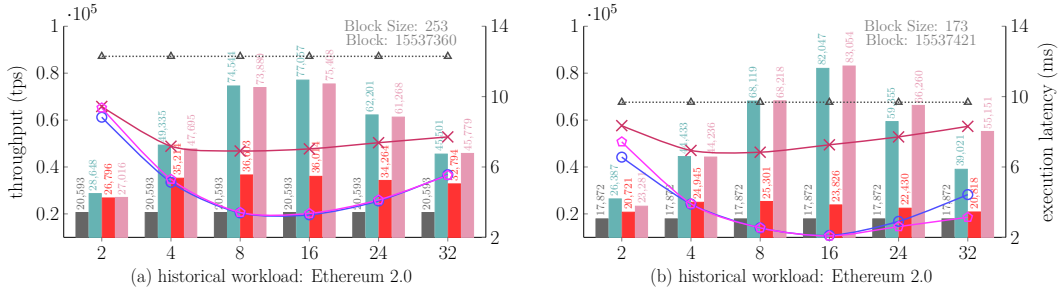
Note that real historical Ethereum blocks contain significantly fewer transactions than to our synthetic tests. To ensure a representative evaluation, we selected larger blocks from different historical periods. Specifically, we analyzed two blocks each from the CryptoKitties contract deployment and Ethereum 2.0 merge periods. We also analyzed two recent blocks with specific characteristics to demonstrate the worst-case and the best-case parallelism.

1. CryptoKitties Contract Deployment (W_{ck}): The famous CryptoKitties [10] contract was deployed on block 4605167, after which an unexpected spike in transactions caused Ethereum to experience high congestion. We analyze block 4605156, which occurred before the contract deployment, and block 4605168, which took place after the deployment.



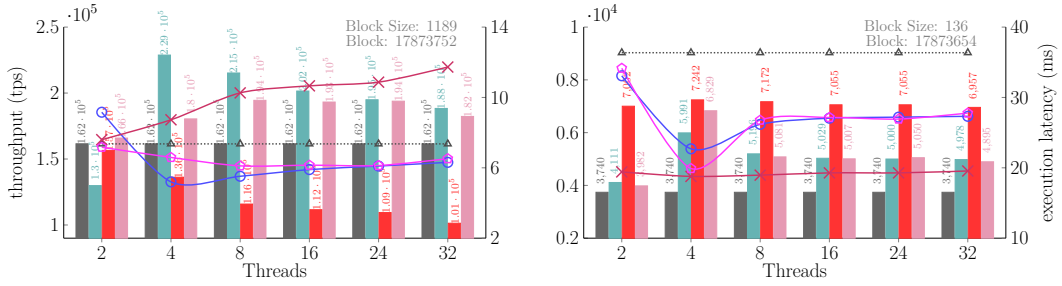
(a) historical workload: CryptoKitties (b) historical workload: CryptoKitties

Figure 5 CryptoKitties historical period (W_{ck}).



(a) historical workload: Ethereum 2.0 (b) historical workload: Ethereum 2.0

Figure 6 Ethereum 2.0 merge historical period (W_{e2}).



(a) historical workload: worst case (b) historical workload: best case

Figure 7 EVM: Historical blocks of worst and best-case (W_{wb}).

Figure 5a demonstrates a significant speedup in parallel execution, while Figure 5b highlights congestion in the block following contract deployment. In dBTM, conflicts are over-approximated due to sender and contract address-level conflict specifications, as real blocks lack read-write and contract storage access-level granularity. This leads to a performance drop compared to synthetic tests. With complete conflict specifications and larger blocks, dBTM could achieve superior performance, as observed in synthetic tests. That said throughput increases with additional threads, peaking at 8 threads, indicating the optimal point for speedup. Notably, oBTM outperforms other approaches in this workload both before and after contract deployment. The average tps for oBTM, PEVM, and dBTM drops from 69k, 68k, and 59k, before contract deployment to 54k, 53k, and 33k after deployment. Additionally, despite smaller blocks in the post-contract deployment, execution time increases for both sequential and other executors, highlighting the impact of congestion on overall performance.

2. Ethereum 2.0 Merge (W_{e2}): The merge [13] took place in block number 15537393, which changed Ethereum's consensus to proof-of-stake along with other protocol-level changes and had an impact on transaction processing, block validation and network traffic in general. We analyze block 15537360 before the merge and 15537421 after the merge. Compared to the W_{ck} historical period, the network had experienced an increase in block size and user activity. As shown in Figure 6, the average throughput for oBTM, PEVM, and dBTM before the merge was 55k (2.68× speedup over sequential), 56k (2.73× speedup), and 33k (1.63× speedup), respectively. Post-merge, these values changed to 56k (3.17× speedup), 53k (2.98× speedup), and 23k (1.29× speedup). Notably, the increased speedup over sequential post-merge shows the growing parallel execution potential to improve throughput.

3. Worst-case and Best-case Blocks (W_{wb}): To analyze performance extremes, we selected Ethereum blocks exhibiting different conflict profiles. The worst-case block, 17873752, represents maximal conflicts with densely interconnected transactions and consists of 1189 transactions, of which 1129 are ETH transfers, 21 are ERC-20 transfers, and 39 are other contract transactions. We observed that most ETH transfers interact with the same address, resulting in a longer dependency chain of length 1104. As a result, the block becomes inherently sequential, and hence a good choice for worst-case analysis. In contrast, the best-case block, 17873654, consists predominantly of independent transactions. There are a total of 136 transactions, including 41 ETH transfers, 18 ERC-20 transfers, and 77 other contract transactions. We chose this block because it has a significant number of internal transactions (435) and the length of the longest conflict dependency chain is 50.

As shown in Figure 7a, in the worst-case, for dBTM the tps remains consistently low, around 100k-137k across all thread counts and 166k-195k for oBTM. This reflects minimal improvement over the sequential tps of 162k, highlighting the limitation of parallel execution under blocks with high access-level dependencies. While PEVM at 4 threads shows 1.41× performance gains over sequential, increasing the threads beyond introduces aborts and re-executions, resulting in overhead. However, the performance gain could be due to the lazy update of ETH transfers, which defers balance changes to EOAs until a later stage at commit time, as well as to independent contract transactions that are computationally intensive compared to ETH transfers−microtransactions with minimal computational logic. Due to the overhead of DAG construction, dBTM shows a noticeable performance degradation. Moreover, 94.95% of the transactions in the block are ETH transfers resulting in smaller execution latency despite the large block size. The average execution latency for PEVM is 6.36 ms, oBTM is 6.45 ms, that are not significantly less compared to 7.36 ms of sequential execution, while dBTM incurs slightly higher latency 9.97 ms.

In contrast, the best-case block in Figure 7b shows that oBTM achieves throughput in the range of 3.9k-6.8k tps, while dBTM ranges from 6.9k-7.2k tps, both significantly outperforming 3.74k tps of the sequential execution. The average execution latency across threads is 27.1 ms for oBTM and 19.2 ms for dBTM, compared to 36.36 ms for sequential and 27.25 ms for PEVM. In terms of average speedup, oBTM achieves a 1.34× improvement over sequential and performs on par with PEVM, while dBTM shows superior efficiency, with an average speedup of 1.89× over sequential and a 1.42× speedup over PEVM. Since, this block contains a higher proportion of smart contract transactions with numerous internal calls, making it computationally intensive for the VM. Consequently, minimizing aborts and re-executions yields tangible performance benefits. Notably, dBTM maintains its competitive edge due to its reduced abort and re-execution overheads.

Observe that from one historical period to another (even with just these blocks), despite the block size increasing, the overall throughput has decreased, highlighting the impact of congestion and resource requirements on overall performance.

Summary. *Our EVM analysis highlights the strong performance of dBTM and oBTM, demonstrating their potential to achieve a near-theoretical maximum parallel execution by leveraging conflict specifications. With more accurate conflict specifications in historical workloads, we can expect to achieve even higher speedups over sequential and PEVM. However, in the absence of conflict specifications, oBTM is expected to perform the same as the optimistic execution of PEVM. On average, across synthetic workloads, dBTM achieves a maximum of* $4\times$ *speedup over sequential and* $3.82\times$ *over PEVM, while oBTM achieves* $1.8\times$ *and* $1.2\times$ *speedups, respectively. In contrast, across historical blocks, dBTM achieves an average speedup of* $1.22\times$ *over sequential and a maximum of* $1.7\times$ *over PEVM, while oBTM achieves an average of* $1.83\times$ *over sequential and a maximum of* $1.41\times$ *over PEVM.*

6 Block Transactional Memory for EVM and MoveVM

Having established the empirical advantages of BTM's leveraging conflict specifications, we now present the design of a *full conflict analyzer* integrated implementation of the EVM parallel execution. We demonstrate with the conflict analyzer that it is possible to algorithmically implement *sound* conflict specifications for real Ethereum blocks (cf. Section 6.1). The derived specifications, though might be *incomplete*, i.e., they may not identify conflict relations for some class of transactions, can be generated efficiently and thus prove consequential for maximizing transaction execution throughput (as established in Section 4.2). Additionally, we present results for a workload adaptive execution leveraging the conflict specifications for EVM in Section 6.3. We then present a limited set of performance numbers for conflict analyzer integrated BTM on the MoveVM in Section 6.4. The BTM implementation for MoveVM leverages a custom-built conflict analyzer that is more conservative than the EVM conflict analyzer because the Aptos Move memory model is quite different from EVM posing several challenges for a general specification derivation scheme. The detailed version of the MoveVM implementation and analysis is delegated to the online arXiv version.

6.1 Conflict Specifications for EVM

Preliminaries. We now detail some technical preliminaries needed to describe the approach for building conflict specifications in EVM.

▶ **Definition 2.** *Ethereum blockchain is made up of a set of* ***accounts*** $A = A_U \cup A_C$*, where* A_U *is the set of EOAs and* A_C *is the set of contract accounts. Each account* $a \in A$ *is associated with some data* $a.\texttt{balance}$ *such that* $a.\texttt{balance} \in a.\texttt{data}$ *where* $a.\texttt{balance}$ *is the ETH balance. For an EOA* $a \in A_U$*,* $a.\texttt{data} = \{a.\texttt{balance}\}$ *whereas a contract account usually has fields associated in* $a.\texttt{data}$*.*

▶ **Definition 3.** *An Ethereum transaction is specified by a tuple* $(\texttt{origin}, \texttt{dest}, \texttt{value}, \texttt{calldata})$ *where* $\texttt{origin} \in A_U$ *is the initiator of the transaction,* $\texttt{dest} \in A$ *is the transaction recipient,* $\texttt{value} \in \mathbb{N}$ *is the number of wei sent by* $\texttt{origin}$ *to* $\texttt{dest}$*,* $\texttt{calldata}$ *is the accompanying data. An Ethereum transaction* T *is a simple ETH payment if* $T.\texttt{dest} \in A_U$ *and is a contract transaction otherwise. For the read/write sets it always holds that Rset(T), Wset(T)* $\subseteq \cup_{a \in A} a.\texttt{data}$*.*

▶ Note 4. A real-world Ethereum transaction contains more elements, but for the purpose of overapproximating read/write sets here, such tuple is a sufficient enough characterization.

▶ **Definition 5.** *For a contract transaction T, the* ***signature*** *of T denoted by $T.\texttt{sig}$ is the first four bytes of $T.\texttt{calldata}$ that specify the function to call in the contract ($T.\texttt{dest}$).*

▶ **Example 6.** Consider a situation where an EOA $a \in A_U$ initiates a transaction T_1 calling the function `receiveMessage` of a *Contract* b (cf. Listing 1) so the corresponding tuple is: $(\texttt{origin} : a, \texttt{dest} : b, \texttt{value} : 1, \texttt{calldata} : \{\texttt{signature} : \texttt{receiveMessage}, \texttt{arg} : \text{“}hello\text{”}\})$. We can see here, that always $Rset(T_1) = \{a.\texttt{balance}, b.\texttt{balance}, b.\texttt{shouldAccept}\}$. Notice that whenever $T.\texttt{value} > 0$ we do have $T.\texttt{origin.balance}, T.\texttt{dest.balance} \in Rset(T)$, $Wset(T)$ as the value needs to be deducted from $T.\texttt{origin}$ and added to $T.\texttt{dest}$. But either $Wset(T_1) = \emptyset$ or $Wset(T_2) = \{b.\texttt{message}\}$ depending on the value of $b.\texttt{shouldAccept}$, because the message field is updated only if the `shouldAccept` value is true.

Now, let another EOA account c initiate a transaction T_2 with the corresponding tuple $(\texttt{origin} : c, \texttt{dest} : b, \texttt{value} : 0, \texttt{calldata} : \{\texttt{signature} : \texttt{setShouldAccept}, \texttt{arg} : \texttt{false}\})$. We have $Rset(T_2) = \emptyset$, $Wset(T_2) = \{b.\texttt{shouldAccept}\}$. We can see from the read and write sets characterized here that T_1, T_2 can never have a read-from conflict if T_1 precedes T_2 (regardless of the control flow in T_1) in the preset order. But if T_2 precedes T_1, then they will have read-from conflict since T_2 writes to $b.\texttt{shouldAccept}$ and T_1 reads from it.

Approach. The EVM conflict analyzer is just labeling simple payments as $SimplePayment$, the designated ERC-20 functions with their respective signature and everything else as $exitsContract$ in line 8. After the labeling, check if a transaction T_j can have read-from conflicts with some $T_i, i < j$, we consider multiple scenarios. If T_j is labeled $exitsContract$, we simply assume that they do have read-from conflict (line 18). If both T_i, T_j are labeled as one of the special ERC-20 transactions and interact with the same contract, it just checks to see if the sender and accounts in the arguments of the functions called for T_j are disjoint from those of T_i (line 25). This is enough as an ERC-20 contract only affects some mapping $(balances, allowances)$ in the contract storage with the keys affected being either the transaction arguments or the initiator ($T.origin$).

In all other cases, it is enough to check that the origin and destination of T_i is disjoint from those of T_j (line 30). Since both ERC-20 transactions and simple payments only affect the source and target of the transactions, this is enough to guarantee independence.

We now go over the Algorithm 3 for transactions interacting with *Token contract* and *Wallet contract*, as shown in Listing 1. Notice that every transaction that calls a function other than $Token.transfer$ in these two contracts will be labeled as $exitsContract$ and thus be assumed to conflict with everything else. Therefore, just consider an account a_1 initiating a transaction T_1 invoking $Token.transfer$ with address a_2 and another account b_1 initiating T_2 invoking $Token.transfer$ with address a_2. Here, both functions are labeled $transfer$. Now, to check whether they are independent, one needs to check whether all a_1, a_2, b_1, b_2 are distinct. Now, let T_3 be a simple payment of c_1 to c_3. Of course T_3 is labeled as $SimplePayment$. Similarly, one only needs to check the distinctness of c_1, c_2 with respect to each of $a_1, Token$ and $b_1, Token$ to see if any conflict arises.

Listing 1 Smart Contracts.

```
contract b {
    string message = "";
    bool shouldAccept = true;
    function receiveMessage(String message) external payable{
        if(shouldAccept){
            this.message = message;
        }
    }
    function setShouldAccept(bool shouldAccept) external {
        this.shouldAccept = shouldAccept;
    }
}
contract Token {
    mapping(address => uint) balances;
    address priceOracle;
    function transfer(address target, uint amount) external {
        require(tokenBalances[msg.sender]>=amount);
        balances[msg.sender] -= amount;
        balances[target] += amount;
    }
    function turnEtherToToken() external payable{
        (, bytes memory data) = priceOracle.staticcall(abi.encodeWithSignature("tokenPrice()"));
        uint tokenPrice = bytesToInt(data);
        balances[msg.sender] = balances[msg.sender] + (msg.value/tokenPrice);
    }
}
contract Wallet {
    mapping(address => uint) balances;
    function addToWallet() external payable {
        balances[msg.sender] += msg.value;
    }
    function withdraw(uint amount) external {
        require(balances[msg.sender]>=amount);
        balances[msg.sender] -= amount;
        msg.sender.transfer(amount);
    }
}
```

Algorithm 3 Output conflict specifications, given as input to Algorithms 1 and 2.

Input: A transaction T_j in a preset order $T_1 \rightarrow, \ldots, \rightarrow T_n$. **Output:** $S \subseteq \{T_i | i < j, Rset(T_j) \cap Wset(T_i) = \emptyset\}$.

Fun isERC-20 *(label)*:
  return $label \in \{transfer, transferFrom, approve\}$

Fun erc20Accounts *(T)*:
  // Get the affected accounts in an ERC-20 transaction, for instance, a *transferFrom()* involves *from*, *to*, and the *sender* address. Two ERC-20 transfers are independent if their affected account sets are disjoint.
  if $label \in \{transfer, approve\}$ **then**
    return $\{T.\texttt{origin}, T.\texttt{calldata.to}\}$
  else if $label = transferFrom$ **then**
    return $\{T.\texttt{origin}, T.\texttt{calldata.from}, T.\texttt{calldata.to}\}$

Fun getLabel *(T)*:
  if $T.dest \in A_U$ **then**
    return *SimplePayment*.
  $sig \leftarrow T.\texttt{sig}$
  if $sig \in \{transfer, transferFrom, approve\}$ *in ERC-20* **then**
    return $T.\texttt{sig}$
  else
    return *exitsContract*

Fun findCSet(T_j):
  $label_j \leftarrow \texttt{getLabel}(T_j)$
  if $label_j = exitsContract$ **then**
    return $\emptyset$
  $Output \leftarrow \emptyset$
  forall $1 \leq i < j$ **do**
    $label_i \leftarrow \texttt{getLabel}(T_i)$.
    if $T_i.origin = T_j.origin$ *or* $label_i = exitsContract$ **then**
      continue;
    if $T_i.dest = T_j.dest$ *and* isERC-20 $(label_i)$ *and* isERC-20 $(label_j)$ **then**
      $affectedAccounts_i \leftarrow \texttt{erc20Accounts}\ (T_i)$
      $affectedAccounts_j \leftarrow \texttt{erc20Accounts}\ (T_j)$
      if $affectedAccounts_i \cap affectedAccounts_j = \emptyset$ **then**
        $Output \leftarrow Output \cup \{T_i\}$
    else if $|\{T_i.origin, T_j.origin, T_i.dest, T_j.dest\}| = 4$ **then**
      $Output \leftarrow Output \cup \{T_i\}$
  return *Output*

▶ **Proposition 7.**

- *Whether T is a simple payment or calls a designated ERC-20 function, only writes to or reads from accounts $T.\texttt{origin}, T.\texttt{dest}$.*
- *If T calls an ERC-20 function in $transfer, transferFrom, approve$, then only $T.\texttt{dest}$ storage is affected. Assuming that the contract adheres to ERC-20 specifications, as long as $T.\texttt{origin}$ and accounts in $T.\texttt{data}$ are disjoint from another T' calling these specific*

functions in T.`dest`, then different parts of the same mappings in T.`dest` are modified. Also, if the contract is implemented as a proxy, we assume that only the proxy contract can call the parent (no EOA can initiate a transaction directly with the parent contract).

▶ **Theorem 8.** *If the Algorithm 3 outputs an independence set $\mathcal{T}$ for a transaction T_j, then for each $T \in \mathcal{T}$ in the preset order before T_j, $Rset(T_j) \cap Wset(T) = \emptyset$.*

Proof. Let $T \in \mathcal{T}$ in the preset order be before T_j. First note that by the algorithm, if T label is *exitsContract*, then it is overapproximated to have read-from conflict with every transaction before in the preset order. So we can assume T is either a simple payment or an ERC-20 transaction. Then (according to the definition of simple payment and Proposition 7 and line 8) T only modifies/reads from $\{T.origin, T.dest\}$ which is guaranteed to be disjoint from $\{T_j.\texttt{origin}, T_j.\texttt{dest}\}$ by line 30. Therefore, by line 8 and Proposition 7, T_j only modifies $\{T_j.\texttt{origin}, T_j.\texttt{dest}\}$ which is disjoint from the read set of T implying that T does not have a read-from conflict with T_j. ◀

6.2 Conflict Analyzer Integration with EVM for Parallel Execution

This section presents the analysis of *full* conflict analyzer integrated with our BTM algorithm for the EVM. We evaluated the performance of the proposed BTM algorithm on the EVM, leveraging conflict specification provided by the conflict analyzer.

Implementation Details. We introduced several implementation-level modifications to the oBTM, resulting in an extended version that we refer to as iBTM. It uses analyzer-provided dependency information (beyond the sender, receiver, and contract addresses used in oBTM) to reduce aborts, and applies the lazy update optimization (of PEVM) for native ETH transfers, while other transactions remain optimistically executed. We choose to perform tests with iBTM, as it required minimal changes to existing implementation and demonstrates the potential benefits of the conflict analyzer when integrated with the optimistic execution of oBTM. Note that for this analysis, we re-run the experiments in our integrated conflict analyzer setup, and the numbers in Section 5.2 can be treated as the ground truth. The entire implementation built on top of PEVM which is 8097 lines of code with 4454 lines added over the original PEVM implementation.

The conflict analyzer is implemented in approximately 700 lines of Rust code. It uses HashMap that maps each accessed address to a vector of transactions that have interacted with it. Transactions labeled as *exitsContract* are pruned in the first pass to reduce the number of pairs evaluated for independence as a transaction labeled *exitsContract* is assumed to conflict with all others. After that if the number of transactions remaining after that is high enough (a manually set parameter), parallelization is done across 16 threads. In addition to the implemented version of the conflict analyzer detailed in Section 6.1, we additionally implemented another *strong* version of the conflict analyzer. Both of these are automated completely and proved to be sound (but not complete) if correctly implemented (cf. Theorem 8). The implemented analyzer main limitation is its inability to deal with non ERC-20 contracts, marking every transaction to such contracts as dependent on all other transactions. Strong analyzer main limitation is that when a contract function calls another contract C not known statically (that is, the address of the C is determined by the input data of the transaction) one cannot know what sequence of blockchain state manipulations after that and thus such transactions are marked dependent on everything else. In the case that C and its payload are known statically, it is rather easy to add more rigorous specifications. There exist inputs which can make the analyzer derive no independence specification making

Table 1 Conflict analyzer integrated with iBTM for parallel execution in EVM; metrics from conflict analysis phase, block composition, and execution performance across different approaches.

		Ethereum Historical Blocks				Synthetic 10k Blocks	
	Metric	4605168 (CryptoKitties)	15537421 (Ethereum 2.0)	17873752 (Worst-case)	17873654 (Best-case)	ERC-20	Mix
Conflict Analysis Phase	Conflict Generation Time	18.314 µs	39.00 µs	106.79 µs	30.09 µs	6.06 ms	3.30 ms
	Specification Size	627	943	50706	1691	49658226	49824316
Block Stats	Block Size	77	173	1189	136	10000	10000
	ETH Transactions	32	33	1129	41	0	5000
	ERC-20 Transactions	9	11	21	18	10000	5000
	Other Contract Transactions	36	129	39	77	0	0
	Dependency Count	10	7	4	0	5683	3215
Average Execution Time (16 threads)	Sequential	2.107 ms	15.60 ms	7.21 ms	36.63 ms	114.43 ms	72.12 ms
	PEVM	1.32 ms	6.93 ms	6.03 ms	27.52 ms	96.75 ms	53.60 ms
	oBTM	1.24 ms	6.88 ms	5.99 ms	27.68 ms	88.67 ms	54.14 ms
	iBTM	1.23 ms	6.43 ms	5.43 ms	26.85 ms	79.42 ms	39.50 ms

the execution as slow as sequential. To be more precise, we analyze the ERC-20 contracts by their function signature, so a malicious contract not adhering to ERC-20 specifications may result in an unsound analysis, however, one can regularly inspect deployed contracts either manually or automatically to adhere to specifications and analyze only those marked to adhere to specifications (which can be done by current tools [14, 23]).

Testing on Synthetic Workloads. Table 1 presents comprehensive details of integrated experiments, including conflict analysis and execution latency on historical Ethereum and large synthetic blocks. The reported metrics capture statistics from the conflict analysis phase, including the time required to generate conflict specifications and their size, as well as block characteristics such as block size, transaction type distribution, and conflict dependency counts for block. Also report the execution time under three configurations: sequential, PEVM and iBTM (execution with 16 threads). The goal is to evaluate computational cost and execution efficiency in both historical and large synthetic blocks.

We choose two synthetic workloads from Ethereum analysis Section 5.2: ERC-20 and Mix workloads, with a block size of 10k transactions and a fixed conflict distribution factor $\alpha = 0.1$, while varying the thread count on X-axis. Figure 8 shows the advantage of the conflict analyzer in saving the abort and re-execution cost for optimistic execution. As the thread count increases, in both workloads, ERC-20 transfer (Figure 8a) and Mix (Figure 8b) a noticeable execution latency improvement can be seen for iBTM over PEVM, highlighting the benefits of leveraging conflict specifications for execution. The benefit is especially pronounced for the ERC-20 workload, which contains more uniform transaction patterns.

Testing on Historical Ethereum Blocks. Figures 8c–f illustrates the performance on Ethereum blocks. We select one block each from the CryptoKitties (same block as in Figure 5b) and the Ethereum 2.0 workloads (same block as in Figure 6b). We also analyzed performance on the worst and best-case blocks of the workload in Figure 7. As shown in Figure 8, with our conflict analyzer, read-write conflict detection improves the performance of iBTM, which can be clearly observed by comparing the relative performance of PEVM with iBTM. The performance of iBTM improves in almost all cases compared to PEVM and oBTM with a noticeable difference, and is expected to improve further with a fully optimized and mature conflict analyzer. This performance could be due to the reduced number of aborts and re-executions (re-validations) in the optimistic execution.

To conclude, *the proposed iBTM achieves up to a* 1.75× *speedup over sequential execution and a* 1.33× *speedup over PEVM, with an average speedup of* 1.24× *over PEVM in synthetic workloads. In historical workloads, it achieves a maximum speedup of* 2.43× *and* 1.14× *over sequential and PEVM, also consistently outperforms PEVM across all evaluated scenarios.*

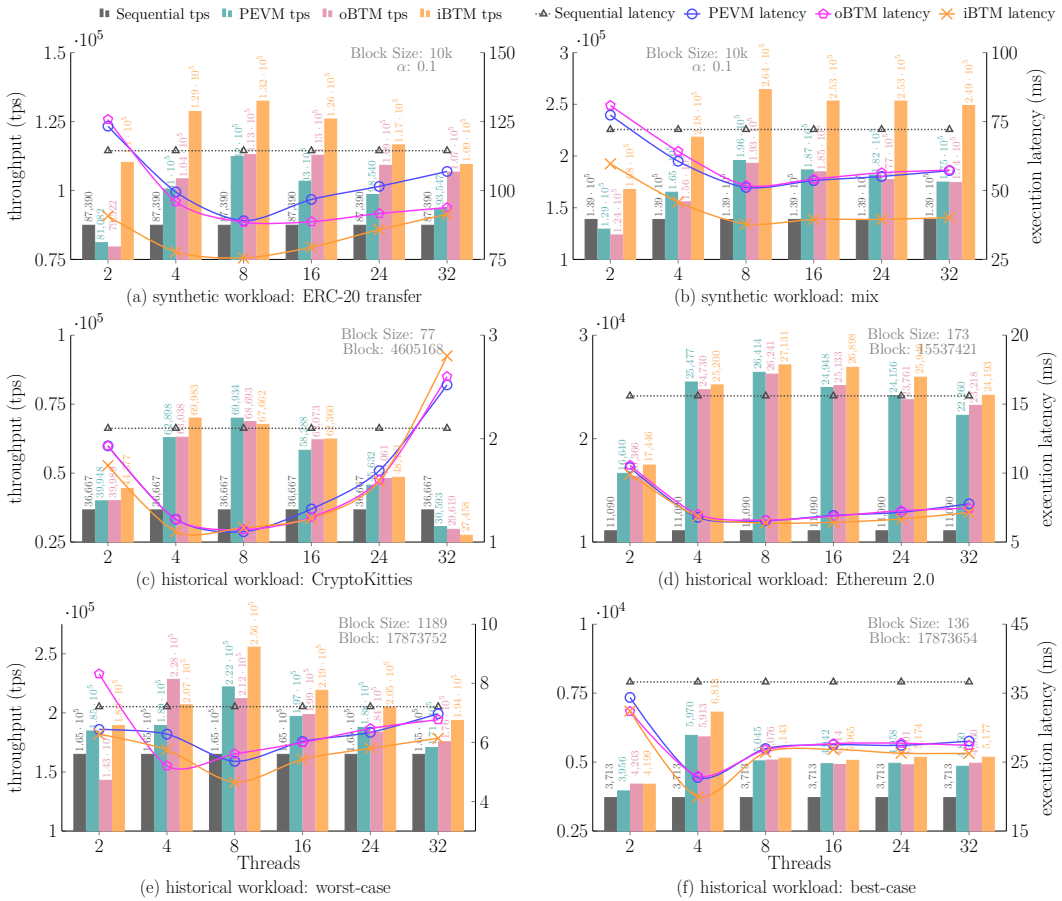
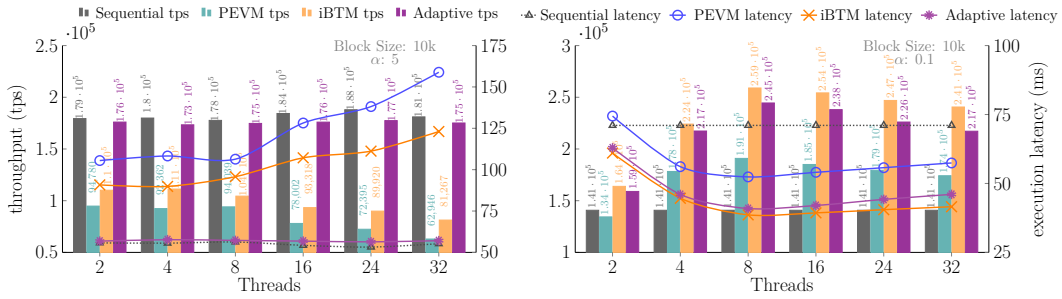


Figure 8 Conflict analyzer integrated with EVM: synthetic and historical workloads.

6.3 Adaptive Implementation Based on Conflict Threshold

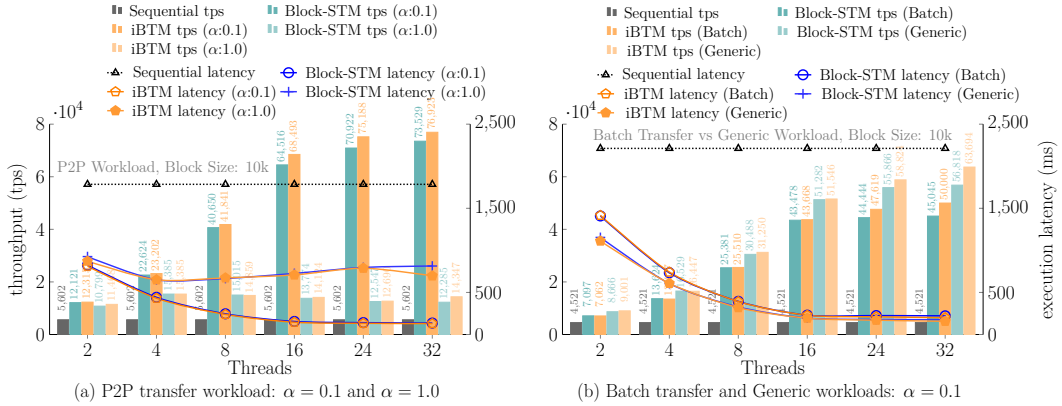
The advantage of the conflict specification is that it allows us to determine how many pairwise conflicts exist in a block of transactions. We demonstrate how we can leverage this for an adaptive implementation that can also handle high-conflict workloads (Figure 9a). Specifically, by computing a *conflict threshold* for each block with minimal overhead, we deterministically fall back to sequential execution in the case of high conflicts.

Figure 9 demonstrate the advantage of an adaptive technique that dynamically selects the most suitable execution path based on the characteristics of the workload. When $\alpha = 0.1$, the iBTM outperforms all other approaches, leading the adaptive technique to choose it for execution. In contrast, when $\alpha = 5$, the workload becomes highly sequential, and all parallel approaches perform worse than sequential execution. The adaptive approach could fall back to sequential execution; however, there could be small overhead with selecting the optimal approach; this is offset by significant execution time savings in highly conflicting workloads. The current adaptive mechanism is implemented and evaluated for two execution paths, sequential or iBTM, and considers the conflict threshold and block size for decision making. Exploring additional metrics beyond the conflict threshold and block size, such as available compute, expected block computation cost, and past n block execution statistics, as well as different adaptive parallel execution paths (e.g., dBTM, iBTM, PEVM, sequential), remains ongoing work and may constitute standalone future work.



(a) synthetic workload: worst-case (b) synthetic workload: best-case

Figure 9 Adaptive on EVM synthetic workloads.



(a) P2P transfer workload: $\alpha = 0.1$ and $\alpha = 1.0$ (b) Batch transfer and Generic workloads: $\alpha = 0.1$

Figure 10 Conflict analyzer integrated with MoveVM: P2P, batch transfer and generic workloads.

6.4 Conflict Analyzer Integration with MoveVM for Parallel Execution

Figure 10 presents the analysis of iBTM for MoveVM on synthetic workloads: P2P, batch, and a generic transfers that combine P2P and batch transfer in equal proportion. Figure 10 shows throughput and latency results for MoveVM across different synthetic workloads leveraging specifications generated by the conflict analyzer. We compared the performance of iBTM with the baseline sequential and the state-of-the-art Block-STM parallel execution while varying the thread count from 2-32, with a block size of 10k transactions. In the low-conflict P2P workload, Figure 10a, where $\alpha = 0.1$, iBTM scales from 12k-77k tps, reducing execution latency 3.20×. Block-STM achieves a comparable 73k tps at 32 threads. In high-conflict P2P, where $\alpha = 1.0$ (Figure 10a), throughput is modestly from 11k-14k tps. However, in Figure 10b, the batch transfer workload when $\alpha = 0.1$ exhibits better scalability, with iBTM increasing from 7k-50k tps, outperforming Block-STM's 47k tps. In the generic mixed workload with $\alpha = 0.1$ (Figure 10b) iBTM again scales well, improving from 9k-63k tps. Overall, iBTM consistently outperforms Block-STM and sequential execution, demonstrating the benefits of leveraging conflict specification for optimistic parallel execution.

7 Discussion and Concluding Remarks

The outstanding objective of this paper is to expose the significance of a execution methodology that prioritizes leveraging transactional conflict specifications as input for the parallel executor.

We demonstrated how this methodology can be constructive by implementing a state-of-the-art EVM parallel execution engine. We also demonstrated the methodology for MoveVM and its empirical benefits. We remark that we chose to exhibit this methodology for the read-write oblivious execution models of EVM and MoveVM when it would be more straightforward to implement our algorithms for the read-write aware execution model like Solana, since it does not require an explicit *conflict analyzer*. We leave that for a future iteration of the paper.

Localized Fee Markets. Our parallel execution methodology possibly lays the foundation for more localized dynamic *gas fee* marketplaces; if dependencies are specified a priori, transactions occurring in a congested contract of the blockchain state might be processed separately from others to prevent a localized state hotspot from increasing fees for the whole network. For example, a popular *non fungible token (NFT)* mint could create a large number of transaction requests in a short period of time [21]. A read-write aware blockchain can detect state hotspots upfront, such as the NFT minting event, to rate limit and charge a higher fee for transactions containing them [20, 34, 35]. This enables ordinary transactions to execute promptly, while transactions related to the minting process are prioritized independently based on the total gas associated with them and resulting congestion.

References

1 Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. Parblockchain: Leveraging transaction parallelism in permissioned blockchain systems. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1337–1347, Los Alamitos, CA, USA, July 2019. IEEE, IEEE Computer Society. doi:10.1109/ICDCS.2019.00134.

2 Parwat Singh Anjana, Hagit Attiya, Sweta Kumari, Sathya Peri, and Archit Somani. Efficient concurrent execution of smart contracts in blockchains using object-based transactional memory. In *Networked Systems*, pages 77–93, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-67087-0_6.

3 Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. An efficient framework for optimistic concurrent execution of smart contracts. In *27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 83–92. IEEE, IEEE Computer Society, February 2019. doi:10.1109/EMPDP.2019.8671637.

4 Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. Optsmart: a space efficient optimistic concurrent execution of smart contracts. *Distributed and Parallel Databases*, pages 1–53, 2022. doi:10.1007/s10619-022-07412-y.

5 Aptos Labs. The aptos blockchain: Safe, scalable, and upgradeable web3 infrastructure, August 2022. [August 11, 2022]. URL: https://aptosfoundation.org/whitepaper/aptos-whitepaper_en.pdf.

6 Shrey Baheti, Parwat Singh Anjana, Sathya Peri, and Yogesh Simmhan. Dipetrans: A framework for distributed parallel execution of transactions of blocks in blockchains. *Concurrency and Computation: Practice and Experience*, 34(10):e6804, 2022. doi:10.1002/cpe.6804.

7 Block-stm implementation. [Github, Accessed: January 2024]. URL: https://github.com/danielxiangzl/Block-STM.

8 BlueAlloy. Revm: A rust implementation of the evm, 2023. [Github, Accessed: January 2025]. URL: https://github.com/bluealloy/revm.

9 Chainstack. Ethereum traceBlockByNumber API Reference, 2025. [Docs, Accessed: January 2025]. URL: https://docs.chainstack.com/reference/ethereum-traceblockbynumber.

10 CryptoKitties. CryptoKitties Webpage, 2024. [Webpage, Accessed: January 2025, Contract Address: 0x06012c8cf97BEaD5deAe237070F9587f8E7A266d, Creation Transaction: 0x691f348ef11e9ef95d540a2da2c5f38e36072619aa44db0827e1b8a276f120f4]. URL: https://www.cryptokitties.co/.

11 Thomas Dickerson, Paul Gazzillo, Maurice Herlihy, and Eric Koskinen. Adding concurrency to smart contracts. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 303–312, New York, NY, USA, 2017. PODC '17, ACM. doi:10.1145/3087801.3087835.

12 Ethereum (ETH): open-source blockchain-based distributed computing platform. [Webpage, Accessed: January 2025]. URL: https://www.ethereum.org/.

13 Ethereum Foundation. Ethereum 2.0 Merge, 2022. [Blog, Accessed: January 2025]. URL: https://ethereum.org/en/upgrades/merge/.

14 Etherscan verifier, July 2025. [Website, Accessed: July 2025]. URL: https://etherscan.io/verifyContract.

15 Mohamed Fouda. The case for parallel processing chains, September 2022. [Blog, Accessed: January 2025]. URL: https://medium.com/alliancedao/the-case-for-parallel-processing-chains-90bac38a6ba4.

16 Sui Foundation. All about parallelization, January 2024. [Blog, Accessed: January 2025]. URL: https://blog.sui.io/parallelization-explained/.

17 Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Dahlia Malkhi, Yu Xia, and Runtian Zhou. Block-stm: Scaling blockchain execution by turning ordering curse to a performance blessing. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, PPoPP '23, pages 232–244, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3572848.3577524.

18 Rachid Guerraoui and Michal Kapalka. *Principles of Transactional Memory, Synthesis Lectures on Distributed Computing Theory*. Morgan and Claypool, 2010.

19 Petr Kuznetsov and Srivatsan Ravi. On the cost of concurrency in transactional memory. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 112–127, 2011. doi:10.1007/978-3-642-25873-2_9.

20 Helius Labs. Solana local fee markets: How they work and why they matter, 2024. [Blog, Accessed: May 2025]. URL: https://www.helius.dev/blog/solana-local-fee-markets.

21 What is lazy minting? introducing a smarter yet economical way to mint nfts, January 2024. [Blog, Accessed: January 2025]. URL: https://www.antiersolutions.com/what-is-lazy-minting-introducing-a-smarter-yet-economical-way-to-mint-nfts/.

22 Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

23 Mythx vulnerability coverage, July 2025. [Website, Accessed: July 2025]. URL: https://mythx.io/detectors/.

24 Pareto distribution, 2023. [Wikipedia, Accessed: January 2025]. URL: https://en.wikipedia.org/wiki/Pareto_distribution.

25 Manaswini Piduguralla, Saheli Chakraborty, Parwat Singh Anjana, and Sathya Peri. Dag-based efficient parallel scheduler for blockchains: Hyperledger sawtooth as a case study. In *European Conference on Parallel Processing*, pages 184–198, Berlin, Heidelberg, 2023. Springer, Springer-Verlag. doi:10.1007/978-3-031-39698-4_13.

26 Polygon Labs. Innovating the main chain: A polygon pos study in parallelization, 2024. [Blog, Accessed: May 2025]. URL: https://polygon.technology/blog/innovating-the-main-chain-a-polygon-pos-study-in-parallelization.

27 RISE Chain. RISE Chain Webpage, 2024. [Webpage, Accessed: January 2025]. URL: https://www.riselabs.xyz/.

28 RISE Labs. PEVM: Parallel Ethereum Virtual Machine, 2023. [Github, Accessed: January 2025]. URL: https://github.com/risechain/pevm.

29 Vikram Saraph and Maurice Herlihy. An empirical study of speculative concurrency in ethereum smart contracts. In *International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019)*, pages 4:1–4:15, Dagstuhl, Germany, 2019. OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.Tokenomics.2019.4.

30 Architecture documentaion v1.2. [Whitepaper, Accessed: January 2025]. URL: https://sawtooth.hyperledger.org/docs/1.2/architecture/transaction_scheduling.html.

31 Sei Labs. Sei Giga: Scaling the EVM through Parallel Execution, 2024. [Whitepaper, Accessed: May 2025]. URL: https://github.com/sei-protocol/sei-chain/blob/main/whitepaper/Sei_Giga.pdf.

32 Nir Shavit and Dan Touitou. Software transactional memory. In *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, pages 204–213, 1995. doi:10.1145/224964.224987.

33 Solana documentation. [Docs, Accessed: January 2025]. URL: https://docs.solana.com/.

34 Solana fees, part 1, December 2023. [Blog, Accessed: May 2025]. URL: https://www.umbraresearch.xyz/writings/solana-fees-part-1.

35 SolanaFloor. Solana's local fee market: A solution to soaring gas prices, 2024. [Blog, Accessed: May 2025]. URL: https://solanafloor.com/news/solanas-local-fee-market-a-solution-to-soaring-gas-prices.

36 Sui documentation: Discover the power of sui through examples, guides, and concepts. [Docs, Accessed: January 2025]. URL: https://docs.sui.io.

37 Umbraresearch. Lifecycle of a solana transaction. [Blog, Accessed: January 2025]. URL: https://www.umbraresearch.xyz/writings/lifecycle-of-a-solana-transaction.

38 Anatoly Yakovenko. Sealevel - parallel processing thousands of smart contracts, September 2019. [Blog, Accessed: January 2025]. URL: https://medium.com/solana-labs/sealevel-parallel-processing-thousands-of-smart-contracts-d814b378192.

Ticket to Ride: Locally Steered Source Routing for the Lightning Network

Sajjad Alizadeh
Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada

Majid Khabbazian
Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada

Abstract

Route discovery in the Lightning Network is challenging because senders observe only static channel capacities while real-time balances remain hidden. Existing locally steered schemes such as SpeedyMurmurs protect path privacy but depend on global landmark trees whose maintenance traffic and detours inflate latency and overhead.

We present *Ticket to Ride* (T2R), a locally steered source-routing framework that encodes the set of channels a payment *may* traverse into a compact *ticket* – an approximate-membership filter keyed with per-hop Diffie–Hellman secrets. Each relay learns only whether its *own* outgoing edges are permitted, yielding the same incident-edge privacy as SpeedyMurmurs while eliminating the need to build and maintain global landmark trees or any other shared routing state.

Extensive simulations on real snapshots – incorporating churn, silent shutdowns, and random channel saturation – show that T2R boosts end-to-end success by up to 9% and cuts median delay by 1.6× relative to SpeedyMurmurs, all with <1 kB total overhead and *no* extra handshakes. Because tickets are processed hop-by-hop and can be prefixed by a trampoline, T2R remains lightweight enough for resource-constrained IoT nodes.

2012 ACM Subject Classification Networks

Keywords and phrases Lightning Network, Source Routing, Approximate Membership Filters

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.30

Supplementary Material *Software (Source Code)*: `https://github.com/sajializ/ticket` archived at `swh:1:dir:2ce93a1800d60437bbe7f68216ba888ac79379c3`

1 Introduction

Payment channel networks (PCNs) such as the Lightning Network (LN) [14] scale blockchains by locking funds into two-party channels and allowing unlimited off-chain transfers that settle on-chain only when channels are opened or closed. By moving most activity off the base ledger, PCNs greatly increase throughput and lower fees, making cryptocurrency payments competitive with traditional systems.

When a payer lacks a direct channel to the payee, the payment must be forwarded through intermediaries. The public LN gossip protocol reveals the topology and the fixed *capacity* of every channel, yet keeps each side's *balance* private. Consequently, path finding is hard: a route that appears viable may fail at run time because an intermediate hop has insufficient outgoing liquidity.

All major LN implementations currently employ *source routing*, including LND [11], Core Lightning [3], and Eclair [1]. The sender uses a shortest-path search on the public graph, packs the resulting hop list into an onion (BOLT #4, the LN's onion routing specification) [10], and launches the payment. This approach leaves intermediaries passive and often requires repeated probing or guessing, so routing remains a bottleneck.

© Sajjad Alizadeh and Majid Khabbazian;
licensed under Creative Commons License CC-BY 4.0

The literature now recognizes three broad classes of routing algorithms: (i) algorithms that keep all control at the source, (ii) multi-path "streaming" schemes that split a payment across several precomputed paths, and (iii) *local algorithms* in which nodes along the path can steer the payment by choosing the next hop on the fly.

Prominent local algorithms include SilentWhispers [12] and SpeedyMurmurs [17]. In each, intermediaries consult local information to influence the forwarding direction, while the destination node has been kept concealed from them. Both SilentWhispers and SpeedyMurmurs share a common architectural premise: they construct an auxiliary overlay – rooted landmark-trees in SilentWhispers and VOUTE-style spanning trees in SpeedyMurmurs. More specifically, SpeedyMurmurs assigns every node a compact *coordinate* vector. Greedy forwarding then reduces to choosing a neighbor that monotonically improves this coordinate with respect to the destination. While SilentWhispers and SpeedyMurmurs are conceptually similar, SpeedyMurmurs refines the idea along several axes: it eliminates the heavy multi-party distance-vector computation of SilentWhispers, shrinks routing state, and achieves lower communication latency in micro-benchmarks and real-world snapshots [17]. Following the authors' empirical evidence that their embedding outperforms prior landmark routing both in success rate and end-to-end delay, we adopt SpeedyMurmurs as the representative local-steering baseline in this work.

Although SpeedyMurmurs achieves low latency and strong path privacy, it imposes a control-plane burden: multi-tree bootstrap floods, periodic stabilisation beacons, and per-tree state at every relay. Each drain or failure of a parent edge forces cascaded coordinate updates, and nodes adjacent to a landmark become fee and censorship choke points. Greedy forwarding on the embedding can also detour from shortest paths. These inefficiencies motivate a lighter local-steering scheme that carries *all* routing hints within each payment itself, removing global trees entirely.

In this paper, we introduce Ticket to Ride (T2R), a locally steered source routing framework that eliminates the need for global landmarks or coordinate trees. We implement this idea with a compact, cryptographically keyed *ticket*: a privacy-preserving set-membership structure that lists exactly which outgoing channels the sender allows. Concretely, the ticket is instantiated using an approximate-membership query (AMQ) filter – such as a Bloom filter or Golomb–Rice coded set – that trades a tunable false-positive rate for sub-kilobyte size. Upon receipt, a relay derives a one-hop secret from the sender's ephemeral public key, hashes each of its outgoing channels under that secret, and tests ticket membership. All channels that match are authorized; the relay then ranks them using a local policy – e.g., lowest cumulative fee if the sender flags cost sensitivity, or highest success probability – and forwards on the top candidate. Links invisible to the filter are ignored, and the keying prevents a relay from inferring which edges are authorized for other nodes. The ticket thus reveals neither the destination's identity nor the remaining route, guarantees monotonic progress toward the receiver, and eliminates the churn, choke points, and detour risk inherent in landmark-tree routing.

Forwarding depends only on per-hop membership tests, so a ticket remains useful even if its channel list is not up to date[1]: a vanished edge is silently skipped, and an unseen new edge merely foregoes a shortcut. In any case, channel openings and closures are already advertised through the routine gossip traffic that every node receives, so the information needed to refresh the filter circulates at no extra network cost. Because the ticket lists only

[1] Empirically, Lightning's topology evolves relatively slowly – channel openings and closures typically number in the hundreds per day [8], thus a snapshot taken approximately one day earlier usually differs from the current graph by roughly 1% of edges.

admissible edges, the sender can blacklist nodes or boost preferred channels – a level of control landmark trees cannot offer. The ticket piggybacks on the handshake message that neighboring relays already exchange, so it introduces no extra latency or bandwidth. If a probe fails, the sender drops the offending node and rebuilds the ticket; the new ticket omits that identifier and guarantees the payment will never be forwarded there again. Such explicit, sender-driven blacklisting is *infeasible* in landmark-tree and other purely local algorithms, where each intermediate node selects its next hop autonomously and the sender has no way to forbid a particular relay.

SpeedyMurmurs is appealing for lightweight nodes, as it shifts mapping responsibilities away from the sender, allowing them to remain passive. However, our ticket-based approach can also accommodate resource-constrained endpoints – such as IoT sensors or mobile wallets that frequently remain offline. Specifically, we propose a lightweight variant called *segment tickets with trampolines*, exploiting the draft trampoline extension [5]. Here, the sender delegates topology management by selecting a short chain of always-online trampoline nodes, each responsible for crafting fresh tickets for their respective segments. Thus, trampolines shoulder the workload of maintaining updated channel filters, making this mode particularly suitable when *both* sender and receiver have limited memory or infrequent connectivity to process topology updates themselves.

As a final contribution, we validate the design through extensive, trace-driven simulations on recent LN snapshots. Across tens of thousands of random payments, T2R achieves consistently higher success probability and lower end-to-end delay than SpeedyMurmurs, with the margin widening in the most demanding scenarios: long source–destination pairs, heavily saturated channels, and bursty link churn. These results confirm that removing landmark trees not only lightens the control plane but also yields tangible data-plane gains in practical, large-scale topologies such as LN.

2 System Model

Overlay graph. We represent the payment-channel network as an undirected multigraph $G = (V, E)$. Each vertex is a node identified by a long-term public key, and each edge $e = (u, v) \in E$ denotes a bidirectional channel of fixed capacity $C_{uv} \in \mathbb{N}$. The directional balances $b(u \to v)$ and $b(v \to u)$ are private, dynamic values that satisfy $b(u \to v) + b(v \to u) = C_{uv}$.

Gossip and topology freshness. Channel openings, closures, and capacity changes are disseminated by the ordinary LN gossip protocol and reach every honest node within Δ_{gossip}. Our protocol reuses this information and introduces *no additional control-plane traffic*, contrasting with the tree-stabilisation beacons required by SpeedyMurmurs.

Communication model. Nodes exchange authenticated, FIFO messages. Delivery latency is bounded by an unknown constant Δ; liveness uses this bound, while security arguments remain delay-agnostic.

Adversary. A static Byzantine adversary corrupts any subset $V_{\mathcal{A}} \subset V$ with $|V_{\mathcal{A}}| \leq f|V|$ for constant $f < 1$, mirroring the threat model of SpeedyMurmurs. Corrupted nodes may drop, delay, or modify messages and deplete balances but cannot break standard cryptography.

Payments and tickets. To transfer an amount x from a source s to a destination d, the sender attaches a cryptographically keyed *ticket* τ to the payment packet. The ticket τ is a privacy-preserving approximate-membership filter that encodes precisely the set $\mathcal{L}_{s,d} \subseteq E$ of edges the sender authorizes for forwarding.

Local forwarding rule. When a relay u receives $\langle x, \tau \rangle$, it derives a one-hop secret s_u from the sender's ephemeral public key, hashes each adjacent edge (u, v) under that secret, and tests membership in τ. Edges that match are ranked by a local policy (e.g. lowest cumulative fee or highest empirical success rate); the highest-ranked authorized edge is chosen as the next hop. No parent pointers or coordinate updates are maintained.

3 T2R Routing

In T2R , the sender appends to each payment a compact probabilistic set-membership filter – called a *ticket* – that, under a per-payment secret key, encodes the set $\mathcal{L}_{s,d}$ of channels the sender authorizes. The ticket is piggy-backed on the first control frame exchanged by neighboring nodes. Upon receipt, a relay hashes every outgoing channel with the secret, tests ticket membership, ranks the authorized matches according to its local policy (e.g., lowest fee or highest success probability), and forwards the payment on the top-ranked edge.

Despite its modest implementation footprint, the mechanism is expressive: it can emulate SpeedyMurmurs without constructing landmark trees. A node locally builds the trees, collects the union of edges that greedy forwarding might traverse, and inserts those identifiers into the ticket. The converse does not hold – landmark embeddings cannot realize sender-defined blacklists or per-edge priorities without redesigning the entire coordinate system.

In the remainder of this section, we first present a generic framework for constructing tickets, then formalize our privacy target – *incident-edge privacy* – and prove that any ticket generated by this framework satisfies it. We next introduce *TwinTicket*, a two-filter refinement that shrinks the filter size required to reach a given success rate.

3.1 Ticket Design

Our *ticket framework* comprises two algorithms: Issue, executed by an *originator* (typically the sender), and Open, executed by each intermediate node. The framework relies on two standard primitives:

- A hash function $H : \{0,1\}^* \to \{0,1\}^\kappa$, modelled as a random oracle and used solely as a source of pseudorandom bits.
- An **approximate-membership query structure (AMQ)** that supports three polynomial time operations:
  - Create(m, α): initializes an empty filter sized for up to m insertions with false-positive probability at most α;
  - Insert(S, e): adds element e to filter S;
  - Query(S, e): returns `present` with probability $\leq \alpha$ if $e \notin \mathsf{S}$, and never returns `absent` for an inserted element.

For each node u controlling an authorized channel, the originator derives a unique per-hop secret using elliptic-curve Diffie–Hellman (ECDH). Specifically, the originator selects an ephemeral public key $\mathrm{PK}_{\mathsf{ep}}$, while each node u has a long-term public key PK_u published through the network gossip. The shared per-hop secret is computed as:

$$s_u \leftarrow \mathrm{DH}(\mathrm{PK}_{\mathsf{ep}}, \mathrm{PK}_u),$$

where DH denotes an ECDH key exchange, followed by a standard key derivation function (e.g., HKDF) to yield a uniform κ-bit key. Because the sender knows the ephemeral secret and node u holds the private key corresponding to PK_u, only these two parties can compute the shared secret s_u. Consequently, any ticket entries derived from s_u appear indistinguishable from random to all other nodes. This per-hop shared-secret derivation mirrors the mechanism already employed by LN's onion-routing protocol [10].

Issue Algorithm

Algorithm 1 describes the ticket issuance process. The algorithm accepts as input:

(i) a set of authorized channels $\mathcal{C} \subseteq E$,
(ii) the public keys $\{\mathrm{PK}_u\}$ of nodes incident to these channels,
(iii) an ephemeral public key $\mathrm{PK}_{\mathsf{ep}}$ chosen by the sender,
(iv) a target false-positive probability ϵ.

The algorithm initializes an AMQ structure sized according to $|\mathcal{C}|$ and ϵ. For each channel $(u, v) \in \mathcal{C}$, the algorithm computes the per-hop secret s_u and inserts the hash value $H(v \parallel s_u)$ into the AMQ. The resulting AMQ structure is output as the ticket τ.

Algorithm 1 $\textsc{Issue}(\mathcal{C}, \{\mathrm{PK}_u\}, \mathrm{PK}_{\mathsf{ep}}, \epsilon)$.

$\mathsf{S} \leftarrow \mathrm{AMQ.}\textsc{Create}(|\mathcal{C}|, \epsilon)$
for all $(u, v) \in \mathcal{C}$ **do**
  $s_u \leftarrow \mathrm{DH}(\mathrm{PK}_{\mathsf{ep}}, \mathrm{PK}_u)$
  $\mathrm{AMQ.}\textsc{Insert}(\mathsf{S}, H(v \parallel s_u))$
end for
return S // *ticket* τ

Open Algorithm

Upon receiving a payment, an intermediate node executes Open to determine whether any of its outgoing channels are authorized by the ticket τ and, if so, to select one for forwarding. The algorithm takes as input:

(i) the ticket τ (AMQ structure) carried with the payment,
(ii) the node's own long-term public/secret key pair $(\mathrm{PK}_u, \mathrm{SK}u)$,
(iii) the sender's ephemeral public key PKep (included in the packet),
(iv) a ranking policy $\textsc{Rank}(\cdot)$ (e.g., lowest fee or highest empirical success rate).

First, the node derives its per-hop secret $s_u \leftarrow \mathrm{DH}(\mathrm{PK}_{\mathsf{ep}}, \mathrm{PK}_u)$. For each outgoing channel $(u, w) \in \mathrm{Adj}(u)$ it queries the ticket using the tag $H(w \parallel s_u)$. The set of channels for which the query returns `present` constitutes the authorized subset $\mathcal{A}_u$. If $\mathcal{A}_u$ is empty the node aborts the payment locally; otherwise it ranks $\mathcal{A}_u$ with Rank and forwards the payment along the highest-ranked edge.

Algorithm 2 $\textsc{Open}(\tau, \mathrm{PK}_{\mathsf{ep}}, \mathrm{PK}_u, \mathrm{SK}u)$.

$s_u \leftarrow \mathrm{DH}(\mathrm{PKep}, \mathrm{PK}_u)$
$\mathcal{A}_u \leftarrow \emptyset$
for all $(u, w) \in \mathrm{Adj}(u)$ **do**
  $y \leftarrow H(w \parallel s_u)$
  if $\mathrm{AMQ.}\textsc{Query}(\tau, y) ==$ `present` **then**
    $\mathcal{A}_u \leftarrow \mathcal{A}_u \cup (u, w)$
  end if
end for
if $\mathcal{A}_u = \emptyset$ **then**
  abort // *no authorized edge*
else
  $(u, w^\star) \leftarrow \textsc{Rank}(\mathcal{A}_u)$
  forward payment over $(u, w^\star)$
end if

The choice of RANK is intentionally left flexible. At a minimum, any channel (u, w) whose local balance cannot cover the payment amount is excluded from consideration. Beyond that filter, RANK can implement a variety of policies: it may prefer the lowest-fee edge, adopt the heuristics proposed in Flare [16], or emulate Spider-style congestion metrics [19]. Nevertheless, because each node independently selects its forwarding edge based solely on local information, the resulting route need not be globally optimal. Designing an effective ranking strategy is orthogonal to the ticket mechanism itself and remains open for future exploration. In our evaluation, we simply select uniformly at random among the authorized edges with sufficient balance.

Incident-edge privacy

A well-formed ticket must limit what any relay can infer about the remainder of the route. Intuitively, an intermediate node u should learn only whether *its own* channels are authorized and nothing about edges it does not control, beyond the unavoidable disclosure of the ticket's overall size. We capture this requirement with the indistinguishability notion below. Let τ denote the real ticket for a payment (s, d, x) and let $\tilde{\tau}_u$ be a *semi-ticket* that keeps u's authorized edges intact while masking every other authorized edge by an independent random λ-bit string, so that τ and $\tilde{\tau}_u$ contain the same number of inserts. A ticket scheme satisfies *incident-edge privacy* if no probabilistic polynomial-time adversary controlling u can distinguish τ from $\tilde{\tau}_u$ with more than negligible advantage in the security parameter λ.

▶ **Definition 1** (Incident-edge privacy). *Let τ be the ticket issued for a payment (s, d, x) and fix an intermediate node $u \in V$. Construct a* semi-ticket *$\tilde{\tau}_u$ in two steps:*

(1) *Insert into an empty AMQ every authorized channel incident to u.*

(2) *Insert uniformly random κ-bit strings in place of all other authorized channels, so that τ and $\tilde{\tau}_u$ contain the same number of inserts.*

A ticket scheme satisfies incident-edge privacy *if, for every probabilistic polynomial-time adversary A controlling only node u,*

$$\Big|\Pr[A(\tau) = 1] - \Pr[A(\tilde{\tau}_u) = 1]\Big| \;\le\; \mathrm{negl}(\kappa),$$

where the probabilities are over the randomness of ticket generation and of A, and $\mathrm{negl}(\kappa)$ *is negligible in the security parameter κ.*

The definition formalizes the guarantee that, apart from its own adjacent channels and the ticket's length, an intermediate node gains no information about which other edges the sender has authorized.

▶ **Theorem 2** (Incident-edge privacy of the ticket framework). *Assume the hash function H is modeled as a random oracle and the Diffie–Hellman key exchange is a secure key-agreement protocol. Then every ticket generated by the* ISSUE *algorithm in our framework satisfies the incident-edge privacy property of Definition 1.*

Proof sketch. Fix an intermediate node $u \in V$ and a probabilistic polynomial-time distinguisher A that corrupts u. Let κ denote the global security parameter, and let $q_H(\kappa)$ bound the number of random-oracle queries that A may issue.

Game G_0 (real world). The challenger runs ISSUE to generate the genuine ticket τ and hands $(\tau, \mathrm{PK}_{\mathsf{ep}})$ to A, which can query the oracle before outputting a bit.

Game G_1 (ideal world). As in G_0, except that for every authorized channel $(u', v') \notin \mathrm{Adj}(u)$ the challenger replaces the tag $H(v' \| s_{u'})$ in τ with an independently uniform κ-bit string. The resulting structure is exactly the semi-ticket $\tilde{\tau}_u$ of Definition 1.

Indistinguishability of G_0 and G_1. The two games differ only in tags for non-adjacent channels. For such an edge, the real tag in G_0 is

$$T = H\big(v' \| s_{u'}\big), \quad s_{u'} = \mathrm{DH}(\mathrm{PK}_{\mathsf{ep}}, \mathrm{PK}_{u'}).$$

Because A lacks the secret key of $\mathrm{PK}_{u'}$, the shared secret $s_{u'}$ is computationally indistinguishable from a uniform κ-bit string. In the random-oracle model, unless A queries H on the exact pre-image $v' \| s_{u'}$, the value T is random. Replacing T with an independent random string, as in G_1, therefore changes A's view only if it guesses $v' \| s_{u'}$.

The probability that A hits any specific κ-bit suffix within at most $q_H(\kappa)$ oracle queries is $q_H(\kappa)\, 2^{-\kappa} = \mathrm{negl}(\kappa)$. A union bound over all non-adjacent channels preserves negligibility, so

$$\big|\Pr[A^{G_0}(1) = 1] - \Pr[A^{G_1}(1) = 1]\big| \le q_H(\kappa)\, 2^{-\kappa} = \mathrm{negl}(\kappa).$$

Because the adversary's advantage is negligible, every ticket produced by Issue satisfies Definition 1. ◀

3.2 TwinTicket

The basic ticket of Section 3.1 stores all authorized channels in a *single* AMQ. Improving its accuracy normally means lowering the false-positive rate, which in turn expands the filter. TwinTicket takes an alternative path: it embeds two AMQs – one that *includes* authorized channels and a second that explicitly *excludes* those few channels misclassified by the first. In other words, TwinTicket contains

- an **inclusion filter** $\mathsf{S}^{\mathsf{inc}}$, which holds every authorized edge $(u, v) \in \mathcal{L}_{s,d}$; and
- an **exclusion filter** $\mathsf{S}^{\mathsf{exc}}$, which holds each false positive discovered when the sender queries $\mathsf{S}^{\mathsf{inc}}$ locally. A channel (u, v) is inserted in encoded form $H\big(v \| s_u\big)$, identical to the inclusion encoding.

Forwarding rule. A relay u first tests every adjacent edge against $\mathsf{S}^{\mathsf{inc}}$. If the result is `present`, the relay performs a second test against $\mathsf{S}^{\mathsf{exc}}$; the edge is deemed authorized only if it is *not* found there. The relay then ranks all authorized edges using its local policy and forwards on the highest-ranked candidate.

Benefits. Both the single-filter ticket and the double-filter TwinTicket grow in size as the target false positive reduces. Simulations in Section 5 show that TwinTicket achieves the same success probability and delay as a finely tuned single AMQ while using fewer bits. *This space saving does not weaken privacy.* Because every unauthorized edge is still added – again in hashed form – only by its own endpoint, TwinTicket preserves incident-edge privacy: an intermediate node u learns solely whether each of *its* channels is allowed or blocked, now with higher classification accuracy.

4 Overhead

Unlike SpeedyMurmurs, T2R dispenses with landmark trees altogether, yet it can reproduce SpeedyMurmurs' forwarding behavior – simply insert into the ticket every edge the payment is authorized to traverse. The reverse is impossible: landmark trees cannot (i) exclude specific channels or nodes, (ii) delegate route selection to third parties (e.g. the receiver), or (iii) guarantee shortest-path delivery without rebuilding the tree.

The price we pay is the *ticket*: it must be (i) generated by an originator and (ii) carried once with the payment. This section quantifies the computational, and communication overhead of those two steps. We start with analyzing the ticket size.

4.1 Ticket Size

The size of a ticket – i.e., the bit length of the AMQ structure that travels with each payment – depends on three parameters:

(i) the particular *AMQ design* employed;
(ii) the target false-positive bound α;
(iii) the number of elements inserted, m.

The literature offers a rich family of AMQ designs, each with its own space–functionality trade-off. Classic examples include the Bloom filter [4]; the counting Bloom filter, which supports deletions [7]; the Cuckoo filter [6]; and the Quotient filter [2]. Golomb–Rice coded sets (GCS) – used in Bitcoin's compact block filters – achieve near-optimal compression for static sets by entropy-coding sorted hashes [15]. More recent proposals, such as the Learned Bloom filter [9], trade a small probability of false negatives for additional space savings. For fixed set size m and target false-positive rate α, structures like the Cuckoo filter, quotient filter, or GCS can be more compact than the original Bloom filter but may incur higher insertion cost, require sorted input, or demand more intricate parameter tuning.

For example, the classic *Bloom filter* – introduced by Bloom in 1970 – supports only two operations, Insert and Query, and provides no deletion primitive. With m insertions and target false-positive probability α, its information-theoretically optimal length is

$$\ell_{\text{Bloom}} \;=\; \left\lceil -\frac{m\,\ln\alpha}{(\ln 2)^2} \right\rceil \;\approx\; 1.44\,m\,\log_2\!\big(1/\alpha\big)\ \text{bits}, \tag{1}$$

obtained with $k = \lceil (\ell_{\text{Bloom}}/m)\,\ln 2 \rceil$ independent hash functions [4].

A second widely deployed AMQ is the *Golomb–Rice Coded Set* used in Bitcoin block filters (BIP 158) [15]. A GCS stores the sorted hash values and compresses their gaps with Golomb–Rice coding, again supporting Insert and Query but requiring no random oracle. Choosing the Golomb–Rice parameter $P = \lceil \log_2(1/\alpha) \rceil$ yields a false-positive rate at most α and an expected length

$$\ell_{\text{GCS}} \;=\; m\,(P+1) \;=\; m\big(\lceil \log_2(1/\alpha) \rceil + 1\big)\ \text{bits}. \tag{2}$$

For the practical false–positive window $\alpha \in [2^{-7},\, 2^{-15}]$ ticket overhead remains modest. In a GCS the expected length per element is $P+1$ bits, where $P = \lceil \log_2(1/\alpha) \rceil$. Thus each authorized channel costs 8 bits when $\alpha = 2^{-7}$ and 16 bits when $\alpha = 2^{-15}$ – only 1–2 bytes apiece. By contrast, a Bloom filter needs $\ell_{\text{Bloom}} \approx 1.44 \log_2(1/\alpha)$ bits per element: about 10 bits (≈ 1.3 B) at $\alpha = 2^{-7}$ and 22 bits (≈ 2.7 B) at $\alpha = 2^{-15}$. The dominant contributor to ticket length is therefore the element count m: in the practical range $\alpha \in [2^{-15}, 2^{-7}]$ each additional authorized channel adds between $c{=}1$ and $c{=}2$ bytes.

If the ticket contains only the channels of a *single* source-to-destination path, our framework reduces to the single-route source routing that all major Lightning implementations already employ. Adding more channels, however, gives intermediate nodes additional "steering" options and thus lowers the probability that an in-flight payment stalls on an exhausted link.

A natural choice is to populate the ticket with the union of edges that lie on *any* shortest path between the sender and the receiver. Although many shortest paths can coexist, the ticket remains compact: on a recent public Lightning snapshot, authorizing about 126 channels suffices for 90% of randomly chosen source–destination pairs, and 611 channels cover 99% of such pairs.

4.2 Ticket Generation

Empirical measurements show that the public Lightning graph is remarkably stable: the average channel lifetime is nearly two hundred of days [21]. Because these modest topology updates propagate through the standard gossip protocol, a well-provisioned node can maintain an up-to-date view at virtually no extra cost. Building a ticket therefore needs only this topology snapshot; rapidly fluctuating balances are irrelevant.

Ultra-constrained senders (IoT tags, single-board wallets) cannot hold the full graph. We therefore combine our ticket mechanism with *trampoline payments* and slice the route into three *segment tickets*:

(a) $\tau_{S \to T_1}$ **(sender-issued)** covers the first few hops from the sender S to the entry trampoline T_1 and is keyed with an ephemeral $\mathrm{PK}_{\mathrm{ep}}^{(1)}$ chosen by S.

(b) $\tau_{T_1 \to T_k}$ **(macro ticket, sender-issued)** encodes the ordered list of trampoline IDs $T_2, \ldots, T_k$, where T_k is supplied by the receiver (see below). Each trampoline T_i reads the next ID, computes its own micro-path to T_{i+1}, and attaches a fresh sub-ticket.

(c) $\tau_{T_k \to R}$ **(receiver-issued)** steers the last one or two hops from T_k to the receiver R, optionally biasing the route toward R's best-funded inbound channel. It is keyed with a second ephemeral $\mathrm{PK}_{\mathrm{ep}}^{(2)}$ chosen by R.

This approach brings the following advantages:

- **Compactness.** The sender and receiver tickets span only a small number of hops (typically one or two) and the macro ticket is merely two or three compressed pubkeys (< 70 B). Total header overhead, therefore, stays well under one kilobyte.
- **Balanced computation.** Lightweight senders route only to T_1; each trampoline solves a local micro-routing instance; only trampolines need the global graph.
- **Privacy.** Every ticket is keyed with a fresh Diffie–Hellman secret for each hop, so an intermediate node learns only whether *its* outgoing edges are permitted. All other tags look uniformly random, achieving the same incident-edge privacy as the original T2R.
- **Fine-grained channel control.** When constructing the final segment of the ticket, the receiver can privilege high-liquidity inbound channels or blacklist unreliable ones. Because this choice is encoded locally in the ticket and masked by the per-hop key, neither the sender nor any trampoline node learns which channels were favored.
- **Minimal invoice overhead.** When R responds to an `invoice_request` (BOLT #12) or encodes a BOLT #11 invoice, it appends the triple $\left(T_k, \mathrm{PK}_{\mathrm{ep}}^{(2)}, \tau_{T_k \to R}\right)$. The extra payload is ≤ 100 bytes for the ticket plus 33 bytes for the key – well below the 1 kB capacity of a level-10 QR code – so invoice usability is unchanged.

With this three-ticket trampoline design, even kilobyte-scale devices can therefore utilize locally steered, privacy-preserving routing without storing the complete Lightning topology.

4.3 Communication and latency overhead

When an intermediate node u forwards a payment over its channel to peer v, it engages in the standard Lightning commitment handshake. The procedure consists of two full-duplex rounds:

(1) u sends an `update_add_htlc` (carrying the onion payload) immediately followed by `commitment_signed`.
(2) After validating the update, v replies with `commitment_signed` and `revoke_and_ack`, completing the commit–reveal exchange.

The ticket is piggy-backed on the first `update_add_htlc`; no extra round trips are introduced, and only a few hundred bytes are appended to a frame that already averages ∼1500 bytes. Importantly, v need not wait for the entire ticket to arrive before it can respond: the Lightning spec allows a node to send its `commitment_signed` as soon as the new Hashed Timelock Contracts (HTLC) has been parsed and validated, while the remaining payload bytes stream in.

▶ **Example 3** (Latency impact)**.** Assume a single Lightning hop with a round-trip propagation delay of 80 ms. Because a commitment update consists of two such round trips, the propagation component alone contributes 160 ms to handshake latency.

Typical ticket ($m = 125$, $\alpha = 0.001\%$). For a GCS filter with $\alpha = 0.00001$ the Golomb parameter is $P = \lceil \log_2(1/\alpha) \rceil = 17$, so the ticket length is

$$\ell = m(P + 1) = 125 \times 18 = 2\,250 \text{ bits} \approx 280 \text{ bytes}.$$

At 10 Mbit/s this transmits in $t = 2\,250/(10 \times 10^6) \approx 0.22$ ms, seven orders of magnitude smaller than one RTT.

Worst-case ticket ($m = 60\,000$, $\alpha = 0.001\%$). If the sender naively inserted *all* public channels the ticket would be

$$\ell = 60\,000 \times 18 = 1\,080\,000 \text{ bits} \approx 130 \text{ kB},$$

which transmits in $t = 1\,080\,000/(10 \times 10^6) = 108$ ms, still below a single RTT.

We note that in LN the ticket travels piggy-backed on the initial `update_add_htlc`; subsequent frames (`commitment_signed`, `revoke_and_ack`) need not wait for the ticket to finish. SpeedyMurmurs defines latency as the duration of the *longest chain of causally dependent messages* in the protocol execution. Because the ticket is *outside* that critical chain, its transfer time – whether 0.5 ms or 48 ms – is absorbed during idle link time, leaving the overall handshake delay dominated by the 160 ms propagation budget.

4.4 Payment Splitting

SpeedyMurmurs can split a payment into partial payments, and send each partial payment along a different landmark–embedding tree, explicitly to *hide the total value from every individual relay* [17]. Achieving this goal requires the protocol to maintain several spanning trees and to send bookkeeping beacons whenever channels appear, disappear, or exhaust liquidity.

Our locally steered source routing realizes the same value privacy at virtually *no extra cost*. The sender simply issues as many tickets as there are partial payments – each keyed with its own per–hop secret – and launches the sub-payments concurrently. No additional control traffic is needed, and ticket construction remains fully local.

In addition, because the sender controls each ticket's edge set, it can impose *explicit path separations*: for two partial payments, the sender may generate disjoint channel sets whenever the topology allows, thus guaranteeing that no single relay observes every share. SpeedyMurmurs cannot enforce such disjointness even when multiple landmark-embedding tree is used.

5 Simulation Methodology and Results

To quantify the practical impact of T2R, we built a LN simulator in Python[2] and ran all experiments on a dedicated 32-core workstation with 32, GiB RAM, Ubuntu 20.04 LTS, and Python 3.11. We benchmarked T2R against SpeedyMurmurs in its single-tree configuration, the variant that the original paper reports as offering the highest overall performance – namely, the greatest success rate and the lowest delay – when payments are not split across trees.

Input graph. All experiments use the public LN gossip dump exported by Pickhardt on 12 April 2022[3]. This snapshot coincides with the historical peak of publicly advertised LN adoption – both in nodes and in channels[4] – and has served as a benchmark in prior LN-routing work. [18] The raw graph contains **14 135** nodes, **60 757** bidirectional channels, and **9 087** unidirectional channels.

A Lightning *channel* is backed by a single 2-of-2 multisignature output of capacity C. At any moment this capacity is split between the two peers. Accordingly, our simulator models each bidirectional channel as two directed *half-channels*: $(u \to v)$ carries the balance held by u, and $(v \to u)$ carries the complementary balance owned by v (C minus u's share). Unidirectional channels are represented by a single directed edge.

Connectivity pruning. SpeedyMurmurs can embed its landmark tree only in a strongly connected graph, thus we restrict the snapshot to its largest strongly connected component, discarding the 1 002 nodes that lie in isolated islands. This pruning is required *only* for SpeedyMurmurs; T2R operates correctly on any topology. We also disable every channel the snapshot marks as inactive, ensuring that the simulator uses only links considered live.

Capacity and initial balances. The snapshot records the on-chain capacity C_{uv} (in satoshis) of each channel (u, v) but, as usual, not its private directional balances. For our *baseline* experiments we assume an even split and assign

$$b(u \to v) = b(v \to u) = \tfrac{1}{2} C_{uv}.$$

To study the effect of systematic skew we repeat the experiments with a fixed global ratio $p \in \{0.6, 0.7, 0.8, 0.9, 1.0\}$, setting

$$b(u \to v) = p\, C_{uv}, \qquad b(v \to u) = (1 - p)\, C_{uv}.$$

These five settings correspond to the (60:40) through (100:0) points on the x-axes of our figures.

[2] `https://github.com/sajializ/ticket`
[3] `https://www.rene-pickhardt.de/listchannels20220412.json`
[4] `https://bitcoinvisuals.com/lightning`, chart "Total public channels / nodes (daily)"

Failure models. Lightning payments often fail because the sender's view is stale: a channel that *looks* usable in gossip may in fact be unusable when the HTLC arrives. We emulate two such hidden failure modes.

Silent channel disablement. A fraction $\rho \in \{10, 20, 30, 40, 50\}\%$ of channels is randomly marked inactive – neither direction can forward – yet their gossip entries remain unchanged. This captures maintenance outages or force-closures that have not yet propagated.

Random saturation. We pick $k \in \{10, 20, 30, 40, 50\}\%$ of channels uniformly at random and set their forward balance to zero in one direction, $b(u \to v) = 0$, while the reverse edge still forwards. This models liquidity depletion rather than shutdown.

5.1 Payment Flow Generation

Our goal is to measure how effectively each algorithm discovers a route *when one truly exists* under the current liquidity snapshot. Accordingly, every data point in a batch of 50 000 payments is generated as follows.

1. **Endpoint sampling.** Pick source s and destination d uniformly at random from the pruned graph ($s \neq d$).
2. **Amount selection.** Draw a payment value of A according to the scenario:
   - One setting of our simulation picks a uniformly random amount between intervals $\{10^0\text{–}10^2, 10^2\text{–}10^3, 10^3\text{–}10^4, 10^4\text{–}10^5, 10^5\text{–}10^6\}$ satoshis for each payment.
   - All other settings pick a random amount A between 10^2–10^3 satoshis. We chose this interval for the amount of the payment because both algorithms have low differences regarding success rate using this interval, which enables us to examine how other behaviors of the network affect them.
3. **Feasibility of the payment.** Before routing, we verify that some *optimal* path can carry (s, d, A) under the current liquidity *and* each channel's HTLC policy:
   - Every announcement advertises `htlc_minimum_msat` and `htlc_maximum_msat`. A hop may forward only if `htlc_min` $\leq A \leq$ `htlc_max`.
   - A directed edge $(u \to v)$ must also have $\mathit{balance}(u \to v) \geq A$ at this moment.
   - Every channel on the optimal path must be active (not flagged as disabled).

   We run a shortest-path search that enforces *all* of the above constraints. The source and destination for which no feasible path exists are discarded and resampled, ensuring every payment is routable in principle, so any failure reflects routing performance, not an *impossible* payment.
4. **Algorithm trials.** Two independent copies of the network are made – one for T2R and one for SpeedyMurmurs. Each algorithm attempts to route (s, d, A) *once* on its private copy:
   a. If the attempt succeeds, the simulator decreases the directional balances along the path and increases the reverse directions, simulating an immediate settle.
   b. If it fails, no balances change.

   Because the two runs start from identical liquidity and evolve independently, their success probabilities can be compared without any interference.
5. **Ticket generation.** For every feasible request we enumerate *all* $s \to d$ paths that satisfy the public HTLC policy at every hop ($\texttt{htlc_min}_{u\to v} \leq A \leq \texttt{htlc_max}_{u\to v}$), and the advertised channel *capacity* is at least the payment amount A. Every channel–direction that appears on at least one such path is inserted into the ticket. To ensure that any residual failure arises from the algorithm rather than ticket collisions, we instantiate the

filter with a near-zero false-positive rate. This setting makes the probability that an unusable edge is authorized negligible. We examine the false-positive rate on success ratio later in the simulation.

6. **Forwarding behavior.** Once a feasible request (s, d, A) is admitted, each algorithm is invoked *exactly once*; a failure at any hop marks the entire payment as unsuccessful – no sender-side retry or multipath splitting is performed. Inside the network the two schemes behave differently:
   - **T2R.** Upon receiving the payment an intermediary node chooses one of its outgoing channels *uniformly at random*. If that channel both (i) meets the HTLC amount/balance constraints and (ii) appears in the ticket, the intermediary node forwards the HTLC along it. Otherwise the node samples another neighbor at random – without replacement – and repeats the test until either a suitable channel is found or all neighbors have been exhausted. If no channel passes the test the payment fails immediately.
   - **SpeedyMurmurs.** Each intermediary follows the greedy rule (as explained in their paper) of the landmark tree: forward to the unique neighbor whose tree distance to the destination is strictly smaller than its own.

This procedure isolates the algorithms' ability to *find* a viable route, not their capacity to judge when no route exists; it also keeps the liquidity landscape dynamic, as successful payments progressively update balances – reflecting real Lightning traffic.

5.2 Evaluation Metrics

From each batch of $n = 50\,000$ payment attempts we extract two quantities to compare both algorithms.

1. **Success probability.** For a batch of n independent payment attempts let $\mathcal{S} \subseteq \{1, \ldots, n\}$ be the index set of payments that are completed successfully. The empirical success rate is

$$P_{\text{success}} = \frac{|\mathcal{S}|}{n} = \frac{\#\text{successful payments}}{n}.$$

2. **Average hop count.** Denote by h_i the hop length of the *realized path* for a successful payment $i \in \mathcal{S}$. If $m = |\mathcal{S}|$ is the number of successes, the sample mean hop count is:

$$\bar{h} = \frac{1}{m} \sum_{i \in \mathcal{S}} h_i.$$

5.3 Results

Figures 1, 2 demonstrates the behavior of the T2R and the single-tree implementation of SpeedyMurmurs across four axes: payment amount (a), different initial balances (b), channel silent disablement (c), and random saturation (d).

Different payment amounts

T2R sustains a $\geq 90\%$ first-attempt success rate up to 10^5 sat, whereas SpeedyMurmurs already drops to $\approx 77\%$ at 10^4 sat and collapses beyond 10^5 sat. Redundancy within the T2R's multipath findings allows intermediate nodes to avoid unexpectedly empty links without sender involvement. The number of hops of T2R is considerably lower than SpeedyMurmurs, meaning for each payment size interval, the T2R finds shorter paths.

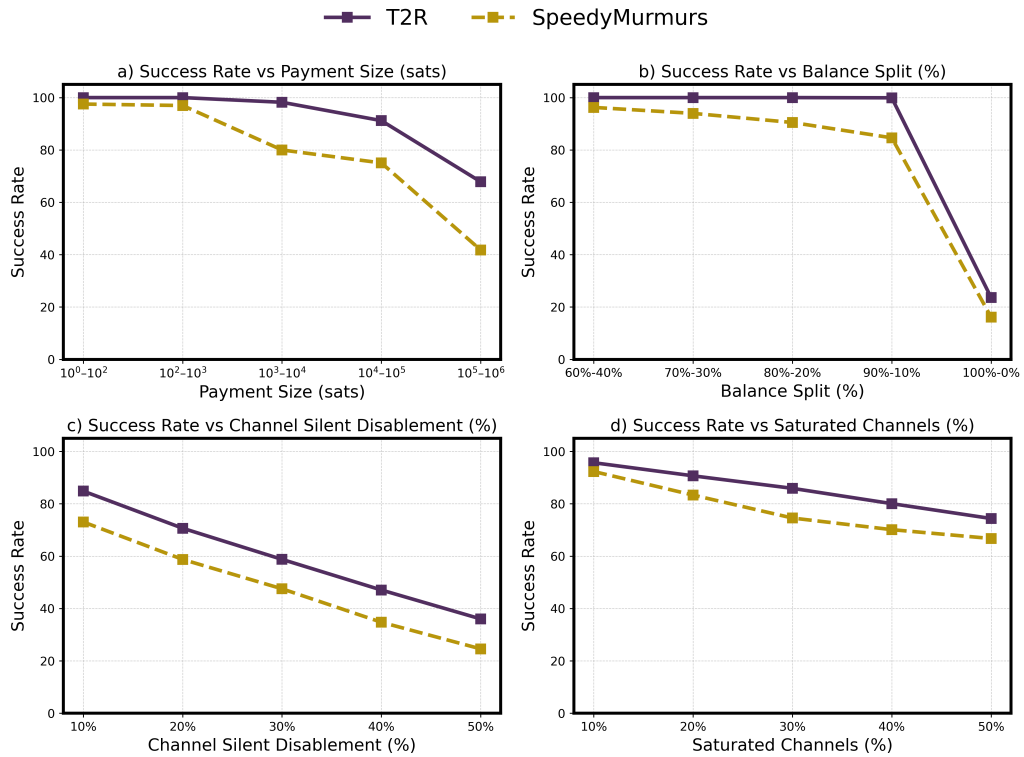


Figure 1 T2R vs. SpeedyMurmurs: success rate (higher is better) under four scenarios – payment size, balance split, silent channel shutdown, and random saturation.

Different initial balances

T2R shows a higher success rate and a lower average hop per payment for each point of split capacity. Both approaches experience a notable drop in success rate as the split becomes extreme. While SpeedyMurmurs completes only a small fraction of payments under the 100:0 skew, T2R retains a modest advantage, successfully routing about one-quarter of payments by exploiting alternative branches. The hop count remains flat until the extreme imbalance, but still lower mean hops count and for the T2R .

Silent Disablement

Across the different shutdown portions, T2R consistently outperforms the SpeedyMurmurs by *roughly twelve percentage points* in success probability. In every subplot, the T2R's curve has a meaningful gap with SpeedyMurmurs. Remarkably, it achieves this gain *while still trimming path length*: T2R's routes are 0.3–0.4 hops shorter compared to SpeedyMurmurs.

Random saturation

T2R achieves higher success rate and a lower mean hop count than SpeedyMurmurs at every level of saturation:

- With only 10 percent saturated edge ($k = 10\%$) T2R holds steady at $\approx 95\%$ success, whereas SpeedyMurmurs slips below 92%.

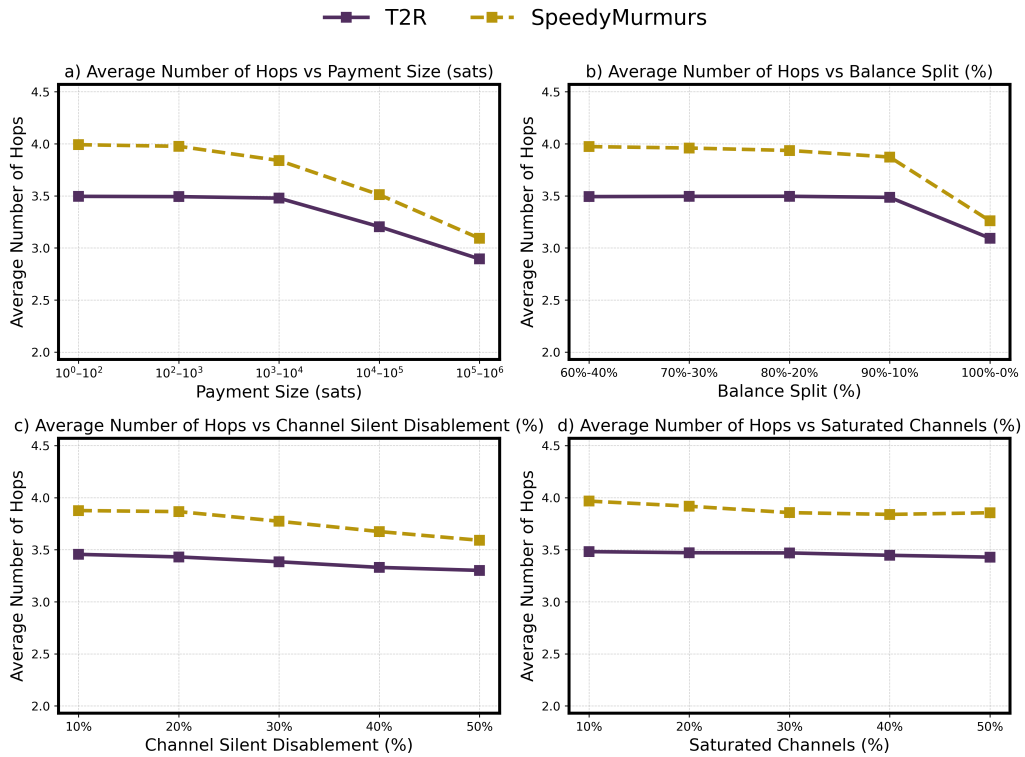


Figure 2 T2R vs. SpeedyMurmurs: average number of hops (lower is better) under four scenarios – payment size, balance split, silent channel shutdown, and random saturation.

- At the harshest setting ($k = 50\%$), both approaches experience a similar downward trend in success probability, with SpeedyMurmurs reaching around 66% and T2R retaining a lead at 74%, a difference of about eight percentage points.
- Across the entire sweep T2R's realized routes remain almost flat at $\bar{h} \simeq 3.4$ hops, consistently ~ 0.4 hops shorter than those of SpeedyMurmurs.

Overall, the two algorithms exhibit qualitatively similar behavior under stress, success rates degrade with increasing imbalance or saturation, but T2R consistently retains a modest advantage across all scenarios.

5.4 Ticket Size

T2R's overhead is dominated by the number of encrypted items inserted into its ticket. Hence, we profile its size under two workloads.

1. **$\mathbf{W_{all}}$: unrestricted pairs.** For the same 50 000 (s, d) pairs used in the main experiments, we build the candidate list for each node *without* capacity or HTLC limits and record $|B|$ – the element count in the resulting ticket. Also, for each node, we select all viable pairs. We ignored all these constraints to reach the *absolute worst-case* scenario regarding the number of elements of the ticket, enabling us to provide an upper bound for the size of the ticket.
2. **$\mathbf{W_{long}}$: long-distance pairs.** To demonstrate the worst case, we generate a separate batch of 50 000 pairs with the shortest path length of ≥ 4 hops; the rest is identical.

Table 1 reports the key statistics (mean, median, 90$^{\text{th}}$ and 99$^{\text{th}}$ percentiles, and maximum) of the number of added elements in the ticket in both scenarios.

Table 1 Number of elements per ticket.

	Mean	Median	P_{90}	P_{99}	Max
W_{all}	51.0	11	126	611	2738
W_{long}	88.4	34	212	847	2997

- **Typical payments (W_{all}).** The median ticket carries just 11 elements, while 90 % of payments stay below 126 elements. With a Bloom filter configured for a target error $f = 10^{-5}$ (0.001 %), the 90$^{\text{th}}$-percentile ticket occupies only 377 B, whereas an equally secure GCS fits in 284 B
- **Long-distance payments (W_{long}).** Deliberately selecting endpoints at least four hops apart roughly doubles the mean element count to 88 and the median to 34. Even so, 90 % of these worst-case payments stay under 212 elements; at the same $f = 10^{-5}$ this translates to 634 B with a Bloom filter and 477 B with a GCS. Even the rare cases of outlier are controllable: it vanishes once we cap the *branch factor* K – that is, allow each node to contribute only a handful of its best channels to the ticket.

5.5 Effect of Ticket False Positives on Success Rate

T2R's reliability degrades only when an *authorized* next hop is a false positive: the intermediary node assumes an unusable edge is valid and finds no real edge left to forward the payment. To isolate this effect, we run same 50 000 payments from our main simulation under a perfect network (no disabled or saturated channels, balance split 50:50), and with balance update after each successful payment. For each payment, the sender inserts m authorized channel directions and creates a ticket sized optimally for the chosen false positive rate f. Payment amounts are drawn uniformly at random in the 100–1000 sat interval used elsewhere in the paper; a payment is declared *failed* if some intermediary node has no authorized next hop. Also, we simulate both single ticket and the TwinTicket.

5.5.1 Size of TwinTicket

We sweep the primary ticket's false–positive target $f \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. The sender builds the TwinTicket by inserting *only* those node–pairs that collide in the primary ticket, meaning every pair that triggers a false positive in the primary ticket. Tables 2, 3 report the resulting element counts under both workloads from Section 5.4.

Table 2 Elements inserted into TwinTicket (τ_2), workload W_{all}.

False Positive f	Mean	Median	P_{90}	P_{99}	Max
10^{-2}	*40.6*	*29*	*92*	*176*	*467*
10^{-3}	*4.2*	*3*	*10*	*20*	*52*
10^{-4}	*0.4*	*0*	*1*	*3*	*10*
10^{-5}	*0.04*	*0*	*0*	*1*	*3*

Table 3 Elements inserted into TwinTicket (τ_2), workload W_{long}.

False Positive f	Mean	Median	P_{90}	P_{99}	Max
10^{-2}	*52.8*	*41*	*110*	*206*	*487*
10^{-3}	*5.3*	*4*	*12*	*22*	*52*
10^{-4}	*0.5*	*0*	*2*	*4*	*10*
10^{-5}	*0.05*	*0*	*0*	*1*	*3*

- **Typical payments (W_{all}).** For the TwinTicket setting we focus on three operating points that all achieve $\geq 99\,\%$ first-attempt success (Table 5 below, which is presented later in this section, illustrates the success rate based on false positives of the tickets) and compute their 90th–percentile sizes under two encodings. We take $m_{90} = 126$ authorized pairs in the primary ticket obtained in Section 5.4.

1. **$(f_1, f_2) = (10^{-2}, 10^{-3})$, $m_1 = 126$, $m_2 = 92$:**
   a. *Bloom Filter:* 150 B + 165 B = 315 B
   b. *GCS:* 126 B + 126 B = 252 B
2. **$(f_1, f_2) = (10^{-3}, 10^{-3})$, $m_1 = 126$, $m_2 = 10$:**
   a. *Bloom Filter:* 226 B + 18 B = 246 B
   b. *GCS:* 174 B + 14 B = 175 B
3. **$(f_1, f_2) = (10^{-4}, 10^{-2})$, $m_1 = 126$, $m_2 = 1$:**
   a. *Bloom Filter:* 301 B + 1 B = 302 B
   b. *GCS:* 237 B + 1 B = 238 B

These figures confirm that even the most redundant ticket ($f_1 = 10^{-2}$) plus its TwinTicket remains well below half a kilobyte in Bloom form and under one-third of a kilobyte with a GCS.

- **Long-distance payments (W_{long}).** Taking the 90th–percentile primary size $m_{90} = 212$ (Table 5) and the corresponding 90th–percentile collision counts from Table 3, we obtain:

1. **$(f_1, f_2) = (10^{-2}, 10^{-3})$, $m_1 = 212$, $m_2 = 110$:**
   a. *Bloom Filter:* 254 B + 197 B = 451 B
   b. *GCS:* 212 B + 152 B = 364 B
2. **$(10^{-3}, 10^{-3})$, $m_1 = 212$, $m_2 = 12$:**
   a. *Bloom Filter:* 380 B + 22 B = 402 B
   b. *GCS:* 292 B + 12 B = 304 B
3. **$(10^{-4}, 10^{-2})$, $m_1 = 212$, $m_2 = 2$:**
   a. *Bloom Filter:* 505 B + 3 B = 508 B
   b. *GCS:* 398 B + 2 B = 400 B

Even in the long–path scenario the most redundant configuration remains below 0.6 kB (Bloom filter) and 0.4 kB (GCS) for 90% of payments. Also, same as single ticket scenario, capping the *branch factor* K – that is, allow each node to contribute only a handful of its best channels to the ticket will reduce size of the ticket for outliers.

5.5.2 Single Ticket vs Success Rate

We vary the ticket target error probability: $f \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. Table 4 lists the success probability achieved at each tested false-positive target. We additionally calculate the size the tickets using the 90^{th}-percentile element counts reported in Section 5.4 using GCS. Once the filter is tightened to $f = 10^{-4} = 0.01\,\%$ the success rate already exceeds 90 %, and at $f = 10^{-5} = 0.001\,\%$ it climbs to $P_{\text{succ}} \approx 99.2\,\%$. Lowering f further adds only fractional improvement.

Table 4 Success rate versus filter error probability.

False-positive target f	Success probability [%]	Ticket size (Bytes)
10^{-2}	*15.0%*	*126B*
10^{-3}	*63.7%*	*173B*
10^{-4}	*93.5%*	*236B*
10^{-5}	*99.2%*	*283B*

5.5.3 TwinTicket vs Success Rate

To mitigate collisions we add TwinTicket. The sender constructs primary ticket with error f_1 as above, scans for collisions, and inserts those pairs into TwinTicket sized for an independent error f_2. Intermediate nodes accept a neighbor only if it is PRESENT in the primary ticket *and* ABSENT in the TwinTicket. We sweep the Cartesian grid $f_1, f_2 \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ and report the resulting success matrix in Table 5. We also calculate the aggregate size of both tickets using the 90^{th}-percentile element counts for the reported in Section 5.4 using GCS and the numbers reported in W_{all}.

Table 5 Dual-filter success probability $P_{\text{succ}}(f_1, f_2)$ [%] (sum of the sizes of both tickets in bytes).

	f_2			
f_1	10^{-2}	10^{-3}	10^{-4}	10^{-5}
10^{-2}	*96.0% (218B)*	*99.7% (252B)*	*100% (298B)*	*100% (333B)*
10^{-3}	*97.9% (183B)*	*99.4% (187B)*	*100% (192B)*	*100% (195B)*
10^{-4}	*99.0% (237B)*	*99.8% (238B)*	*100% (238B)*	*100% (238B)*
10^{-5}	*99.8% (283B)*	*100% (283B)*	*100% (283B)*	*100% (283B)*

Table 5 demonstrates how quickly the second ticket drives the failures to zero. With the primary ticket set to $f_1 = 10^{-2} = 1\,\%$ the success rate already reaches 96 %; tightening f_2 beyond that yields no visible benefit. A practical sweet spot is therefore $(f_1, f_2) = (10^{-2}, 10^{-3})$: it already delivers $P_{\text{succ}} \geq 99.7\%$. Pushing either filter to 10^{-4} or 10^{-5} achieves a rise toward 100% but at a sharply higher bit cost than the primary ticket. Choosing $f_1 = 10^{-2}$ and $f_2 = 10^{-3}$ yields $P_{\text{succ}} > 99\,\%$ while the *sum* of the primary and TwinTicket is only ≈ 180 B.

6 Related Work

Early approaches

The first proposals for payment-channel routing simply ran Dijkstra or A* on the public graph and probed candidate paths until one cleared liquidity. Because balances evolve with every transfer and are hidden from third parties, this "trial-and-error" strategy suffers a high failure rate and leaks balance information through repeated probes.

Lightning-specific schemes

Research quickly turned to designs that respect Lightning's hop-by-hop HTLC semantics and privacy constraints. Flare [16] lets each node maintain topology within k hops and query the receiver's neighborhood; it is fast but leaks per-hop liquidity to the sender, as observed by SpeedyMurmurs' authors [17]. Multipart approaches try to tame hidden liquidity by splitting a payment. Spider [19] treats each fragment as a packet and applies congestion control, boosting aggregate throughput but stretching completion times. Flash [20] sends "mice" along cached shortest paths and reserves expensive max-flow computation for "elephants." Pickhardt–Richter flows [13] formalize the optimization as sequential min-cost-flow problems, achieving the highest known success probability at the expense of heavy sender-side computation. To assist lightweight wallets, introduces *trampoline payments* [5]: the sender routes to a hub that owns the global graph, and the hub computes the remainder of the route. Our three-ticket variant (§4.2) extends trampolines with fine-grained, privacy-preserving steering.

Credit-network lineage

Off-chain credit networks such as Ripple and Stellar store bilateral IOUs on a ledger. Early Ripple relied on global consensus, exposing the entire path and amount on-chain. To avoid public state, SilentWhispers [12] used landmark nodes plus secure multiparty computation, guaranteeing privacy at high communication cost. SpeedyMurmurs [17] used metric-embedding trees as landmarks, optimizing latency and balancing load, but tree maintenance remains expensive and cannot encode sender blacklists. Our framework achieves the same privacy guarantee (no node learns non-incident edges) while discarding global trees entirely.

Approximate-membership filters

AMQs are a standard tool for compact set representation: Bloom filters [4], counting Bloom filters [7], Cuckoo filters [6], quotient filters [2], and Golomb–Rice Coded Sets, adopted in Bitcoin for light-client block filters [15]. We leverage AMQs as "tickets" that travel with a payment and reveal only incident edges to each hop.

Position of our work

Our T2R, locally steered source routing (i) sidesteps global trees, (ii) inherits incident-edge privacy from SpeedyMurmurs, and (iii) adds new control knobs – blacklists, receiver-issued tickets, trampoline segment steering – while keeping header overhead under one kilobyte and introducing no additional handshake round-trips.

7 Conclusion

This paper demonstrates that global landmarks or embeddings are *not* a prerequisite for privacy-preserving routing in payment-channel networks. By replacing them with a compact, cryptographically-keyed AMQ that *travels with the payment*, our **ticket-based framework** approach (i) removes continuous tree maintenance, (ii) keeps path privacy intact, and (iii) unlocks new control knobs – blacklisting misbehaving nodes, delegating route choices to receivers, and steering across trampoline hubs – without protocol round-trips.

In Lightning-sized topologies the TwinTicket configuration achieves $> 99\,\%$ first-try success at sub-kilobyte overhead, comfortably outperforming the SpeedyMurmurs baseline in both robustness and latency. Because ticket size scales linearly with the *number* (not length) of authorized channels, operators can trade header bytes for reliability on a per-payment basis.

References

1 ACINQ. Eclair. `https://github.com/ACINQ/eclair`, 2025. Version 0.9.0, accessed 27 May 2025.

2 Michael A. Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, Bradley C. Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P. Spillane, and Erez Zadok. Don't thrash: How to cache your hash on flash. *Proc. VLDB Endow.*, 5(11):1627–1637, 2012. `doi:10.14778/2350229.2350275`.

3 Blockstream. Core lightning. `https://github.com/ElementsProject/lightning`, 2025. Version 24.02, accessed 27 May 2025.

4 Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970. `doi:10.1145/362686.362692`.

5 Christian Decker et al. Trampoline onion routing (draft proposal). `https://github.com/lightningnetwork/lightning-rfc/pull/829`, 2020. (accessed May 2025).

6 Bin Fan, David G. Andersen, Michael Kaminsky, and Michael Mitzenmacher. Cuckoo filter: Practically better than bloom. In Aruna Seneviratne, Christophe Diot, Jim Kurose, Augustin Chaintreau, and Luigi Rizzo, editors, *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, CoNEXT 2014, Sydney, Australia, December 2-5, 2014*, pages 75–88. ACM, 2014. `doi:10.1145/2674005.2674994`.

7 Li Fan, Pei Cao, Jussara M. Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000. `doi:10.1109/90.851975`.

8 Florian Grötschla, Lioba Heimbach, Severin Richner, and Roger Wattenhofer. On the lifecycle of a lightning network payment channel. *CoRR*, abs/2409.15930, 2024. `doi:10.48550/arXiv.2409.15930`.

9 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 489–504. ACM, 2018. `doi:10.1145/3183713.3196909`.

10 Lightning Labs. BOLT #4: Onion routing protocol. URL: `https://github.com/lightning/bolts/blob/master/04-onion-routing.md`.

11 Lightning Labs. lnd: Lightning network daemon. `https://github.com/lightningnetwork/lnd`, 2025. Version 0.18.0-beta, accessed 27 May 2025.

12 Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Silentwhispers: Enforcing security and privacy in decentralized credit networks. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. URL: `https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/silentwhispers-enforcing-security-and-privacy-decentralized-credit-networks/`.

13 Rene Pickhardt and Stefan Richter. Optimally reliable & cheap payment flows on the lightning network. *CoRR*, abs/2107.05322, 2021. arXiv:2107.05322.

14 Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. Technical report, Self-published, 2016. Draft. URL: https://lightning.network/lightning-network-paper.pdf.

15 Jim Posen. Bip 158: Compact block filters for light clients. https://github.com/bitcoin/bips/blob/master/bip-0158.mediawiki, 2019.

16 Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An approach to routing in the lightning network. White paper, Bitfury Group Limited, July 2016. URL: https://bitfury.com/content/downloads/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf.

17 Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. URL: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_09-3_Roos_paper.pdf.

18 Sindura Saraswathi and Christian Kümmerle. An exposition of pathfinding strategies within lightning network clients. *CoRR*, abs/2410.13784, 2024. doi:10.48550/arXiv.2410.13784.

19 Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. High throughput cryptocurrency routing in payment channel networks. In Ranjita Bhagwan and George Porter, editors, *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 777–796. USENIX Association, 2020. URL: https://www.usenix.org/conference/nsdi20/presentation/sivaraman.

20 Peng Wang, Hong Xu, Xin Jin, and Tao Wang. Flash: efficient dynamic routing for offchain networks. In Aziz Mohaisen and Zhi-Li Zhang, editors, *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies, CoNEXT 2019, Orlando, FL, USA, December 09-12, 2019*, pages 370–381. ACM, 2019. doi:10.1145/3359989.3365411.

21 Philipp Zabka, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. Empirical evaluation of nodes and channels of the lightning network. *Pervasive Mob. Comput.*, 83:101584, 2022. doi:10.1016/J.PMCJ.2022.101584.

Trustless Bridges via Random Sampling Light Clients

Bhargav Nagaraja Bhatt
Web3 Foundation, Zug, Switzerland

Fatemeh Shirazi
Web3 Foundation, Zug, Switzerland

Alistair Stewart
Web3 Foundation, Zug, Switzerland

Abstract

The increasing number of blockchain projects introduced annually has led to a pressing need for secure and efficient interoperability solutions. Currently, the lack of such solutions forces end-users to rely on centralized intermediaries, contradicting the core principle of decentralization and trust minimization in blockchain technology. We propose a decentralized and efficient interoperability solution (aka *Bridge Protocol*) that operates without additional trust assumptions, relying solely on the Byzantine Fault Tolerance (BFT) properties of the two chains being connected. In particular, *relayers* (actors that exchange messages between networks) are permissionless and decentralized, hence eliminating any single point of failure. We introduce *Random Sampling*, a novel technique for on-chain light clients to efficiently follow the history of PoS blockchains by reducing the signature verifications required. Here, the randomness is drawn on-chain, for example, using Ethereum's RANDAO. We analyze the security of the bridge from a crypto- economic perspective and provide a framework to derive the security parameters. This includes handling subtle concurrency issues and randomness bias in strawman designs. While the protocol is applicable to various PoS chains, we demonstrate the protocol's practical feasibility by showcasing an instantiated bridge between Polkadot and Ethereum (currently deployed), and discuss some practical security challenges. Furthermore, we evaluate the efficiency of our on-chain light client verifier (implemented as an Ethereum smart contract) against SNARK-based approaches, demonstrating significantly lower gas costs for signature verification – even for validator sets up to 10^6.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases PoS Blockchains, Trustless Bridges, Light Clients, Decentralised Relayers, RANDAO Bias

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.31

Acknowledgements We thank Aidan Musnitzky, Alistair Singh, and Vincent Geddes from Snowfork and the Bridge team at Parity Technologies for fruitful discussions on practical challenges in implementation. We also thank Jeffrey Burdges, Robert Hambrock, and Syed Hosseini for useful feedback and for reviewing drafts of this work. Lastly, we thank Alfonso Cevallos and Handan Kilinç Alper for their involvement in the early stages of the project.

1 Introduction

Blockchains are designed as islands, it is easy to verify that a transaction is valid within the originating blockchain when one is following its history, but challenging otherwise. While interoperability may be easily resolved with trusted centralized intermediaries, it is not a desirable solution. Recent history has shown how risky this can be - centralised entities can be compromised or even act maliciously. According to a Chainalysis report [19], failures in centralized bridges account for over 60% of all crypto hacks, resulting in losses exceeding

© Bhargav Nagaraja Bhatt, Fatemeh Shirazi, and Alistair Stewart;
licensed under Creative Commons License CC-BY 4.0

$2B to date. In fact, four out of the top five incidents on the *rekt leaderboard* [47] are bridge-related hacks. The security of public blockchains hinges on decentralization, therefore, the mantra is to avoid relying on trusted intermediaries.

Bridges are by far the most attacked components in the blockchain space [28, 29], because secure and efficient bridges are difficult to design and resolving attacks requires cooperation between chains which is nearly impossible. Hence it is important that the bridge must not have weaker security than either of the source or target chain. We introduce the notion of *crypto-economically sound* bridges, where we carefully trade off soundness for efficiency. Assuming the honesty assumption used in the consensus of both chains, in expectation an attack on our bridge protocol would be as expensive as the lower market cap between the two chains. To this end, we present *Random Sampling*, an interactive *Commit-Challenge-Response* protocol for interoperability between a source and target PoS blockchain. Our protocol has an on-chain light client of the source chain deployed on the target chain (e.g., as a smart contract). Computation on blockchain networks is expensive, particularly in the context of verifying all signatures from Proof-of-Stake (PoS) validators on the source chain, and hence not feasible. Our light-client protocol (specifically designed for bridge applications) improves the efficiency of following the history of a chain by randomly subsampling the signatures to be checked. Intuitively, the protocol works as follows: a set of relayers claim that 2/3rd +1 of validators signed a message (e.g., a block's finality). On-chain light client draws randomness from the target chain that determines a small subset of signatures to be checked out of those claimed. If the checks pass, we accept that the message was signed by at least one honest validator. The main advantages of our approach are:

1. With permissionless relayers, our bridge eliminates single points of failure without needing additional trust assumption for bridge safety, unlike centralized alternatives.
2. Our approach prioritises and achieves efficient on-chain verification, which is by far the most significant costs in operating trust-minimised bridges. We also provide a framework leveraging crypto-economic arguments to optimise it further.
3. Our protocol is relatively simple to implement and does not require custom SNARKs or Threshold Signature Schemes. Hence, we steer clear of implementation-level vulnerabilities of advanced cryptographic primitives, which are hard to get right [57, 20].
4. The main security parameter (i.e., the number of signatures checked by light client) only grows logarithmically with the validator set size. This property enables random sampling to scale well against large validators sets.

Our crypto-economic security analysis of the *random sampling* protocol considers attacks such as griefing and safety violations. This includes detecting subtle concurrency issues in strawman designs and fixes which rely on dynamically increasing the key security parameter (i.e., number of signatures checked by verifier) only in case of an attack. We then provide a framework to derive security parameters. Further, we apply the Fiat-Shamir heuristic to transform our interactive protocol into a non-interactive digital signature, and explore its impact on efficiency and interplay with crypto-economic arguments.

Random Sampling is applicable to diverse PoS networks, including Ethereum, Algorand, Sui, Polkadot, and Stellar, which satisfy most of the requirements for source-chains stated in Sec 2.1. Our protocol requires an additional *justification layer* where validators vote on finalised blocks, facilitating efficient verification by light clients. Consensus mechanisms are typically not designed for efficient light clients. Fortunately, we show that a light-weight justification layer can be modularly added on top of any deterministic finality gadget with low compute overheads and without the need to modify the underlying consensus mechanism. Secondly, crypto-economic arguments for allowing a larger soundness error can't be applied for

source chains without slashing provisions (e.g., Cardano, XRP, Near, ICP). Hence, although random sampling is applicable, its full efficiency can not be harnessed for chains without slashing provisions. On the destination chain, we only require on-chain randomness and support for smart contracts.

Finally, we instantiate our solution for a Polkadot-Ethereum bridge BEEFY, which has been deployed on the main-nets [53]. We discuss practical challenges like implementing slashing and handling bias in RANDAO beacon. We perform a thorough analysis to quantify RANDAO biasability, which offers insights valuable to the security of any protocol relying on RANDAO.

A prominent interoperability solution comparable to our work is Ethereum's Altair Light Client protocol [3] using sync-committees. In Altair, the source chain subsamples the validators and is fixed for the whole epoch, while our solution lets the verifier on the target chain sample randomly. This key insight lets us improve over Altair along three dimensions:

- for the same soundness gurantees, our protocol is more efficient by a factor of 20 for large validator sets ($\approx 10^6$)
- handles adaptive corruption of validators. An elected *sync-committee* in Altair remains stagnant for a whole epoch, exposing the committee against targeted attacks. In contrast, validators are sampled from randomness on the destination chain in our approach, leaving only a short window to adaptively corrupt the sampled validators.
- our protocol exposes a security parameter to the light client that allows tuning the security and efficiency based on the application requirements.

2 Preliminaries and System Model

Proof of Stake consensus mechanisms for blockchains require the nodes to *stake* the native cryptocurrency for a fixed period. In return, these nodes earn the opportunity to become *validators*, receiving rewards for producing blocks and participating in consensus. The security of the network is derived by the fact that the stakes of misbehaving nodes can be slashed (forfeited).

Light-clients are blockchain nodes that run in resource constrained environments like browsers or mobile devices to follow a decentralized consensus protocol. They do not maintain the entire blockchain history but instead validate the most critical pieces of information, such as block headers, to verify the integrity of the blockchain state. In particular, light clients have direct applications in building trustless and decentralized bridges between blockchains. Our definition of a bridge is in the broadest sense and application-agnostic. In this work, we define **bridges** as protocols that let two chains communicate and follow the finality (thru block headers) of each other. Applications like asset-swaps can be built on top of this basic functionality.

2.1 System Model

We formalize our setting, where the objective of an on-chain light client $\mathcal{LC}$ on *destination* chain $\mathbb{D}$ is to follow the finality of a *source* PoS blockchain $\mathbb{S}$. For brevity, we model the light client as a smart contract, however, our results hold for other computation models for updating state on $\mathbb{D}$. Our setting comprises of three actors: Validators on the source chain $\mathbb{S}$, trustless relayer $\mathbb{R}$, and a light-client $\mathcal{LC}$ deployed on $\mathbb{D}$. We make some standard assumptions on the **PoS model** of $\mathbb{S}$, satisfied by most PoS networks like Cosmos, Ethereum, Polkadot:

1. **Consensus:** The blockchain $\mathbb{S}$ runs a consensus mechanism with deterministic finality, e.g., *Byzantine Fault Tolerant* algorithms like CasperFFG, GRANDPA, Tendermint, PBFT, etc. [15, 54, 11, 17].
2. **Payloads:** Each block B in $\mathbb{S}$ contains a payload $\mathcal{P}_B$ capturing the state of $\mathbb{S}$ after executing block B. Typically, the payload is the hash of a crypto accumulator (e.g., Merkle Tree or Merkle Mountain Range Root [56]) where its leaves are the state of $\mathbb{S}$.
3. **Justifications:** Oblivious of the underlying consensus mechanism, once a block B is finalised in $\mathbb{S}$, the payload $\mathcal{P}_B$ is signed by all honest validators in the associated validator set $\mathcal{V}_B$ of size n. We assume at most f validators are malicious, such that $3f + 1 < n$. The validator use an unforgeable signature scheme σ to attest the payloads. Let σ_i be the signature of V_i on $\mathcal{P}_B$, which can be verified against its public-key pk_i. We often identify the validator V_i by its public-key. A block B is considered *justified* (or finalised when context is clear) if its payload $\mathcal{P}_B$ has been signed by at least $^{2n}/_3 + 1$ validators in $\mathcal{V}_B$. The set of such valid signatures are called justifications, denoted $\mathcal{J}_B = \{\sigma_i | i \in [1, .., n]\}$. We say a justification is valid iff $|\mathcal{J}_B| \geq {}^{2n}/_3 + 1$ and σ_i's are valid signatures from validators in $\mathcal{V}_B$. Assuming the underlying consensus mechanism is safe, note that for a particular block-height h, only a unique block B can have valid justifications.
4. **Stakes:** Validator V_i has a stake s_i locked on $\mathbb{S}$. Any malicious behavior by V_i results in partial or full slash of validator's stake. We assume the delay in detecting an offense and enforcing the slash and is bounded by Δ, the *slashing delay*.
5. **Epochs:** The blockchain S is divided into consecutive set of blocks called *epochs*, denoted $\mathcal{E} = \{\mathcal{E}_0, \cdots, \mathcal{E}_i, \cdots\}$. For any i, the validator set $\mathcal{V}_{\mathcal{E}_i}$ remains unchanged for all blocks within an epoch $\mathcal{E}_i$.
6. **Epoch Transitions:** A commitment (e.g. Merkle Root) to the next validator set $\mathcal{V}_{\mathcal{E}_{i+1}}$ is included in the payload of $\mathcal{E}_i$'s last block. PoS networks provide this functionality to facilitate light-clients to sync-up efficiently to the head of the chain by tracking only blocks with validator set changes.

Note: *Justifications* are succinct proofs of finality designed to simplify the light-client's process for following the chain head. We require honest validators to justify a payload only if the payload has already been finalized by the underlying finality gadget. Unlike consensus mechanism where an honest validator may sign a non-finalized block, an honest validator never signs justification for a block that has not already been finalized. Hence, the justification of a block guarantees stronger properties than being finalized by the underlying finality gadget like Grandpa or Casper-FFG. We show it is straightforward to construct a justification layer on top of any finality gadget in Section 5.1.

Relayers (denoted $\mathbb{R}$) are entities that collect justifications $\mathcal{J}_B$ from $\mathbb{S}$ and interact with the light-client $\mathcal{LC}$ to convince it of the newly finalised blocks. In the process, they may collect fees for their efforts. Relayers are completely permissionless, i.e, they do not deposit any stakes either on $\mathbb{S}$ or $\mathbb{D}$ nor there is any registration. Relayers can communicate with $\mathbb{D}$ by calling transactions of the smart contract $\mathcal{LC}$. The only requirement on $\mathbb{D}$ is:

1. **Randomness:** $\mathbb{D}$ provides a source of randomness $\mathcal{R}$ accessible to $\mathcal{LC}$. Ideally, $\mathcal{R}$ is unbiasable by the relayers, and for that matter, even by $\mathbb{D}$'s validators. We model the source of randomness as a random function $\mathcal{R} : Blocks_{\mathbb{D}} \to \mathbb{N}$, where $\mathcal{R}(B^{\mathbb{D}}) \in \mathbb{N}$ is the randomness generated at block $B^{\mathbb{D}}$.

2.2 Attacker Model

We model the attackers as rational agents, ie., an attack is launched only if the expected outcome of an attack is positive. Our attacker model allows collusion between the relayers and validators on $\mathbb{S}$, as well as validators on $\mathbb{D}$. We tolerate adversarial faction up to the limit

determined by the underlying consensus mechanism on both $\mathbb{S}$ and $\mathbb{D}$, without adding new trust assumption for safety of the bridge. In particular, relayers are completely permissionless and trustless. The relayer have no stake attached on either $\mathbb{S}$ nor $\mathbb{D}$. The relayer also can arbitrarily initiate and break an interactive session with the light client. This also means relayer can spawn multiple session concurrently. However, we assume the existence of at least one honest relayers for liveness of the bridge.

2.3 Problem Statement

We tackle the problem of building trustless bridges using light clients [63]. Typically, a light client synchronizing block headers of the source chain is implemented as a smart contract on the destination chain. This ensures the destination chain can verify information about the state of $\mathbb{S}$ without relying on external parties. Moreover, it allows anyone to prove the existence of transactions on $\mathbb{S}$ for smart contracts on $\mathbb{D}$ (using Merkle proofs), paving the way for generic applications. We formally lay out the desired properties of a light client protocol between verifier $\mathcal{LC}$ and prover $\mathbb{R}$, with an objective to update $\mathcal{LC}$'s view of the latest finalised block on $\mathbb{S}$. The $\mathbb{R}$ (prover) wants to convince the $\mathcal{LC}$ (verifier) that at least one honest validator on $\mathbb{S}$ signed the payload $\mathcal{P}_B$ of a recently finalised block B. We introduce the notion of *crypto-economic soundness* for light-client protocols which lets us trade-off negligible soundness guarantees for efficiency.

▶ **Definition 1 (ϵ-Soundness).** *Let Π be the protocol between $\mathbb{R}$ and $\mathcal{LC}$. Assume no honest validator in $\mathbb{S}$ signed $\mathcal{P}_B$. If the $\mathbb{R}$ (prover) can convince $\mathcal{LC}$ (verifier) of $\mathcal{P}_B$'s authenticity with probability at most ϵ, then Π is defined to be ϵ-sound. We term ϵ as the soundness error of Π.*

Our attacker model does not make any honesty assumptions on relayers. Any safety violation in our approach is traced back to the validators signing malicious payloads. We require the light client protocol to be *accountable*, i.e, malicious validators on $\mathbb{S}$ can be identified and slashed. Our goal is to design a light client protocol that is *crypto-economically sound*, as defined below:

▶ **Definition 2 (Crypto-economic Soundness).** *Let Π be ϵ-Sound. If ϵ is such that the expected outcome of an adversary attacking Π is negative, then Π is Crypto-economically Sound.*

3 Interactive Random Sampling Protocol

We describe an interactive light-client protocol Π_{Int} between a relayer $\mathbb{R}$ listening to finalised blocks on $\mathbb{S}$ and light-client $\mathcal{LC}$ deployed on $\mathbb{D}$. $\mathbb{R}$ wishes to convince $\mathcal{LC}$ of a new finalised block B on $\mathbb{S}$ which succeeds the latest finalised block known to $\mathcal{LC}$. Here, by *convince*, we mean that the light-client is ensured that at least one honest validator signed B.

Once a block B is finalised on $\mathbb{S}$, the relayers collects the justifications $\mathcal{J}_B$ and initiates Π_{Int}. In practice, the relayer can obtain the justifications by running a full-node of $\mathbb{S}$ and listening to it's consensus gossip-network. Naively, $\mathcal{LC}$ can check the finality of B by verifying $f+1$ signatures (where f denotes the malicious nodes on $\mathbb{S}$) in $\mathcal{J}_B$, ensuring at least one honest validator signed B. Combining the above observation with byzantine assumptions ($3f+1 < n$) on $\mathbb{S}$, the light-client is required to verify $n/3+1$ signatures in $\mathcal{J}_B$. Π_{Int} is a *Commit-Challenge-Response* [25] which reduces the number of signature verifications (sub-linear w.r.t validator-set size) performed on the verifier's side. In particular, Π_{Int} can be

viewed as an instantiation of Interactive Oracle Proofs (IOP) [8] for the language of *digital signatures* with extensions to equip it with accountability. We synonymously use the term *Prover* for the relayer $\mathbb{R}$ and *Verifier* for the light-client $\mathcal{LC}$.

3.1 Description

The prover $\mathbb{R}$ initiates the protocol claiming to have valid justification $\mathcal{J}_B = \{\sigma_i | V_i \in \mathcal{V}_B\}$ for the finality of block B, where σ_i's are signatures of V_i on payload $\mathcal{P}_B$. Instead of sharing the whole justification (super-majority of signatures), the relayer shares its *Claims*, a bitfield of length $|V|$ (validator-set size) which represents the validator signatures claimed to be possessed by $\mathbb{R}$. For accountability reasons, the tuple in the *commit* phase also includes a validator signature σ_j along with the claims, which we term as the *backing validator* of the current claim. Note that *Backing Validators* is only an expository term we use to describe the protocol and prove its security. As such, backing validators are not special actors in the protocol. We assume the verifier knows the Merkle Root $\mathcal{C}$ of the set of validators (identified by their public keys) for the epoch in which B is finalised. As outlined in section 2.1, the epoch transition mechanism on source chain comprises of a *hand-over* process, where the Merkle Root of the validator set responsible for next epoch is included in the state by the last block of the current epoch. Such hand-overs are mandatorily enforced as part of the protocol. In unforeseen circumstances when such *handovers* are unsuccessful, fall-backs mechanism can designed using on-chain governance. To bootstrap the protocol, either the genesis block or a trusted snapshot can be used. The verifier has access to public randomness $\mathcal{R} : Blocks_{\mathbb{D}} \to \mathbb{N}$ which it can query at a particular block on $\mathbb{D}$. In the *challenge* phase, it uses $\mathcal{R}$ to query a random subset of the signatures. The prover responds by sending the signatures and openings of the randomly sampled signatures in the *response* phase. The verifier maintains two variables in its state: $latestSyncedBlockHeight$ and $latestSyncedPayload$, denoting the block-height and payload of the latest know block finalised on $\mathbb{S}$. If the signatures and opening submitted in the *response* phase verifies, the payload ($\mathcal{P}_B$) is accepted the state is accordingly updated.

We assume the signature scheme σ is unforgeable and the commitment scheme $\mathcal{C}$ is binding. We ignore the negligible probability that the prover can find a signature that verifies against the public key of an honest validator that did not sign the message or that they can find an opening of $\mathcal{C}$ at position i to a value other than pk_i that verifies. To ensure randomness is unpredictable to the prover at *commit* phase, verifier uses the randomness $\mathcal{R}(d)$ only revealed after the commit phase as concluded. Equipped with the above assumptions, we now present the soundness and completeness results for Π_{Int}.

▶ **Theorem 3 (ϵ-Soundness).** *Consider prover $\mathbb{R}$ initiates Π_{Int} for block B. If no honest validator in $\mathbb{S}$ signed $\mathcal{P}_B$, then the verifier $\mathcal{SC}$ accepts $\mathcal{P}$ with probability at most 2^{-m}, where m is the security parameter, i.e., the number of signatures randomly sampled by the verifier.*

Proof. Let's assume no honest validator signed $\mathcal{P}_B$. The prover must provide a bitfield *Claims* with at least $n - f$ 1s. Consider k for some $1 \leq k \leq m$. The public key pk_{i_k} belongs to a dishonest validator only if i_k is one of at most f possible values out of $n - f$ 1s at the least. Since $n > 3f$ and $\mathcal{R}$ is unpredictable and uniformly random, with probability at least $1 - f/(n - f) > 1 - f/2f = 1/2$, pk_{i_k} belongs to an honest validator. Since the i_ks are each chosen independently, the probability that no pk_{i_k} for any k belongs to an honest validator is at most 2^{-m}.

It remains to show that if some pk_{i_k} belongs to an honest validator, then the prover cannot convince the verifier. The prover cannot provide an opening of $\mathcal{C}$ at position i_k to a value other than pk_{i_k} because the commitment scheme is binding. The prover cannot provide

Protocol 1 Π_{Int} : Interactive Random Sampling.

1. **Commit:** Prover $\mathbb{R}$ sends a tuple $(\mathcal{P}_B, \mathit{Claims}, \sigma_j, op_j)$ to the verifier, where:
   - $\mathcal{P}_B$: Payload for the block B.
   - $\mathit{Claims} = [b_1, \cdots, b_n]$: a bitvector of length n. An honest prover sets $b_i = 1$ iff they possess the signature σ_i on $\mathcal{P}_B$ that verifies against pk_i.
   - σ_j: signature of the backing validator v_j.
   - op_j: opening of $\mathcal{C}$ to pk_j (i.e., the Merkle co-path).
2. **Challenge:** Verifier $\mathcal{LC}$ checks if *Claims* has at least $n - f$ indices set to 1s, else it terminates. If the signature σ_j is valid and op_j opens to $\mathcal{C}$ for pk_j, then the verifier samples m indices $i_1, \ldots i_m$ where each i_k is chosen uniformly at random (using $\mathcal{R}$) from the positions of bits in *Claims* set to 1s.
3. **Response:** Prover sends signatures σ_{i_k}, the public keys pk_{i_k} and openings op_{i_k} of $\mathcal{C}$ to pk_{i_k} for $k \in [1, m]$ to the verifier.
4. **Verify:** Verifier performs the following checks and terminates if any fail:
   - the openings $op_{i_1}, \ldots, op_{i_m}$ against the corresponding public keys $pk_{i_1}, \ldots, pk_{i_m}$ and $\mathcal{C}$ at the randomly chosen indices $\{i_1 \ldots i_k\}$.
   - signatures $\sigma_{i_1}, \ldots, \sigma_{i_m}$ against the public keys $pk_{i_1}, \ldots, pk_{i_m}$ and $\mathcal{P}_B$.
   - If $height(B) > latestSyncedBlockHeight$, $\mathcal{LC}$ makes the following state updates:
     - $latestSyncedBlockHeight := height(B)$
     - $latestSyncedPayload := \mathcal{P}_B$

     Else, B is stale and state remains unchanged.

Figure 1 Π_{Int}: Verifier randomly samples a subset of validator signatures to be verified.

a signature σ_{i_k} that verifies against pk_{i_k} because the signature scheme is unforgeable and honest validators did not sign $\mathcal{P}_B$. Hence the prover can not convince the verifier in this case. We note that if $m > f$, the protocol is deterministically sound. If the verifier needs to be absolutely sure, the verifier can verify $m > f$ signatures, since at least one of those signatures will be of an honest validator. ◀

▶ **Theorem 4 (Completeness).** Π_{Int} *is* complete. *If relayer (prover) posses valid justifications* $\mathcal{J}_B$ *of a valid block* B*, then the light clients (verifier) updates it's state of the latest synced block to* B*.*

▶ **Example 5.** Suppose $\mathbb{S}$ has 100 validators of which at most 33 are byzantine. A block gets finalised with at least 67 signatures in its justifications. The relayer collects these justifications and starts Π_{Int} to convince the verifier that the block has been finalised. If the verifier samples $m = 34$ signatures in the *challenge* phase, it can be sure that at least one of these signatures is from an honest validator. If the verifier only samples $m = 10$ signatures in challenge phase and the *verify*-phase goes thru, then Theorem 3 guarantees that the probability of the light client accepting a malicious (invalid justifications) block is at most $1/2^{10}$.

3.2 Optimisation: Sampling Without Replacement

Observe that the verifier samples m indices in the *Challenge* phase in Protocol 1, where indices are chosen uniformly at random from the positions of bits in *Claims* set to 1s. This sampling is done with replacement, i.e., the same index can be sampled multiple times. However, the soundness error can be improved by sampling without replacement, i.e., all the

sampled indices are unique. Assuming the validator set size is n of which at most $n/3$ are malicious, and m' signatures chosen uniquely at random by the verifier, the soundness error is bounded by:

$$\frac{(n/3)!}{(n/3-m')!} * \frac{(n-m')!}{n!} \tag{1}$$

The soundness error bounds for sampling *with* and *without* replacement converge asymptotically with respect to n, but for practical validator set sizes $\leq 10^6$, the difference is significant. More importantly, one can get away sampling fewer signatures without replacement for the same soundness error. For example, sampling 41 signatures without replacement provides the same soundness error as sampling 128 signatures with replacement to achieve a soundness error of 2^{-128}.

3.3 Crypto-Economic Security

Π_{Int} guarantees probabilistic ϵ-Soundness with the soundness error $1/2^m$, solely dependent on m, the number of signatures sampled in the *challenge* phase. This leads to a natural question: how to set the security parameter m? We provide a crypto-economic framework for deriving the security parameter, striking a balance between efficiency and security.

Accountability and Slashing Exposure. We specifically require $\mathbb{R}$ to include at least one signature σ_j (i.e., backing validator V_j's signature) from their *Claims* in the *commit*-phase. If the payload is malicious, then the *backing validator* who signed the malicious payload can be identified and slashed on $\mathbb{S}$. In absence of backing validator signatures in the *commit* phase, the provers can initiate arbitrarily many instances of Π_{Int} sequentially or concurrently. The relayer would then have the choice to continue only in instances where the randomness drawn by the verifier in *challenge* phase is favorable. The validators have no economic disincentive for signing malicious payloads. Consequently, the relayers can increase the number of attempts (more turns at rolling the dice) in submitting malicious payloads. Therefore, any attempt by validators at signing malicious blocks (i.e., a block not finalised on $\mathbb{S}$) needs to be penalised/slashed.

We assume adversaries are rational, implying that an attack is initiated only if the expected economic value of the attack is positive. Our model considers the economic value of a successful attack to be the market-cap of $\mathbb{S}$, denoted M. If the attack is unsuccessful, we consider the lowest stake s of the validators on $\mathbb{S}$ to be the economic loss for the attackers. We assume the stake of all validators on $\mathbb{S}$ are identical, however, our framework can be extended to weighted PoS systems. The expectation of the economic value of an attack is computed as: $E(X) = (\epsilon \cdot M) - s$, where X is random variable for the economic value of the attack and ϵ is soundness error of Π_{Int}. Setting $E(X) < 0$, we derive $m \geq log_2(M/s)$.

Consequently, assuming the ratio of staked token to total issuance is α and the stakes are equally distributed among the validators, the slashable stake per validator is $s = \alpha \cdot M/|V|$, where $|V|$ is the validator set size. Hence, the derived security parameter $m \geq log_2(|V|/\alpha)$ is logarithmically dependent on $|V|$.

3.4 Discussions on Safety and Liveness

If a relayer initiates the *Commit* phase for a canonical block but does not respond to the *Challenges* by the verifier (either due to genuine or malicious reasons), there is no necessity to slash the *backing validator* revealed in the commit phase. Slashing any validator that signs a non-canonical block (i.e. with no valid justifications) suffices for the ϵ-soundness of Π_{Int}. A

soundness attack (i.e. relaying a malicious block) is public already at the *Commit* phase and eventually the *backing validator* who signed a malicious payload gets slashed, guaranteeing ϵ-soundness without any honesty assumptions on the relayers.

We assume there is one honest relayer who relays at least one block per epoch for liveness of the bridge, i.e., the verifier is in sync with the latest finalised blocks on the source chain. Presence of at least one honest relayer per epoch ensures that the verifier can track the validator set changes. A subtle corner case impacting soundness arises when there is no honest relayer for a time-period greater than the *unbonding period*, i.e, time allowed for an active validator to unbond their stake and exit the network (usually 20-30 days). If no blocks are relayed and updated by the light client over such a long period, then there is a possibility that validators on source chain unbond their stake and hence there is no stake to slash for Π_{Int} attempts after the unbonding period. However, we feel this is unlikely for bridge applications with a clear monetary incentive to stay live. Moreover, the lengthy unbonding period provides ample time to detect and rectify the situation. The problem of bootstrapping efficiently from genesis [1] is more of a concern for light clients that are frequently offline (e.g. light clients embedded in user wallets) unlike bridge applications.

4 Dynamic Random Sampling

Π_{Int} is prone to a subtle concurrency attack that increases the soundness error (ϵ) exploiting the following observations:

- Relayers are trustless and permissionless (in-line with our design principle). Hence, an adversary can spawn arbitrarily many relayers that initiate Π_{Int}.
- It takes non-negligible time Δ for slashing a validator (on $\mathbb{S}$) after detecting its signature on a malicious payload. Moreover, it takes the duration of a full epoch ($|\mathcal{E}|$) for the light client to discover the change in validator set (i.e., malicious validator has been ejected).

Concretely, lets assume an adversary $\mathcal{A}$ controls f validators on $\mathbb{S}$, implying $\mathcal{A}$ can obtain $n/3$ validators signatures on any malicious payload. The adversary proceeds by initiating c concurrent instances $\mathcal{I}_1, \ldots, \mathcal{I}_c$ of Π_{Int} for a particular block B (with malicious payload signed by the f malicious validators) with the same backing signature σ but different claims bitvectors $\{b_1, \ldots, b_c\}$ in the *Commit* phase. Since there is a delay $\Delta + |\mathcal{E}|$ before the light-client is aware of the slashing and validator set change, the adversarial relayer can reuse the same backing validators signature without increasing slashing exposure of his instances. The adversary then proceed with the protocol only in those instances where the random sampling of challenge indices in the *Challenge* phase is favorable, i.e., the challenge indices correspond to malicious validators. Concretely, the success probability of the attack can now be quantified as $1 - (1 - \epsilon)^c$. However, the slashing exposure is at most s_j, as only σ_j, the signature of the malicious backing validator is exposed. Analogous to the expected obtained in Section 3.3, we can compute expected outcome under the above attack scenarios with c concurrent instance of Π_{Int} as:

$$E_c(X) = (1 - (1 - \epsilon)^c)M - s \tag{2}$$

The expected attack value increases with c, the number of concurrent relay instances spawned. For every ϵ, there exists a c such that $E_c(X) > 0$. This is a clear attack on the crypto-economic soundness of Π_{Int}.

4.1 Counter-measure: Dynamically Increasing Signature Checks

To counter the concurrency issue described above, we propose Π_{Dyn}, an extension of Π_{Int}. In Π_{Dyn}, the security parameter (signatures sampled) dynamically increases based on the number of relay instance backed by the same *backing* validator. The light client (verifier) now keeps a counter $u_{e,v}$ for each *backing* validator v in epoch e that increments by 1 whenever there is an initial claim made by a relayer. For any further relay instances by a relayer (or a set of relayer) using the same backing validator v, the number of signatures sampled during challenge phase is increased by $1 + 2 * \lceil log_2(u_{e,v}) \rceil$. If c instances of Π_{Dyn} are initiated concurrently using the same *backing* validator, the probability of attack succeeding is bounded by summation over all the c instances using union bounds:

$$\sum_{i=1}^{c} \frac{1}{2^{m+1+2*\lceil log_2 i \rceil}} \leq \frac{1}{2^m} \tag{3}$$

This dynamic increase in the number of signature checks ensures that the probability of successful attack is bounded irrespective of the number of concurrent Π_{Dyn} initiated, and the advantage gained by using the same *backing* validator multiple times is neutralized.

The setup for Π_{Dyn} is very similar to Π_{Int} except that the verifier ($\mathcal{LC}$) maintains an additional mapping $u : (\texttt{Epoch}, PK_{\mathcal{V}_B}) \to \mathbb{N}$, where $u(e, v)$ captures the number of time a validator has been used for the backing signature in the commit-phase. Here, the *backing* validator is identified by its public-key. We do not keep track of relayers as they are permissionless and any attempt to keep track is not sybil resistant. Intuitively, the dynamic random sampling described in Protocol 2 ensures the adversary does not increase probability of successful attack without increasing its slashing exposure. Importantly, note that the security parameter remains unchanged if no concurrency attack is launched.

▶ **Theorem 6 (Dynamic Random Sampling Soundness).** *Let $\Pi_{dyn,\epsilon}$ denote an instantiation of Π_{Dyn} (Protocol 2) where security parameter m is set s.t $\epsilon = 1/2^m < s/\mathcal{M}$. For any PPT adversary $\mathcal{A}$ controlling at most $n/3$ validator on $\mathbb{S}$, the expected incentive for an attacker $\mathcal{A}$ is negative.*

Proof. We first compute the probability of successful attack by defining the following events:

- $W_{\mathcal{A}}$: adversary convinces the verifier an un-finalised block B without *justifications*, i.e., the adversary successfully attacks Π_{Dyn}.
- $W_{\mathcal{A},v,k}$: the adversary succeeds and $u(e, v) = k$ in the verifiers state.
- $T_{\mathcal{A},v,k}$: the adversary attempts using v as the backing validator in *commit* phase for the k^{th} attempt, i.e., $u(e, v) = k$ after the attempt.

The success probability for an adversary is computed as follows:

$$\mathbb{P}(W_{\mathcal{A}}) = \sum_{v \in V} \sum_{k=1}^{\infty} \mathbb{P}(W_{\mathcal{A},v,k}) \tag{4}$$

$$= \sum_{v \in V} \sum_{k=1}^{\infty} \mathbb{P}(W_{\mathcal{A},v,k} | T_{\mathcal{A},v,k}) \cdot \mathbb{P}(T_{\mathcal{A},v,k}) \tag{5}$$

$$\leq \sum_{v \in V} \mathbb{P}(T_{\mathcal{A},v,1}) \cdot \sum_{k=1}^{\infty} \frac{\epsilon}{2^{1+2\lceil log k \rceil}} \tag{6}$$

$$\leq \frac{3\epsilon}{4} \cdot \sum_{v \in V} \mathbb{P}(T_{\mathcal{A},v,1}) \tag{7}$$

$$\leq \frac{3\epsilon}{4} \cdot |\{v \in V : \mathcal{A} \text{ uses } v \text{ as backing validator}\}| \tag{8}$$

Protocol 2 Π_{Dyn}: Dynamic Random Sampling.

1. **Commit:** The prover $\mathbb{R}$ sends a tuple $(\mathcal{P}_B, \mathit{Claims}, \sigma_j, op_j)$ to the verifier, where:
   - $\mathcal{P}_B$: Payload for the block B.
   - $\mathit{Claims} = [b_1, \cdots, b_n]$: a bitvector of length n. An honest prover sets $b_i = 1$ iff they possess the signature σ_i on $\mathcal{P}_B$ that verifies against pk_i.
   - σ_j: signature of the backing validator v_j.
   - op_j: opening of $\mathcal{C}$ to pk_j (i.e., the Merkle co-path).
2. **Challenge:** Verifier increments $u(e, pk_j) := u(e, pk_j) + 1$ corresponding to the current epoch e and validator with public-key pk_j. Verifier checks if *Claims* has at least $n - f$ indices set to 1s, else it terminates. If the signature op_j opens to C and σ_j is valid, then the verifier samples $m' = m + 1 + 2 * \lceil log_2(u_{e,v}) \rceil$ indices $i_1, \ldots i_{m'}$ where each i_k is chosen uniformly at random (using random function $\mathcal{R}$) from the positions of bits in *Claims* set to 1s. Here m is a statically chosen security parameter.
3. **Response:** Identical to Π_{Int}, the prover sends signatures, public keys and openings for the m' random indices requested in *challenge*-phase.
4. **Verify:** Identical to Π_{Int}, the verifier checks the signatures and validates the openings for the response. If all checks pass, the state is updated:
   - $\mathit{latestSyncedBlockHeight} := \mathit{height}(B)$
   - $\mathit{latestSyncedPayload} := \mathcal{P}_B$
   - delete entries in u for all epochs preceding B's epoch

   Else, B is stale and verifier's state remains unchanged.

Figure 2 Π_{Dyn}: dynamically increases the number of signature checks based on the number of relay instances backed by the same validator.

Hence, for any adversary $\mathcal{A}$, the expected incentive is:

$$E_{\mathcal{A}} = \mathbb{P}(W_{\mathcal{A}}) \cdot \mathcal{M} + SE_{\mathcal{A}} \cdot (-s) \tag{9}$$

$$\leq |\{v \in V : \mathcal{A} \text{ uses } v \text{ as backing validator}\}|(\frac{3s}{4} - s) \tag{10}$$

In fact, the expectation decreases linearly w.r.t the number of backing validators used by the adversary. ◀

4.2 Griefing Attacks are Expensive

Since initiating Π_{Dyn} is permissionless, an adversary can attempt to grief honest relayers by intentionally inflating the dynamic security parameter. This results in increased network workload and added costs, however, we show such griefing attacks are expensive and hence impractical for a rational adversary. Assuming the mapping $u(e, v)$ on the destination chain is publicly observable, an honest relayer in the *commit* phase can always pick a validator with the least usage, i.e., $\text{argmin}_v\, u(e, v)$ in the epoch e. Thus, an adversary with objective of griefing honest relayers has to uniformly increase the *usage counter* (u) across all possible *backing* validators for the epoch. Let C_{init} be the cost for a relayer to initiate Π_{Dyn} and C_{ver} be the cost of checking each additional signature in the *verify* phase. $\mathcal{O}(2^{x-1} \cdot |V|)$ need to be initiated by the adversary for increasing the security parameter by only x. Recall that the security parameter grows by $2\lceil log_2 u(e, v) \rceil$. We compute the griefing factor GF [14] (ratio of additional costs on victims to cost incurred by the attacker) for the above optimal strategy.

$$GF(x) = \frac{x \cdot C_{\text{ver}}}{2^{x-1} \cdot C_{\text{init}} \cdot \frac{2}{3}|V|} \tag{11}$$

Therefore, the griefing factor asymptotically drops exponentially for as the attack prolongs. In most practical scenarios, $C_{\text{ver}} \approx C_{\text{init}}$ if not $C_{\text{ver}} \ll C_{\text{init}}$, resulting in low griefing factor for even small values of x. Moreover, the griefing attacks do not extend beyond the epoch in which they are launched, as the usage counter u is reset every epoch.

4.3 Safety and Liveness

Theorem 3 proves that Π_{Int} is ϵ-sound for a single interaction and theorem 6 shows that Π_{Dyn} is unconditionally safe with soundness error ϵ within the same epoch. We do not make any trust assumptions on the relayer for the safety results. However, theorem 4 in conjunction with one honest relayer ensures liveness. Note that our soundness depends on validators being slashed for signing malicious payloads. Similar to Π_{Int}, soundness of Π_{Dyn} is affected in the highly unlikely case where blocks relayed for a period longer than the unbonding period (described in Section 3.4). The overall requirements for dynamic sampling is similar to changes adopted in the interactive protocol

5 BEEFY: Polkadot-Ethereum Bridge

The protocol Π_{Dyn} has been instantiated to implement a trustless and decentralized bridge BEEFY from Polkadot to Ethereum and is currently live. This section outlines the key design decisions and security considerations. We start with describing how a light-weight justification layer can be modularly added on top of any deterministic finality gadget without modifying the underlying consensus mechanism.

5.1 Light-weight Succinct Finality on Polkadot

Polkadot [13] uses GRANDPA [54], a deterministic finality gadget, for finalising blocks. GRANDPA is designed for secure and fast finalisation for the network, but was not primarily designed for light clients. In particular, finality proofs for GRANDPA are large (votes on forks not blocks) and light clients are required to maintain forks to follow finality. We argue that optimising the on-chain verifier workload is a priority. Hence, adding a light-weight finality layer that requires all validators to sign on the source-chain is a fair trade-off if it facilitates more efficient on-chain verification using destination-chain randomness. We design BEEFY [7] as an additional light-weight finality layer on GRANDPA such that:

- BEEFY justifications satisfy the properties in Section 2.1
- BEEFY uses the ECDSA (`secp256k1` [61]) signature scheme which is efficiently verifiable on-chain on Ethereum.
- The validator sets for BEEFY and GRANDPA are identical for each epoch.

In our current design, BEEFY finality lags GRANDPA finality by a few seconds. Note that it is not necessary for every Grandpa finalised block to be BEEFY finalised. The payload is root of an MMR (Merkle Mountain Range) capturing the state of Polkadot, allowing efficient append operations as the chain and state grow [56]. MMRs also allow more efficient block inclusion and ancestry checks on the verifier side [12].

We now describe at a high-level the BEEFY protocol in a partially synchronous setting. The protocol is designed to be modular and can be plugged into PoS chains with any deterministic finality gadget. We extend the consensus mechanism with an extra voting round. Each validator has a local belief of the following:

1. Highest GRANDPA finalized block number (G).
2. Highest BEEFY finalized block number (B).
3. Last block of the previous epoch $n-1$ (E^{n-1}).

Initially, they are all set to the genesis block. GRANDPA is independently finalising blocks and BEEFY's objective is to produce justifications for a subset of those blocks. Since GRANDPA does not allow forks, BEEFY validators are essentially voting on a single chain. Hence, the crucial part of the protocol is to determine the block height to vote on.

The protocol runs in rounds from the viewpoint of validators. A *BEEFY round* is an attempt by validators to produce a BEEFY Justification. Round number is defined as the block height of a GRANDPA finalised block that the validators vote on. The protocol expects honest validators to gossip their vote. From the local view of a validator, a round ends if either of the following events occur:

1. Validator collects $2n/3+1$ valid votes for the current round, i.e. the block obtains BEEFY justifications.
2. Validator receives a BEEFY Justification for a block higher than the currently known highest BEEFY block.

The validator proceeds to determine the new round number in both the cases. As a strawman approach, BEEFY validators could just pick the next GRANDPA finalised block. However, this would be inefficient as BEEFY finality is not required (due to MMRs) for every GRANDPA finalised block. Moreover, if for unknown reasons (e.g. network latency) the two mechanism drift apart, then there is a risk of BEEFY significantly lagging behind GRANDPA. To address this inefficiency, we introduce a more sophisticated *round selection* mechanism which ensures the gap between GRANDPA and BEEFY finalised blocks is bounded, even if the rate at which GRANDPA finalises block is exponentially more than the rate at which BEEFY finalises blocks.

Round Selection. We define two kinds of blocks from the perspective of BEEFY validators:

1. *Mandatory Blocks:* ones that MUST have BEEFY justification. Validators are required to mandatorily provide BEEFY justifications for these blocks. Last block in each epoch is defined to be a mandatory block.
2. *Non-mandatory Blocks:* validators are encouraged to finalize as many blocks as possible to enable lower latency for light clients and hence end users, without significantly lagging behind GRANDPA finalised blocks.[1]

The next round number r to participate is determined by BEEFY validator based on its local view of $\mathsf{B}, \mathsf{G}, \mathsf{E}^{n-1}$, and E^n as follows:

$$r = (1 - M) * \mathsf{E}^{n-1} \tag{12}$$

$$+ M * min(\mathsf{E}^n, (\mathsf{B} + \mathfrak{P}((\mathsf{G} - \mathsf{B} + 1)/2))) \tag{13}$$

where:

- M is 1 if the mandatory block in the previous epoch is already finalized or 0 otherwise.
- $\mathfrak{P}(x)$ returns the smallest number greater than or equal to x that is a power of two.
- min is the minimum of its arguments.

[1] Since the BEEFY authority set is the same as the GRANDPA authority set for any GRANDPA finalized block, the epoch boundaries for BEEFY are exactly the same as the ones for GRANDPA.

Intuitively, the next round number selected is either the mandatory block (with least height) without a BEEFY justification, or the highest GRANDPA-finalized block whose block number difference with B block is a power of two. The mental model for round selection is to first finalize the mandatory block and then to attempt to pick a block taking into account how fast BEEFY catches up with GRANDPA. In case GRANDPA makes progress, but BEEFY lags behind, validators are changing rounds less often to increase the chances of BEEFY catching up with GRANDPA. This is mainly due to the fact that BEEFY justifications are not required for every GRANDPA finalised block, and hence the round selection mechanism is designed to ensure that BEEFY justifications are produced for mandatory blocks, while also allowing BEEFY to catch up with GRANDPA using $\mathfrak{P}$ function.

5.2 Slashing for BEEFY misbehaviors

Slashing on-chain for BEEFY participants signing is crucial for security guarantees of Π_{Int} and Π_{Dyn}. To this end, we store recent payloads on-chain, however, a slash reporter can always generate an MMR ancestry proof [56] to show that a block (not stored on-chain) was the prefix of a recent block. The slashing conditions are straight-forward: validators in BEEFY are slashed for signing a block that is not in the current chain (GRANDPA finalised). This includes blocks with height less than or equal to the head of the current chain but are not in the chain, and blocks with a higher block number. As long as GRANDPA is safe, validators can only be slashed for voting for blocks they do not see as finalised by GRANDPA, which honest validators will never do. In theory the full-stake can be slashed but the validator can go rogue until the slash is enforced. Hence, we settle on slashing only half the stake, keeping room for slashing misbehaviors in other subsystems of the Polkadot protocol.

5.3 Verifier Accessing RANDAO on Ethereum

We describe an implementation of a light-client verifier, specifically, the *challenge* phase in Π_{Int}: The prover sends a transaction including the commit message to a smart contract which stores the message and the block number n_{init} in which the transaction is included. The verifier's challenge is derived from the RANDAO randomness from some block with number $n_{\text{challenge}}$ in the range $n_{\text{init}} + b_{\text{delay}} \leq n_{\text{challenge}} \leq n_{\text{init}} + b_{\text{delay}} + b_{\text{window}}$ for some parameters $b_{\text{delay}}, b_{\text{window}}$. b_{delay} represents the number of slots to be waited to ensure the RANDAO value is unpredictable, while b_{window} represents the number of slots of opportunity provided to the relayer to respond to the challenge after the delay. A smart contract call made by the prover included in block $n_{challenge} + 1$ records this challenge. Then the prover can send a final transaction including the response to the smart contract, which verifies according to the interactive protocol. We note that smart contracts on Ethereum have access to the RANDAO randomness from the previous block as well as the block number, as well as the slot numbers since the merge of EIP-4788 [55] as part of Dencun upgrade [45] on Ethereum Mainnet [2].

[2] The analysis is more involved if the smart contract has only access to block numbers. Block producers are assigned to slot numbers in an epoch, not block numbers. Some slot numbers will have more adversarial slots before them and hence more choices for the randomness. By choosing whether or not to produce blocks in earlier slots, which do not affect the number of choices for the randomness, the adversary may be able to ensure that the sampling block occurs at their choice of slot number.

5.4 RANDAO Bias Analysis

We instantiate the random function $\mathcal{R}$ (used in the *challenge*-phase) in Π_{Dyn} with the RANDAO beacon on Ethereum. While we assumed $\mathcal{R}$ is uniformly random and unbiasable, it is well-known that RANDAO is biasable [27, 2]. We focus on quantifying the bias for our specific objective: an attacker who wants to bias an interactive protocol (e.g., Π_{Dyn}) that uses RANDAO. We analyze the bias and how it affects our security parameter.

In this section, we consider public-coin protocols with *Commit-Challenge-Response* phases and abstract away from Π_{Dyn}. In a public-coin protocol, the verifier challenge is chosen uniformly at random from some challenge set S_c, however if the verifier is implemented on a blockchain, adversarial participants can introduce bias in the randomness. We quantify this bias as follows:

▶ **Definition 7.** *A verifier V of a public-coin protocol is μ-biasable if for any adversary, for any challenge $c \in S_c$,* $\Pr[V \text{ produces } c] \leq \mu/|S_c|$.

Assumptions. We assume that at least 2/3rd validators are honest, and the adversary cannot forge signatures or predict honest validator's randomness contributions (for Ethereum, both covered by the unforgeability of BLS signatures). As in the previous analyzes, though they don't make it explicit, we assume that an attacker is unable to prevent an honest block producer's block from being included in the chain. Though unlikely, such attacks are feasible using the attack on LMD Casper outlined in [41, 52] or by performing denial of service attacks on the block producers whose identity is public, however we exclude it from our analysis. Under the above assumptions, we derive a μ s.t the interactive protocol $\Pi_{\text{Int}}^{\text{Eth}}$ (Π_{Int} instantiated with Ethereum as the target chain) has a μ-biasable verifier protocol.

The key quantity is the tail length T, the number of slots with adversarial block producers in sequence, before the RANDAO value is used at the end of the epoch or the challenge block for BEEFY. The last honest block producer before this point produces a block that must be included, whose contribution to the randomness is random and unknown in advance. The adversarial contributions to the randomness are fixed by this point, so the adversary has the choice of publishing a block or not. This gives them 2^T choices of randomness. We build upon the TAIL-MAX strategy described in [2], and modeled as a Markov Decision Process M'_G in [2].

Adversarial Strategy. The adversary can employ TAIL-MAX continuously and wait until the current or next epoch has an exceptionally high T_n. After b_{delay} blocks for b_{delay} longer than two epoch lengths, an adversary waits until the current epoch n_{commit} (or a close epoch) before committing. Consequently, this results in several adversarial validators at the end. Then b_{delay} blocks later, the current epoch $n_{challenge}$ may still be biasable, allowing the adversary to have a more than usual chance to get many adversarial blocks before the trigger block. We can bound this bias by calculating how much the adversary can bias by running TAIL-MAX (synonymously TM) until epoch $n_{challenge} - 2$ which gives the optimal biasability.

Analysis. We denote st_{commit} as the state at the time of commit and write $\Pr_{A,st_{\text{commit}}}[E]$ as the probability that E happens with adversary (or a policy in the MDP) A. We denote by $T_n^{(s)}$, the number of adversarial slots before a slot s occurs in epoch n. Maximising this is very similar to T_n, except that the sequence of consecutive adversarially controlled blocks may extend into the previous epoch. The optimal policy $T_n^{(s)}$-MAX for the MDP M'_G for maximising $T_n^{(s)}$ runs TAIL-MAX until epoch $n - 2$.

We denote $T_n'^{(s)}$ as the length of the sequence which is in epoch n so we have

$$T_n^{(s)} = \begin{cases} T_n'^{(s)} \text{ when } T_n'^{(s)} \leq s-2 \\ s-1+T_{n-1} \text{ when } T_n'^{(s)} = s-1 \end{cases} \tag{14}$$

The distribution of $T_n'^{(s)}$ under $T_n^{(s)}$-MAX is similar to a truncated version of that of T_n under TAIL-MAX.

Concretely, we have:

$$\Pr_{\text{TAIL-MAX},st_{\text{commit}}}[T_n] \geq k] = \Pr_{T_n^{(s)}-MAX,st_{\text{commit}}}[T_n'^{(s)} \geq k] \tag{15}$$

for $k \leq s-1$ since both require a particular k slots to be adversarially controlled. We hence have the following bound:

$$E_A[2^{T_n'^{(s)}}] \leq E_{T_n^{(s)}-MAX}[2^{T_n'^{(s)}}] \leq E_{TAIL-MAX}[2^{T_n}] \tag{16}$$

Note that for the policy $T_n^{(s)}$-MAX, $T_{n_{\text{challenge}-1}}$ and $T'^{(s)}$ are independent because the Markov chains for odd and even epochs are independent.

We now compute a reasonable bound on the number of slots an adversary can feasibly wait for at the end of an epoch. From our simulations in [4], we obtained from the stationary distribution, i.e. under the continuous attack above, tail lengths of $15, 16, 17$ occur in expectation once in every 8, 24.1 and 72.3 years respectively. It seems rather expensive in terms of missed block rewards to carry out the attack for that long. Without the continuous attack, tail lengths 15,16,17 only occur in expectation every 26.2,78.6,235.8 years respectively. We fix $T_{init} = 16$ as the maximum feasible tail length for the adversary to wait.

The adversary could feasibly know the block producers in epochs $n_{\text{commit}}+1$ and $n_{\text{commit}}+2$ if the commit slot is close to the end of epoch n_{commit}. We assume that both have tail length at most T_{init}. If there is a tail length of 16 in future epoch n, then the distribution of T_{n+2d} is that of d transitions of the Markov chain from the state corresponding to tail length 16. Concretely we compute that e.g. for $d=2$, $E_{\text{TAIL-MAX}}[2^{T_{n+4}}]$ is 172.8. Note that this decreases in d.

Putting It All Together: The adversary can take T, the number of adversarially controlled slots before the RANDAO randomness is sampled for the challenge, to be the maximum value of $T_n^{(s)}$ over b_{window} slot numbers. They have 2^T choices of RANDAO samples. Each of these choices is uniformly distributed and random (though they are not independent). Thus by a union bound for a particular challenge value $c \in S_{challenge}$, they have at most $E_A[2^T]/|S_{challenge}|$ probability of getting c as the challenge. So the verifier is μ-biasable for $\mu = \max_A E_A[2^T]$. Plugging in the parameters for $\Pi_{\text{Int}}^{\text{Eth}}$, we obtain that $\Pi_{\text{Int}}^{\text{Eth}}$ is at most 864-biasable[3]. This results in an additional $\lceil log_2 \mu \rceil = 10$ signature checks in the challenge phase of Π_{Int} to negate the bias in RANDAO induced by adversarial validators on Ethereum in collusion with relayers.

5.5 Gas costs for Verification

Gas costs for verifying ECDSA signatures on Ethereum using the `ecrecover` precompile is ≈ 3500 Gwei. We compare gas costs for verifying BEEFY finality justifications using 3 different approaches across validator set sizes (assuming Polkadot's total stake rate of $\approx 50\%$, i.e., the ratio of tokens staked to total issuance):

[3] The analysis in this section is backed by numerical computations in [4].

1. Π_{Int} with no concurrency attack, but including the extra signatures required for negating RANDAO bias.
2. Naive approach of verifying all signatures.
3. SNARK circuits for verifying ECDSA signatures using UltraPlonk [58].

Table 1 Comparison of isolated gas costs (in Gwei) for verifying the ECDSA `secp256k1` signatures in the three approaches. Here $|V|$ is the validator set size of the source chain. We only consider the signatures required for accepting BEEFY finality proofs. The gas costs for UltraPLONK also include membership proofs. The values for Naive approach are computed by multiplying the gas costs for verifying a single signature by $2|V|/3 + 1$.

$\lvert V\rvert$	M/s	Π_{Int}	Naive	UltraPLONK
10	20	44811	24129	797216
10^2	$2 * 10^2$	55152	230919	816107
10^3	$2 * 10^3$	65493	2299148	811825
10^4	$2 * 10^4$	79281	22981129	819116
10^6	$2 * 10^6$	99963	2298001141	943720

As evident from the table, the costs for random sampling logarithmically increase with $|V|$ (explained in Section 3.3) and are an order of magnitude less than the SNARK verification even for $|V| = 10^6$. For $|V| \geq 10^4$, the gas requirement for Naive approach already exceeds the gas limit per block in Ethereum mainnet, making it infeasible. Further, such bulk usage for verifying signatures drives up the price of gas. As of 27/11/2024, each BEEFY consensus update on Ethereum mainnet costs 0.014 ETH (50USD).

6 Applying Fiat-Shamir Heuristic

As a natural extension, the Fiat-Shamir heuristic can be applied to transform the interactive protocol Π_{Int} into a non-interactive proof of knowledge protocol Π_{FS}, enabling the generation of a compact certificate that convinces the verifier that at least one honest validator has signed the payload. Albeit in a more general setting, Compact Certificates for Collective Knowledge introduced in [40] tackles a similar problem. Their non-interactive protocol is in spirit similar to our work but caters to a setting with weighted votes. Our work leverages on-chain randomness and crypto-economic arguments to drastically reduce the on-chain costs compared to Compact Certificates [44] for real-world PoS networks provisioning slashing mechanism. In the compact certificates approach, the randomness sampled in *challenge* phase of Π_{Int} is replaced by the prover computing a hash on all publicly know data to the verifier. Π_{Int} can similarly be transformed such that the prover uses a cryptographic hash function h over $\mathcal{C}$, $\mathcal{P}_B$, *Claims*. The security parameter (number of signatures in the certificate to be checked by verifier) in Π_{FS} purely depends on the assumed hash power of the adversary to break h. Assuming that an adversary can query $Q = 2^q$ hashes, Theorem-1 from [40], shows that $m + q$ signatures are required to be checked in the certificate for Π_{FS} to achieve a soundness error of 2^{-m}. In contrast, Π_{Int} requires only m signature checks for the same soundness error 2^{-m}.

To ensure 256-bit security for Π_{FS}, $m + q$ can be set to 256. However, can we have a more realistic bound on the hash power of adversary? Interestingly, we can rely on the rationality of the adversary. We assume that the amount of value gained in attacking the bridge (Π_{FS}) is less than the market cap of Bitcoin. If the number of hashes in expectation required to mine a bitcoin block, should if used to brute force Π_{FS} gives no high a probability for an

attack than the fraction of all bitcoin given as a mining reward currently, it is rational to use any hash power to mine bitcoin rather than attack the bridge. Since mining a block requires getting a 256 bit hash lower than the CurrentTarget, we can bound the maximum hash power Q of an adversary (see Equation 17). Plugging in parameters with the values at the time of writing, we obtain $q \leq 101$, where $2^q = Q$.

$$Q \leq \frac{2^{256}}{\text{CurrentTarget}} \cdot \frac{\text{CurrentBTCSupply}}{\text{BlockReward}} \tag{17}$$

Table 2 compares the signature checks required by verifier for Polkadot and Ethereum for the various versions of our protocol and Altair.

Table 2 Comparison of Signature checks required in Π_{Int} with Fiat-Shamir versions and Altair for PoS networks with varying M/s values. M denotes the market cap and s denotes the minimum stake of validator. Comparison of signature checks required by the protocols Π_{Int}, Π_{FS} with 256-bit security, and $\Pi_{\text{FS}}^{\ddot{\mathbb{B}}}$ with hash power bounded via rationality assumption relative to bitcoin mining.

Network	$log_2(M/s)$	Π_{Int}	Π_{FS}^{256}	$\Pi_{\text{FS}}^{\ddot{\mathbb{B}}}$	Altair
Polkadot	576	10	256	111	201
Ethereum	3761875	21	256	122	512
Algorand	460	9	256	110	61
Binance SC	260	9	256	110	30

7 Related Work

While there has been advances in application specific (e.g. token swaps and payment channels) bridging solutions [49, 39, 64, 36], we focus on approaches supporting functionality of following finality for PoS networks. Most bridge architectures [48, 6, 62] involve a centralised and trusted intermediary (via a multi-sig) that run full-nodes of the bridged chains. Time and again, the trusted centralised entities have been compromised [5] resulting in massive financial implications. In essence, centralised bridges are optimistic protocols where the claims by intermediary are accepted by the target chain without verifying the finality of the claim on the source chain. On the other end of the spectrum, the target chain can verify the full quorum certificate of the claims finality, however, this approach is expensive and not scalable with respect to validator set size on the source chain. Our work bridges the gap between these two extremes by proposing a light-client protocol that is efficient and secure without trust assumption on intermediaries. Our random sampling protocol introduces negligible and controllable soundness error with the objective of drastically reducing on-chain computation and storage costs compared to verification of full certificates. The soundness error falls exponentially with respect to the security parameter m (i.e. # of signature checks) and hence can be arbitrarily reduced by increasing m. Assuming a rational adversary, the crypto-economics arguments introduced in Section 3.3 justifies tolerating a higher soundness error, hence further improving the efficiency. Within our crypto-economic framework, the security parameter m itself depends logarithmically on the validator set size, therefore, our protocol scales well with large validator sets.

Source-chain Randomness. Ethereum's Altair upgrade [3] introduced a light-client protocol based based on *sync-committees* [26], a significantly smaller subset of validators (currently 512) responsible for attesting the finality of blocks. The sync-committee is randomly chosen

for each epoch ($\approx$ 27 hours) using RANDAO beacon, and remains unchanged for a given epoch. Similar to Altair, Π_{Dyn} does not require any custom crypto or SNARK primitives on the prover or verifier side, hence can be integrated readily on existing blockchains, and particularly efficient for large validator sets. We skip the already know issues with Altair [31, 44] and focus on the advantages of Π_{Dyn} specifically for bridge applications which demand higher security guarantees and efficiency requirements:

1. For the same soundness error, the signature checks required in Π_{Dyn} are significantly lower than Altair. The current sync-committee size of 512 guarantees soundness error of $8 * 10^{-54}$ and requires 342 signature checks, while Π_{Dyn} requires only 172. Detailed further in Appendix A.
2. Π_{Dyn} is more resilient to adaptive adversaries. The sync-committee for the upcoming epoch is known in advance, hence the adversary can use the whole epoch duration to adaptively corrupt members. In contrast, the adversary has a much shorter duration, the difference between the *challenge* and *response* phase (configured per scenario), in Π_{Dyn} to corrupt adaptively.
3. All bridges would be affected if the sync-committee is compromised, while bridges using Π_{Dyn} are independent as the randomness is sourced on the destination chain. An attack on one bridge does not affect the rest.
4. Π_{Dyn} lets the verifier configure the security parameter enabling it to trade-off security and efficiency depending on the crypto-economic setting.

Approximate Lower-Bound Arguments [18] by Chaidos *et al.* is another protocol which uses source chain randomness to generate comparct certificates. Our work focuses on the rational setting with slashing, where we use crypto-economics to argue that a larger soundness-error is tolerable, hence improving efficiency. The fair comparison with Dynamic Telescope (the most efficient version of the protocols described in [18]) would be to our interactive protocol also aiming for negligible 2^{-128} soundness-error. For soundness-error 2^{-128}, assuming our destination-chain randomness is unbiasable, we require 128 signatures in contrast to 159 (setting parameters used in Fig-2 [18]) for Decentralized Telescopes [18]. Our interactive protocol leveraging destination-chain randomness requires $\leq \lambda_{sec} + \delta$ signature checks, where δ accounts for negating randomness bias. Our implementation also uses an optimization (Section 3.2) where the sampling is performed without replacement but the efficiency gained by this optimization depends on the validator set size.

Optimistic Techniques With Fishermen. Several bridges like Nomad [42] and Near's Rainbow [46] are examples of optimistic protocols (aka Claim and Challenge schemes) that leverage fraud-proofs [16]. Typically, the light client optimistically accepts a state without verifying and relies on economically incentivised Fishermen/WatchDogs to detect invalid updates. Unlike our approach, the relayers and fishermen need to be staked to avoid spamming and forcing all signatures being verified by making false positive challenges. Further, the security of such bridges depend on the censorship resistance of target chain during the challenge period. Dynamic transaction fees on the target chain worsen the issue, resulting in high challenge rewards and stakes. Moreover, the challenge period needs to be long trading off latency and security. In comparison, our solution requires less data since all signatures do not need to be published on chain and our bridge has lower confirmation latency. If the target chain is censored, our protocol loses liveness rather than safety but optimistic protocols also lose safety.

SNARK Based Bridges. Recently, embedding SNARK-based on-chain light-clients has been quite popular approach for trustless bridges [10, 60, 63]. Accountable Light Client system introduce in [24] guarantees that a large number (e.g.1/3rd) misbehaving validators can be identified when a light client is misled, a crucial property for PoS blockchains. These solutions require custom SNARKs and cryptographic primitives like aggregatable signatures which are difficult to implement in existing networks. Our approach is simpler and a secure implementation is less involved [21].

Threshold Signatures. An alternative approach is using a threshold signature scheme with a single public key for all validators, as adopted by Dfinity [33]. These schemes typically use secret sharing for the secret key, which has two main drawbacks. First, they require a communication-intensive distributed key generation protocol for setup, which is challenging to scale for large validator sets (even with 100 validators). Despite recent advancements [34, 33, 32], implementing such setups across a large peer-to-peer network remains difficult and may need repeating when the validator set changes. Second, secret sharing-based threshold signatures do not reveal which subset of validators signed, lacking accountability. Dfinity [33] employs a re-shareable BLS threshold signature, maintaining the same public key even with validator set changes. This provides a constant-size proof for the verifier but fails to identify misbehaving validators or the specific validator set responsible, as the signature remains the same for any threshold subset. Gazi et al. intorduce a new cryptographic primitive Ad-Hoc Threshold Multisignatures (ATMS) [30] to enable cross-chain certifications for PoS networks, where the signers are selected on the fly from an existing key set. Their protocol is designed to be efficient and secure against adaptive adversaries, i.e., the adversary can corrupt validators adaptively during the protocol execution. Our interactive protocol and the Compact Certificates [40] satisfy relaxations of the ATMS definition applied to the whole validator set. A key difference is that the side chain protocol described in [30] uses source-side committees, similar to Altair though adaptively secure. We believe adaptive security for our protocol would require key evolving signatures as well as a timing guarantee that ensures the randomness is only predictable after honest validators have forgotten their keys.

Randomness Beacons. There is a series of work on randomness beacons for blockchains [23]. Our work directly benefits from advances in unbiasabale and unpredictable randomness generation by blockchains. Schemes based on Publicly Verifiable Secret Sharing [9, 51] and Verifiable Delay Functions [37, 50] that are immune against *last-revealer attacks* directly improves the efficiency of our random sampling protocol, since we do not need to increase the security parameter to negate the bias.

8 Conclusion and Future Work

We presented an interactive light-client protocol Π_{Dyn} for PoS blockchains using on-chain randomness. Π_{Dyn} leverages crypto-economic arguments to drastically improve efficiency (on-chain computation usage) without compromising security. The practicality of our protocol is demonstrated by instantiating a trustless and decentralized bridge between Polkadot and Ethereum. As future work, we plan to design incentive mechanism's for the relayer market. We are interested in settings where the relayers are incentivised by *public good funding* (DAO treasuries) [38, 59, 43] as well as fees generated by users of the bridge.

References

1 Shresth Agrawal, Joachim Neu, Ertem Nusret Tas, and Dionysis Zindros. Proofs of proof-of-stake with sublinear complexity. *CoRR*, abs/2209.08673, 2022. doi:10.48550/arXiv.2209.08673.

2 Kaya Alpturer and S. Matthew Weinberg. Optimal randao manipulation in ethereum, 2024. doi:10.48550/arXiv.2409.19883.

3 Altair light client – sync protocol. https://ethereum.github.io/consensus-specs/specs/altair/light-client/sync-protocol/, 2024.

4 Anonymised Artifact. Randao analysis. https://anonymous.4open.science/r/fc25-randao-analysis-AFCD/, 2025.

5 André Augusto, Rafael Belchior, Miguel Correia, André Vasconcelos, Luyao Zhang, and Thomas Hardjono. Sok: Security and privacy of blockchain interoperability. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3840–3865, 2024. doi:10.1109/SP54263.2024.00255.

6 Axelar: Connecting applications with blockchain ecosystems. https://www.axelar.network/whitepaper, 2021.

7 Polkadot Specifications: Beefy. https://spec.polkadot.network/sect-finality#sect-grandpa-beefy, 2023.

8 Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, 2016. doi:10.1007/978-3-662-53644-5_2.

9 Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. RandPiper – reconfiguration-friendly random beacons with quadratic communication. Cryptology ePrint Archive, Paper 2020/1590, 2020. doi:10.1145/3460120.3484574.

10 Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. *IACR Cryptol. ePrint Arch.*, page 352, 2020. URL: https://eprint.iacr.org/2020/352.

11 Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018. arXiv:1807.04938.

12 Benedikt Bunz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 928–946. IEEE, 2020. doi:10.1109/SP40000.2020.00049.

13 Jeff Burdges, Alfonso Cevallos, Peter Czaban, Rob Habermeier, Syed Hosseini, Fabio Lama, Handan Kilinç Alper, Ximin Luo, Fatemeh Shirazi, Alistair Stewart, and Gavin Wood. Overview of polkadot and its design considerations. *CoRR*, abs/2005.13456, 2020. arXiv:2005.13456.

14 Vitalik Buterin. A griefing factor analysis model. https://ethresear.ch/t/a-griefing-factor-analysis-model/2338, 2018.

15 Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017. arXiv:1710.09437.

16 Ran Canetti, Ben Riva, and Guy N. Rothblum. Refereed delegation of computation. *Information and Computation*, 226:16–36, 2013. Special Issue: Information Security as a Resource. doi:10.1016/j.ic.2013.03.003.

17 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In Margo I. Seltzer and Paul J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999. URL: https://dl.acm.org/citation.cfm?id=296824.

18 Pyrros Chaidos, Aggelos Kiayias, Leonid Reyzin, and Anatoliy Zinovyev. Approximate lower bound arguments. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part IV*, volume 14654 of *Lecture Notes in Computer Science*, pages 55–84. Springer, 2024. doi:10.1007/978-3-031-58737-5_3.

19 Chainalysis. Vulnerabilities in cross-chain bridge protocols emerge as top security risk. https://www.chainalysis.com/blog/cross-chain-bridge-hacks-2022/, 2022.

20 Stefanos Chaliasos, Jens Ernstberger, David Theodore, David Wong, Mohammad Jahanara, and Benjamin Livshits. Sok: what don't we know? understanding security vulnerabilities in snarks. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, SEC '24, USA, 2024. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/chaliasos.

21 Stefanos Chaliasos, Jens Ernstberger, David Theodore, David Wong, Mohammad Jahanara, and Benjamin Livshits. Sok: What don't we know? understanding security vulnerabilities in snarks. *CoRR*, abs/2402.15293, 2024. doi:10.48550/arXiv.2402.15293.

22 Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952. doi:10.1214/aoms/1177729330.

23 Kevin Choi, Aathira Manoj, and Joseph Bonneau. Sok: Distributed randomness beacons. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 75–92, 2023. doi:10.1109/SP46215.2023.10179419.

24 Oana Ciobotaru, Fatemeh Shirazi, Alistair Stewart, and Sergey Vasilyev. Accountable light client systems for pos blockchains. *IACR Cryptol. ePrint Arch.*, page 1205, 2022. URL: https://eprint.iacr.org/2022/1205.

25 Ronald Cramer. Modular design of secure yet practical cryptographic protocols. *Ph. D.-thesis, CWI and U. of Amsterdam*, 2, 1996.

26 Ben Edgington. One page annotated spec. https://eth2book.info/capella/annotated-spec/, 2022.

27 Ben Edgington. *Upgrading Ethereum*, chapter RANDAO Biasability, pages 152–153. Ben Edgington, 2023. URL: https://eth2book.info/capella/part2/building_blocks/randomness/#randao-biasability.

28 Over $1 billion stolen from bridges so far in 2022 as harmonys horizon bridge becomes latest victim in $100 million hack, 2022. URL: bit.ly/3fvlIME.

29 Nomad loses $156 million in seventh major crypto exploit of 2022. https://hub.elliptic.co/analysis/nomad-loses-156-million-in-seventh-major-crypto-bridge-exploit-of-2022/, 2022.

30 Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 139–156. IEEE, 2019. doi:10.1109/SP.2019.00040.

31 Vincent Geddes and Aidan Musnitsky. Snowfork's analysis of sync committee security. https://forum.polkadot.network/t/snowforks-analysis-of-sync-committee-security/2712/10, 2023.

32 Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. ePrint 2021/1397, 2021. URL: https://eprint.iacr.org/2021/1397.

33 Jens Groth. Non-interactive distributed key generation and key resharing. ePrint 2021/339, 2021. URL: https://eprint.iacr.org/2021/339.

34 Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. ePrint 2021/005, 2021. URL: https://eprint.iacr.org/2021/005.

35 Mor Harchol-Balter. Introduction to probability for computing. https://www.cs.cmu.edu/~harchol/Probability/chapters/chpt18.pdf, 2024.

36 Rongjian Lan, Ganesha Upadhyaya, Stephen Tse, and Mahdi Zamani. Horizon: A gas-efficient, trustless bridge for cross-chain transactions. *CoRR*, abs/2101.06000, 2021. arXiv:2101.06000.

37 Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Paper 2015/366, 2015. URL: https://eprint.iacr.org/2015/366.

38 Lu Liu, Sicong Zhou, Huawei Huang, and Zibin Zheng. From technology to society: An overview of blockchain-based dao. *IEEE Open Journal of the Computer Society*, 2:204–215, 2021. doi:10.1109/OJCS.2021.3072661.

39 Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. URL: https://www.ndss-symposium.org/ndss-paper/anonymous-multi-hop-locks-for-blockchain-scalability-and-interoperability/.

40 Silvio Micali, Leonid Reyzin, Georgios Vlachos, Riad S. Wahby, and Nickolai Zeldovich. Compact certificates of collective knowledge. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 626–641. IEEE, 2021. doi:10.1109/SP40001.2021.00096.

41 Michael Neuder, Daniel J. Moroz, Rithvik Rao, and David C. Parkes. Low-cost attacks on ethereum 2.0 by sub-1/3 stakeholders. *CoRR*, abs/2102.02247, 2021. arXiv:2102.02247.

42 Nomad protocol. https://docs.nomad.xyz/the-nomad-protocol/overview, 2021.

43 Polkadot Wiki: Treasury. https://wiki.polkadot.network/docs/learn-polkadot-opengov-treasury, 2024.

44 James Preswitch. Altair has no light client. https://prestwich.substack.com/p/altair, 2023.

45 Ethereum Foundation Protocol Support Team. Dencun mainnet announcement. https://blog.ethereum.org/2024/02/27/dencun-mainnet-announcement, 2024.

46 What is the rainbow bridge? https://doc.aurora.dev/bridge/introduction/, 2024.

47 Rekt leaderboard. https://rekt.news/tr/leaderboard/, 2024.

48 Ronin bridge documentation. https://docs.roninchain.com/apps/ronin-bridge, 2024.

49 Giulia Scaffino, Lukas Aumayr, Mahsa Bastankhah, Zeta Avarikioti, and Matteo Maffei. Alba: The dawn of scalable bridges for blockchains. *IACR Cryptol. ePrint Arch.*, page 197, 2024. URL: https://eprint.iacr.org/2024/197.

50 Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. RandRunner: Distributed randomness from trapdoor VDFs with strong uniqueness. Cryptology ePrint Archive, Paper 2020/942, 2020. URL: https://eprint.iacr.org/2020/942.

51 Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. Hydrand: Efficient continuous distributed randomness. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 73–89, 2020. doi:10.1109/SP40000.2020.00003.

52 Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In Ittay Eyal and Juan A. Garay, editors, *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 560–576. Springer, 2022. doi:10.1007/978-3-031-18283-9_28.

53 Snowbridge: A trustless bridge between polkadot and ethereum. https://github.com/snowfork/snowbridge/, 2024.

54 Alistair Stewart and Eleftherios Kokoris-Kogia. GRANDPA: a byzantine finality gadget. *CoRR*, abs/2007.01560, 2020. arXiv:2007.01560.

55 Alex Stokes, Ansgar Dietrichs, Danny Ryan, Martin Holst Swende, and lightclient. Eip-4788: Beacon block root in the evm. https://eips.ethereum.org/EIPS/eip-4788, 2022.

56 Peter Todd. Making utxo set growth irrelevant with low-latency delayed txo commitments. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2016-May/012715.html, 2016.

57 Dmytro Tymokhanov and Omer Shlomovits. Alpha-rays: Key extraction attacks on threshold ECDSA implementations. Cryptology ePrint Archive, Paper 2021/1621, 2021. URL: https://eprint.iacr.org/2021/1621.

58 Ultraplonk verifier by aztec. https://github.com/AztecProtocol/barretenberg/tree/master/sol/src/ultra, 2024.

59 Paul Van Vulpen and Slinger Jansen. Decentralized autonomous organization design for the commons and the common good. *Frontiers in Blockchain*, 6:1287249, 2023. doi:10.3389/FBLOC.2023.1287249.

60 Psi Vesely, Kobi Gurkan, Michael Straka, Ariel Gabizon, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, and Eran Tromer. Plumo: An ultralight blockchain client. In Ittay Eyal and Juan A. Garay, editors, *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 597–614. Springer, 2022. doi:10.1007/978-3-031-18283-9_30.

61 Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

62 Wormhole architecture overview. https://wormhole.com/docs/learn/fundamentals/architecture/, 2024.

63 Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 3003–3017. ACM, 2022. doi:10.1145/3548606.3560652.

64 Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William J. Knottenbelt. XCLAIM: trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 193–210. IEEE, 2019. doi:10.1109/SP.2019.00085.

A Efficiency Comparison with Altair

Let X be the random variable denoting the number of malicious validators being picked in the Sync-Committee. Since the validator set on Ethereum is $\geq 10^6$, we can approximate the hypergeometric distribution of X to $X \sim Bin(512, p)$, where $p = 1/3$ is the probability of picking a malicious validator from byzantine assumptions. The probability that a majority of malicious validators comprise the sync-committee can be bounded using Pretty Chernoff's inequality [22, 35] as follows:

$$Pr(X - E[X] \geq \lambda) \leq e^{-\frac{2\lambda^2}{n}}$$

Substituting $\lambda = n/3$ and $E[X] = n/3$, we obtain soundness error bound

$$Pr(X \geq 2n/3) \leq e^{-2n/9}$$

Plugging the values for Ethereum, we obtain a bound on probability the a super-majority malicious validators were elected in the sync-committee (soundness error) $Pr(X \geq 342) = 8 \times 10^{-54}$. To obtain the same soundness error via Π_{Dyn}, the number of signature checks (security parameter) required is 176 (i.e., $log_2(1/Pr(X \geq 342))$). In fact, the result can be generalised: the ratio of signature checks required in Π_{Dyn} to Altair for achieving the same soundness errors is $(log_2\ e)/3$.

4-Swap: Achieving Grief-Free and Bribery-Safe Atomic Swaps Using Four Transactions

Kirti Singh
Indian Institute of Technology Bombay, India
Institute for Development and Research in Banking Technology, Hyderabad, India

Vinay J. Ribeiro
Indian Institute of Technology Bombay, India

Susmita Mandal
Institute for Development and Research in Banking Technology, Hyderabad, India

Abstract

Cross-chain asset exchange is crucial for blockchain interoperability. Existing solutions rely on trusted third parties and risk asset loss, or use decentralized alternatives like atomic swaps, which suffer from grief attacks. Griefing occurs when a party prematurely exits, locking the counterparty's assets until a timelock expires. Hedged Atomic Swaps mitigate griefing by introducing a penalty premium; however, they increase the number of transactions from four (as in Tier Nolan's swap) to six, which in turn introduces new griefing risks. Grief-Free (GF) Swap reduces this to five transactions by consolidating assets and premiums on a single chain. However, no existing protocol achieves grief-free asset exchange in just four transactions.

This paper presents 4-Swap, the first cross-chain atomic swap protocol that is both grief-free and bribery-safe, while completing asset exchange in just four transactions. By combining the griefing premium and principal into a single transaction per chain, 4-Swap reduces on-chain transactions, leading to faster execution compared to previous grief-free solutions. It is fully compatible with Bitcoin and operates without the need for any new opcodes. A game-theoretic analysis shows that rational participants have no incentive to deviate from the protocol, ensuring robust compliance and security.

2012 ACM Subject Classification Security and privacy → Security protocols

Keywords and phrases Atomic Swaps, Griefing, Bribery, HTLC

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.32

Related Version *Extended Version*: `https://arxiv.org/abs/2508.04641` [17]

1 Introduction

The trade of cross-chain assets has been a significant problem in blockchains. Centralized exchanges such as Binance, Coinbase, and CoinDCX solve this problem by taking ownership of users' assets and exchanging them among the interested parties. Using centralized exchanges makes the exchange efficient and easy. However, the involvement of a trusted third party leads to the risk of users losing their assets, as happened in the MTGox hack [6, 16] and FTX collapse [7]. Thus, there is a need for trustless exchange protocols. Some cross-chain decentralized exchange solutions exist, such as two-way pegged tokens [3] and using a third chain [22]. However, they face some issues: pegged tokens do not give native tokens, and using a third chain increases the overall overhead for the exchange.

Atomic swaps solve the issues by providing native tokens without needing a third coordinating chain. Tier Nolan introduced it in a Bitcoin forum in 2013 [15, 14]. It typically involves four transactions, the locking of tokens by the two parties, which later claim each others' tokens before a timelock expires. Atomic swaps primarily use Hashed Time-Locked Contracts (HTLCs), which provide a mechanism to lock assets on the blockchain with a

© Kirti Singh, Vinay J. Ribeiro, and Susmita Mandal;
licensed under Creative Commons License CC-BY 4.0

hashlock and to refund the locked assets after the timelock expires. However, they can also leverage multi-signature transactions [8, 10], adaptor signatures [10, 18], and verifiable timed signatures [18] for implementation. Although Atomic Swap provides a decentralized exchange solution, it suffers from the problem of grief.

Griefing is a common problem in Atomic Swaps wherein a party who locks the tokens waits for the timelock to expire when the counterparty abandons the exchange. To deter this premiums are used as collateral. In 2021, Xue and Herlihy [21] used the concept of cross-locking of griefing premium in the two chains to solve griefing. However, this led to griefing on the premium, and the number of transactions in the swap increased to six.

In 2022, Nadahalli *et al.* proposed combining the locking of griefing premium with the assets in one transaction [13]. Thus, the problem of griefing on the premium is solved. However, they failed to combine the premium on both chains, requiring five transactions for a grief-free atomic swap. Mazumdar also solves the griefing problem by allowing the participating parties to leave the exchange whenever they like, increasing the total number of transactions to eight [11]. This leads to our question: is achieving a grief-free atomic swap in only four transactions possible?

We present the 4-Swap (or 4S, for short) protocol in two versions designed to simplify the exposition of this complex protocol. By breaking it down into incremental versions, we aim to make its functionality and security properties more accessible. We begin with 4S-v1, which operates under the simplifying assumption of zero delay in transactions being added to the blockchain. In this idealized setting, a published transaction is immediately included without waiting in the mempool, effectively eliminating the risks of bribery or censorship attacks. This version allows us to focus on the core mechanics of the protocol, which achieves grief-free execution in just four steps. It does so by leveraging cross-locking of principals and premiums, as seen in prior works [21, 13]. We introduce two novel concepts of cross-publishing of locking transactions and a new timelock in the initiating chain for early refunds. These concepts help combine the principal and griefing premiums for both parties in both chains, reducing the total number of transactions to just four.

Building on this foundation, we present the next version that removes the zero-delay assumption, thus accounting for real-world conditions where transaction delays and susceptibility to bribery attacks are practical concerns. To address this, we integrate the concept of mutual destruction from [19] coupled in both the chains, ensuring the protocol remains bribery-safe. The final version retains its simplicity and efficiency, maintaining grief-free and bribery safety within a four-step process. We do not claim the protocol to be bribery free as it does not considers the reverse bribery attacks where the miners incentivizes the involved parties to publish the conflicting transactions. The protocol's game-theoretical analysis demonstrates that it consistently follows its intended execution path under the assumption of rational behaviour from all parties. We perform a security analysis of the protocol using the UC Framework [4], and present possible redeem paths in the extended version [17], which also includes Bitcoin scripts showing that the protocol requires no new opcodes.

Our Contributions

1. We propose 4-Swap, the first grief-free and bribery-safe atomic swap protocol that uses only four transactions, matching the number of transactions in the TN-Swap.
2. We then present a game-theoretical analysis of our protocol to show that participants have no incentive to deviate from the protocol under rational behaviour.
3. We perform a security analysis of the protocol using the UC framework [4] in the extended version of this paper [17].

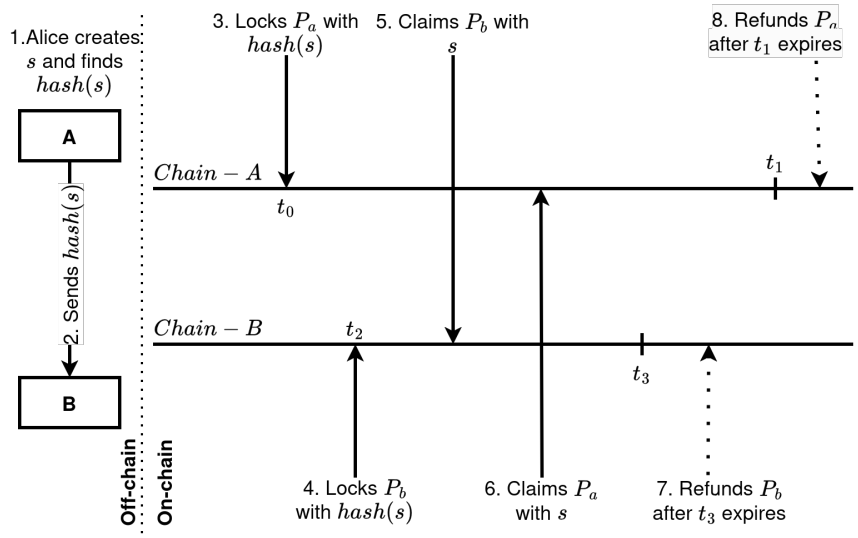


Figure 1 Atomic Swap Protocol.

2 Background

2.1 Atomic Swaps – TN Swap

Atomic swap is a decentralized method of performing cross-chain exchanges. Tier Nolan first introduced it in 2013 [15]. It typically uses two HTLCs with the same hash to lock the tokens. The parties on the successful locking claim each other's tokens before the set timeout expires in their respective chains [9].

Consider A and B, two parties interested in exchanging tokens P_a on *Chain-A* and P_b on *Chain-B*, respectively. They must follow the following steps for a successful atomic swap exchange, as shown in Figure 1.

1. A generates secret s, finds $\mathcal{H}(s)$ and sends $\mathcal{H}(s)$ to B.
2. A locks her tokens on *Chain-A* with $\mathcal{H}(s)$ setting up a refund timelock of t_1.
3. B locks his tokens on *Chain-B* with same $\mathcal{H}(s)$ setting a refund timelock of t_3.
4. A claims B's locked tokens on *Chain-B* with s.
5. B uses the revealed preimage s to claim A's tokens.
6. If A does not claim B's tokens, B can refund after t_3.
7. If B does not claim A's tokens, A can refund after t_1.

The protocol ensures that no party loses their tokens in the exchange. Moreover, the gap in two timelocks ensures that B gets enough time to claim A's token when A has already claimed B's token. Although atomic swap solves the problem of decentralized cross-chain asset exchange, it gives rise to the problem of griefing.

2.1.1 Griefing Attack

Griefing occurs when a party locks the tokens, and the counterparty exits the exchange, forcing the counterparty to wait until the refund timelock expires to retrieve their tokens.

In Figure 1, there are two possible scenarios of griefing.

- A locks tokens, but B leaves the exchange. A waits until t_1 to refund the locked tokens.
- A and B lock their tokens, but A leaves without claiming B's tokens. Since A only knows the preimage s, B waits until the refund timelock t_3 expires to get back his tokens.

2.1.2 Bribery Attacks

Malicious parties bribe the miners to censor some transactions in the blockchain to gain an unfair advantage. HTLCs suffer from bribery attacks [12]. Suppose A pays B using HTLC and bribes the miners to censor B's claim transaction until the refund timelock expires. After the timelock expires, A can refund the locked tokens, and B fails to get the tokens. Since atomic swaps also use HTLCs, they are prone to bribery attacks. Bribery usually happens by either a party putting up a conflicting transaction(like the refund in HTLC) with a very high transaction fee or putting the refund transaction along with a future transaction that takes input from the refund transaction and can be redeemed by anyone(particularly miners).

In Figure 1, there are two possible scenarios of bribery attacks.

- A and B lock their tokens, but B bribes the miners to censor A's claim transaction until t_3. Since B can see A's claim transaction, he fetches s and claims A's tokens P_a. After t_3, he refunds his tokens P_b and A gets nothing.
- A claims B 's tokens after both successfully locked them. She then bribes the miners to censor B 's claim transaction until t_1. After t_1, A refunds her locked tokens P_a, and B gets nothing.

2.1.3 Bribery Attacks Solutions for HTLCs

In 2021, Tsabary *et al.* [19] proposed MAD-HTLC to remove bribery in HTLC. They modified the HTLC by adding a premium to be locked by the sender in addition to the principal value. The idea was to allow the miners to slash all the locked tokens, disincentivizing the parties to attempt bribery.

Suppose there are two parties where A is paying B using HTLC. The solution works as follows:

- A locks X (principal) and Y (premium) with $H(pre_a)$. Y can be refunded after timelock.
- To refund X, A must reveal pre_b after the timelock.
- B claims X before the timelock by revealing pre_a.
- If A attempts to censor B's claim by bribing miners with a high-fee refund transaction, miners can extract pre_b from A's refund transaction. Anyone with pre_a, pre_b can claim $X + Y$, so miners take the funds. Bribery fails; A gains nothing.

The sender can refund the premium after the timelock expires. Also, to refund the principal tokens locked by the hash of pre_a, the sender has to release a new preimage pre_b after the timelock expires. By bribing miners, the sender can still try to censor the receiver's claimed transaction, which reveals pre_a. However, if the sender tries to bribe the miners by putting in a refund transaction with very high transaction fees, the miners can fetch the pre_b from it. Since MAD-HTLC allows anyone with pre_a and pre_b to claim the tokens and the premium, miners can create their transaction and try to create a block with it. Thus, the sender does not get any benefit from bribing.

2.1.4 Reverse Bribery Attacks

MAD-HTLC does not address scenarios where miners incentivize protocol participants to behave maliciously, a situation known as a reverse bribery attack. This attack was introduced by Wadhwa *et al.* [20] in 2022, along with the proposal of He-HTLC, a protocol designed to mitigate such threats. Unlike MAD-HTLC, He-HTLC introduces a delay in the sender's refund process and allows multiple miners to collectively punish a malicious sender.

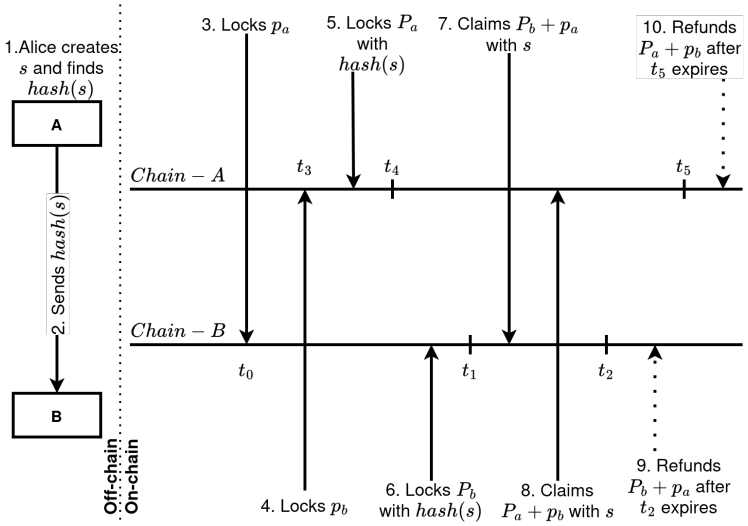


Figure 2 Hedged Atomic Swap Protocol.

Additionally, when a miner redeems funds using pre_a and pre_b, they receive only a portion of the locked tokens; the remainder is burned. This partial reward mechanism discourages collusion between miners and protocol participants.

However, He-HTLC does not offer a construction for atomic swaps within its framework. Rapidash [5], also published in 2022, builds on similar ideas. It introduces a two-step refund mechanism and extends the protocol to support atomic swaps. While this approach solves the reverse bribery attacks, it introduces increased on-chain transaction overhead and fails to be grief-free as one party can abandon the protocol after the counterparty has already locked their assets.

3 Related Works

We now discuss the related work of minimizing the number of on-chain transactions of atomic swaps while keeping it grief-free. Both works assume that parties have some tokens in the chain they want to receive tokens to safeguard from griefing attacks.

3.1 Hedged Atomic Swap

Xue and Herlihy [21] in 2021 proposed a Hedged Atomic Swap protocol that tried to solve the problem of griefing attacks using griefing premiums. The parties have to cross-lock the premium amounts with the principal amounts of the counterparty. The premiums serve as an amount to compensate the parties from griefing. Thus, it introduces the assumption that parties need a small amount of tokens in the other chain where they want to exchange them. The cross-locking helps get the refund premium and claim principal in one step. However, since the protocol requires premium locking, the number of transactions rose to six, two more than the TN-Swap.

Consider A and B, two parties interested in exchanging tokens P_a on $Chain\text{-}A$ and P_b on $Chain\text{-}B$, respectively. A must have p_a on $Chain\text{-}B$ and B must have p_b on $Chain\text{-}A$ as griefing premium. They must follow the following steps for a successful atomic swap exchange, as shown in Figure 2.

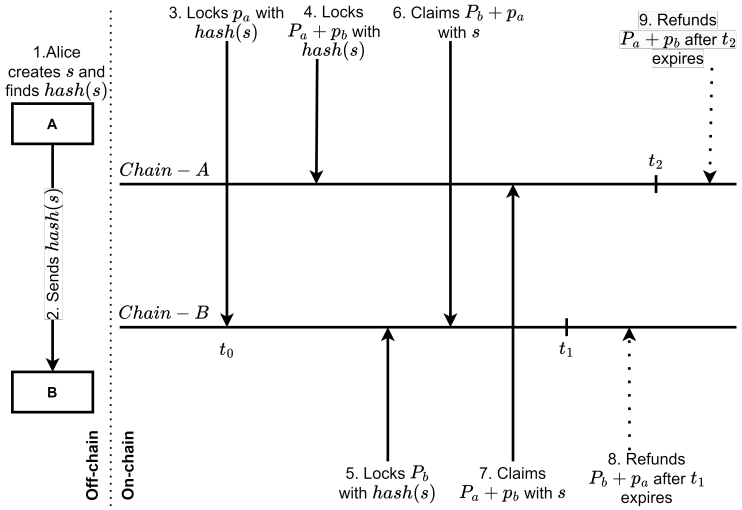


Figure 3 Grief-Free Atomic Swap Protocol.

1. A generates a secret s, finds $\mathcal{H}(s)$.
2. A sends $\mathcal{H}(s)$ to B.
3. A locks her premium p_a on *Chain-B* with $\mathcal{H}(s)$ setting up a timeout of t_1 and refund timelock of t_2 on *Chain-B*. B should lock his principal before t_1 on *Chain-B*.
4. B locks his premium p_b on *Chain-A* with $\mathcal{H}(s)$ setting up a timeout of t_4 and a refund timelock of t_5 on *Chain-A*. t_4 is the timeout for A to lock her principal.
5. A locks her tokens on *Chain-A* with $\mathcal{H}(s)$ before t_4. This locks $P_a + p_b$ until timelock t_5.
6. B locks his tokens on *Chain-B* with $\mathcal{H}(s)$. This locks $P_b + p_a$ until timelock t_2.
7. A claims B's principal P_b and her premium p_a on *Chain-B* with s.
8. B uses the revealed preimage s to claim A's principal P_a and his premium p_b.
9. If A does not claim B's tokens, B can refund $P_b + p_a$ after t_2 expires.
10. If B does not claim A's tokens, A can refund $P_a + p_b$ after t_5 expires.

Hedged Atomic Swap handles the two griefing cases mentioned in Section 2.1.1. For the first case where A locks P_a before t_4 and B leaves the exchange, A can refund $P_a + p_b$ after t_5 because B cannot refund his premium after A locks his principal. Since B has not locked his principal, A can refund her premium p_a after t_1. Thus, B loses p_b to A, disincentivizing him from leaving the exchange. For the second case, where A refuses to claim after A and B lock their principal, A and B can be refunded after t_5 and t_2, respectively. Thus, A gets $P_a + p_b$ and B gets $P_b + p_a$ by refunding. To ensure that A is disincentivized from leaving the exchange for this case, p_a is kept larger than p_b. Thus, A loses $p_a - p_b$ by leaving the exchange. However, this introduces the problem of griefing on premiums as the B can leave the swap after A locks her premium, and similarly, A can leave the swap after B locks his premium. This leads to the waiting until their respective refund timelock expires.

3.2 Grief-Free Atomic Swap

In 2022, Nadahalli *et al.* [13] proposed Grief-free atomic swaps that guarantee no griefing while reducing the on-chain transactions to five. The idea is to combine the locking of the premium with the principal. However, it could not match the number of transactions in the TN-Swap as the combination was done only in one chain.

Again, consider the same setup as in the Section 3.1. The following steps help achieve a grief-free atomic swap in five steps, as shown in Figure 3.

1. A generates secret s, finds $\mathcal{H}(s)$.
2. A sends $\mathcal{H}(s)$ to B.
3. A locks her premium p_a with $\mathcal{H}(s)$ setting up a refund timelock of t_1 on *Chain-B*.
4. A and B create a locking transaction $P_a + p_b$ and A publishes it on *Chain-A*. This also sets up refund timelock t_2. If B leaves before creating this, A can refund p_a by using s.
5. B locks his tokens with same $\mathcal{H}(s)$. This locks $P_b + p_a$ on *Chain-B* until timelock t_1.
6. A claims B's principal P_b and her premium p_a on *Chain-B* with s.
7. B uses the revealed preimage s to claim A's principal P_a and his premium p_b.
8. If A does not claim B's tokens, B can refund $P_b + p_a$ after t_1 expires.
9. If B does not claim A's tokens, A can refund $P_a + p_b$ after t_2 expires.

The GF-Swap handles the griefing case similarly to the XH-Swap; the difference is that it combines the premium with the principal and solves the griefing problem on premiums. Table 1 compares various atomic swap protocols along with 4-Swap. We now present the model and our assumptions for the 4-Swap protocol.

Table 1 Comparison of Atomic Swap Protocols. The column *Txns* indicates the total number of on-chain transactions required by the protocol in the worst-case scenario.

Protocol	Griefing Resistance	Bribery Safety	Rev. Bribery Safety	Txns
Tier-Nolan [15]	No	No	No	4
Hedged [21]	Yes	No	No	6
Grief-Free [13]	Yes	No	No	5
Rapidash [5]	No	Yes	Yes	6
4-Swap	Yes	Yes	No	4

4 System Model

We assume two UTXO-based blockchains that allow multi-signature transactions. All the interacting parties, including miners, are assumed to be rational, and they try their best to increase their incentives. The transactions in both blockchains first go into a pool of unconfirmed transactions known as the mempool, from which miners then take some transactions to create a block published in the blockchain. Transactions that utilize the same UTXO (conflicting transactions) can also be published in the mempool, from which only one eventually gets into the blockchain, and the rest become invalid. We also assume synchronous communication and the presence of a global clock. A transaction may be active or inactive based on whether it can be added to a block immediately or not. Thus, the transactions that can be added to a block after some timelock are called inactive transactions; otherwise, if they can be immediately added to a block, they are called active transactions. Both blockchains allow the publishing of both active and inactive transactions, which are stored in the mempool. Additionally, a collision-resistant hash function $\mathcal{H}$ converts inputs of any length to constant-length outputs. Also, all the parties can see the state of both the blockchains along with their mempools. The 4-Swap protocol assumes that each party must hold some tokens on the chain where they want to receive the tokens to safeguard against potential attacks, as done in all related work [21, 13]. Also, we assume that there is a significant difference in the principal and premium amount locked by the participating parties

to accommodate fluctuations in the price of tokens. The premium amount is significantly higher than the transaction fees in both chains, and the value gained by exchanging the tokens is less than losing any of the premium. Moreover, the protocol does not consider the reverse bribery attacks i.e when the miners incentivizes the involved parties to publish conflicting transactions.

4.1 UTXO Model

We adopt the notation for UTXO-based blockchains from [2], and use charts to visualize the process and clarify the transaction flow as shown in Figure 4. Transactions are represented as rounded rectangles. Transactions already confirmed on the blockchain are shown with double borders, while those not yet confirmed have single borders. Each transaction (rounded rectangle) includes one or more output boxes (with squared corners) showing the coins assigned to each output. Arrows indicate the conditions for spending them.

Typical conditions are abbreviated for simplicity. Most outputs require signatures from one or more parties, with public keys shown under the arrow. Additional conditions like timelocks are shown above the arrow. For example, a relative timelock, requiring that t rounds have passed since the transaction was published, is represented as $+t$ above the arrow. Based on the number of rounds since the blockchain's creation, absolute timelocks are labelled $> t$.

If the output's spending condition includes multiple options (a disjunction), the condition is written as $\phi = \phi_1 \vee \cdots \vee \phi_n$, where $n \in \mathbb{N}$. The charts depict this using a diamond shape for the output, and each condition ϕ_i is displayed on a separate arrow. It also uses $\vee$ notation on conditions to represent disjunction. In the case of conjunction, it is represented in the chart as $\phi = \phi_1 \wedge \cdots \wedge \phi_n$. Also, any transaction of the form tx_x^y depicts action y happening to funds of x. It does not depict who is publishing the transaction. Certain outputs can be spent by anyone labelled as ACS in Figure 4 without the need for signatures. The grey square represents who can spend the output.

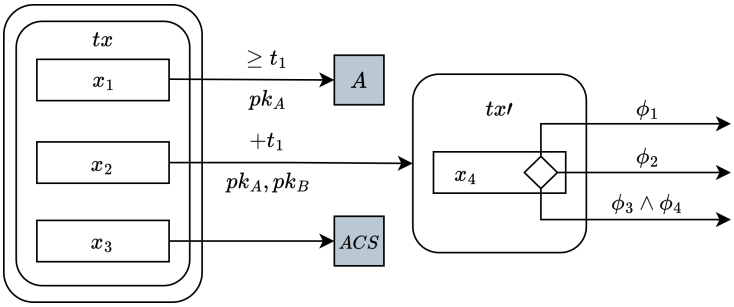


Figure 4 Transaction Schema Representation.

5 4-Swap Protocol

We present the development of the protocol through two versions, detailing the rationale behind each modification leading to the final 4-Swap protocol. The initial version (v1) focuses on facilitating grief-free atomic swaps that require only four on-chain transactions while making some assumptions. The next version then keeps the grief-free property and makes the protocol bribery safe within four steps. The notations used in the protocol are listed in Table 2 for two parties A and B on two chains *Chain-A* and *Chain-B*.

Table 2 Notation used in the protocol.

Notation	Description
$\mathcal{H}(s)$	Hash of some secret s
s_m	Main secret created by B
P_a	Principal amount A wants to exchange on $Chain\text{-}A$
p_a	Griefing premium of A on $Chain\text{-}B$
x_a	Bribery premium of A on $Chain\text{-}A$
P_b	Principal amount B wants to exchange on $Chain\text{-}B$
p_b	Griefing premium of B on $Chain\text{-}A$
x_b	Bribery premium of B on $Chain\text{-}B$
t_i	Timestamps in the chains
tx_x^y	Transaction that performs action y on tokens of x
s_{r1}	Refund secret created by A
s_{r2}	Refund secret created by B
s_{br}	Bribery secret created by A
s_e	Early execution secret created by A

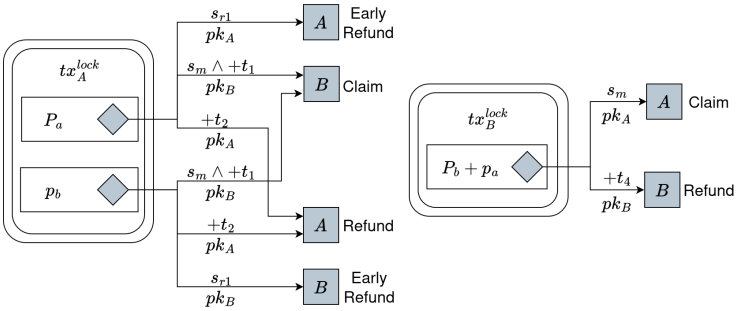


Figure 5 Transaction Schema for $4S\text{-}v1$.

5.1 4S-v1

In this initial version (v1), we relax the constraints typically associated with transaction processing by assuming that transactions are added to the blockchain immediately upon publication, with zero delay. This assumption is essential for considering conflicting transactions using the same UTXO. It ensures that the first active transaction put on the blockchain amongst many active or inactive conflicting transactions gets into it and can never be censored, eliminating the possibility of griefing. The transaction schema of $4S - v1$ is provided in Figure 5.

Atomic swaps use griefing premiums to prevent griefing attacks, but locking these premiums separately increases the number of on-chain transactions [13]. To reduce this, we combine the premium with the principal amount, limiting the total number of on-chain transactions to four. The combination is done crosswise: A's premium is locked with B's principal, and B's premium is locked with A's principal, ensuring the premium is returned during the claim transaction. The premium structure is designed such that the party claiming first has to pay a higher griefing premium, as it can cause grief to the counterparty by not claiming. In the first case of griefing, if a party quits after the counterparty has already locked, the counterparty can refund its principal and the quitter's premium, thereby penalizing the

quitter. However, in the second case, if a party quits before claiming (after both have locked), both parties can refund each other's premiums after the refund timelock. Thus, the first claimer has to lock a higher premium to disincentivize it from quitting the exchange.

We introduce cross-publishing of locking transactions and an option for an early refund for the party whose tokens get locked first. Cross-publishing means only B can publish A's locking transaction, and only A can publish B's. Cross-publishing helps to let the party know that their locking transaction is put on the chain by the counterparty, and they should do something based on it if they have the counterparty's locking transaction. Suppose B publishes A's locking transaction; if A has B's locking transaction, he should publish it; otherwise, he should immediately do an early refund on the locking transaction B put on the chain. If A makes an early refund of the principal amount, B should also be able to make the refund on the premium locked in the locking transaction; thus, another secret s_{r1}, which, when revealed during the early refund of A's principal amount, helps B to early refund his premium. s_{r1} just indicates that A is trying to refund his tokens. To ensure that A gets enough time to make an early refund, we introduce a new timelock t_1 as shown in Figure 6, which helps to restrict B from claiming until the timelock expires. Cross-publishing also eliminates the first case of griefing, where a party leaves after the counterparty locks the tokens. If A leaves after B publishes A's locking transaction, B can claim A's principal after t_1. Also, suppose A publishes B's locking transaction first by moving away from the protocol. In that case, B can either put A's locking transaction or wait until refund timelock to get back the principal and A's griefing premium. Thus, the protocol is grief-free, as shown in Lemma 1. We define the 4-Swap v1 protocol in three phases of setup, initiation and redeeming as in [19].

The protocol 4S-v1 describes an atomic swap between participants A and B, with B initiating the process. The principal amounts are denoted as P_a for A and P_b for B, while p_a and p_b represent the respective griefing premium amounts. t_1 is the timelock for the early refund on *Chain-A* and t_2, t_4 are the refund timelocks for *Chain-A* and *Chain-B* respectively. The following are the steps:

1. **Setup Phase**
   - **a.** B generates a secret s_m, finds $\mathcal{H}(s_m)$ and sends $\mathcal{H}(s_m)$, UTXO p_b to A.
   - **b.** A generates a secret s_{r1}, finds $\mathcal{H}(s_{r1})$, creates multi-sig transaction tx_A^{lock} that locks $P_a + p_b$ with $\mathcal{H}(s_m)$ and sets up two refund timelocks t_1(early refund) and t_2. s_{r1} is required if A wants to refund the locked tokens early. A then sends $tx_{lock,A}$ (only signed by A), $\mathcal{H}(s_r1)$ and UTXO p_a to B.
   - **c.** B creates a multi-sig transaction tx_B^{lock} that locks $P_b + p_a$ with $\mathcal{H}(s_m)$ and sets up refund timelock t_4. B then sends $tx_{lock,B}$ (only signed by B) to A. Nothing has been published in either of the chain until now.
2. **Initiation Phase**
   - **a.** B publishes tx_A^{lock} to *Chain-A*, which is immediately included in a block.
   - **b.** A then publishes tx_B^{lock} to *Chain-B* which is immediately included in a block.
3. **Redeeming Phase**
   - **a.** If A creates and publishes a transaction for early refunding P_a using s_{r1}, B can also publish a transaction for early refunding p_b using the revealed s_{r1}.
   - **b.** If A does not make an early refund, B waits for t_1 to expire and creates and publishes a transaction to claim P_a and p_b.
   - **c.** If B publishes tx_A^{claim}, then A also creates and publishes tx_B^{claim} to claim P_b and p_a
   - **d.** If B doesn't claim by t_2, A creates and publishes a refund transaction to get $P_a + p_b$.
   - **e.** If A doesn't claim by t_4, B creates and publishes a refund transaction to get $P_b + p_a$.

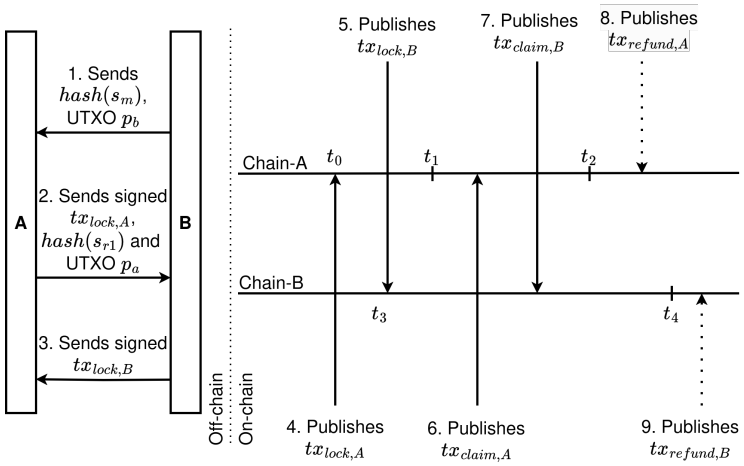


Figure 6 Transaction flow for $4S$-$v1$ and $4S$, with $4S$ differing slightly in off-chain interactions, as detailed in Section 5.2.

▶ **Lemma 1** (Griefing Lemma). *If any party abandons the protocol after tokens of the counterparty are locked on either of the two blockchains, the abandoning party incurs a penalty.*

Proof. Griefing can occur in two scenarios: first, if party B publishes party A's locking transaction, but A abandons the protocol, and second, if B abandons the protocol after both locking transactions are confirmed. In the first case, after the timelock t_1, B can submit a claim transaction because it knows the secret s, thereby retrieving the total amount $P_a + p_b$, where P_a and p_b are the tokens locked by A and B, respectively. Consequently, A incurs a loss of P_a. In the second case, if B abandons the protocol after both locking transactions are confirmed, A can refund its locked tokens $P_a + p_b$ after the timelock t_2, but A still incurs a penalty of p_a, the tokens it initially locked. To discourage B from abandoning the protocol, the value of p_b is set to be greater than p_a, ensuring that B faces a larger penalty in such scenarios. This mechanism ensures the protocol's resilience against griefing attacks. ◀

5.2 4-Swap Protocol Details

In this section, we remove the assumption of zero delay in adding transactions to the blockchain after publication. In practice, transactions first enter the mempool and are confirmed only when miners include them in a block. To address this delay and ensure protocol security, we incorporate the concept of mutual destruction from [19]. This approach disincentivizes participants from publishing conflicting transactions(bribery scenarios) in the mempool. However, we do not address reverse bribery attacks, as seen in HTLCs [20] because it would increase the total on-chain transactions.

Mutual destruction works by embedding secrets within conflicting transactions. If these secrets enter the mempool together, any participant (typically miners) can use them by creating a transaction claiming the entire locked amount, making both conflicting transactions invalid. This mechanism discourages parties from attempting bribery. To implement this in the 4S protocol, we introduce the following two secrets in addition to s_{r1} and the main secret s_m:

- s_{br}: A creates s_{br} for claiming B's tokens. It helps the protocol to know that A is claiming.
- s_{r2}: B creates s_{r2} for refunding its tokens. It helps the protocol to know that B is refunded.

Thus, whenever a pair of claim and refund transactions are published together in the blockchains, miner can create their slashing transaction to claim all the locked tokens. The transaction schema and all possible redeem paths for 4-Swap are shown in Figure 7 and Figure 8. This also covers the bribery case where an early refund transaction of A is censored by claim transaction of B (p_b works as bribery premium of B).

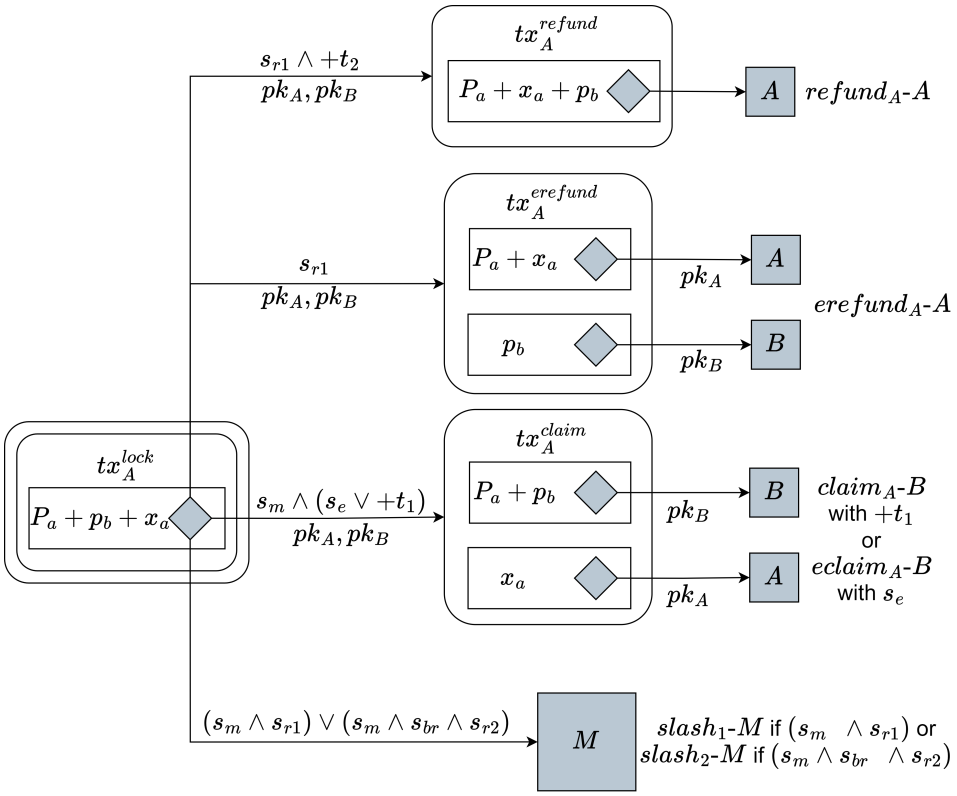


Figure 7 Transaction Schema of $4S$ for A.

Additionally, in the setup shown in Figure 6, even when both parties act honestly, B must wait for t_1 to expire before claiming the funds. We introduce another secret, s_e, generated by A for early execution. Once both locking transactions are confirmed on-chain, A can share s_e with B, allowing B to claim the funds without waiting for t_1 to expire.

However, griefing premiums alone are insufficient to address the bribery issue due to the following scenarios:

- After B successfully claims A's tokens in *Chain-A*, B may attempt to bribe miners by publishing conflicting refund transactions, as B no longer has anything at risk.
- B's claim transaction remains in the mempool, while A observes the corresponding secret from the mempool of *Chain-A*. If A's claim transaction is confirmed on *Chain-B* first, A may attempt to bribe miners by submitting a conflicting refund transaction since A similarly has nothing to lose.

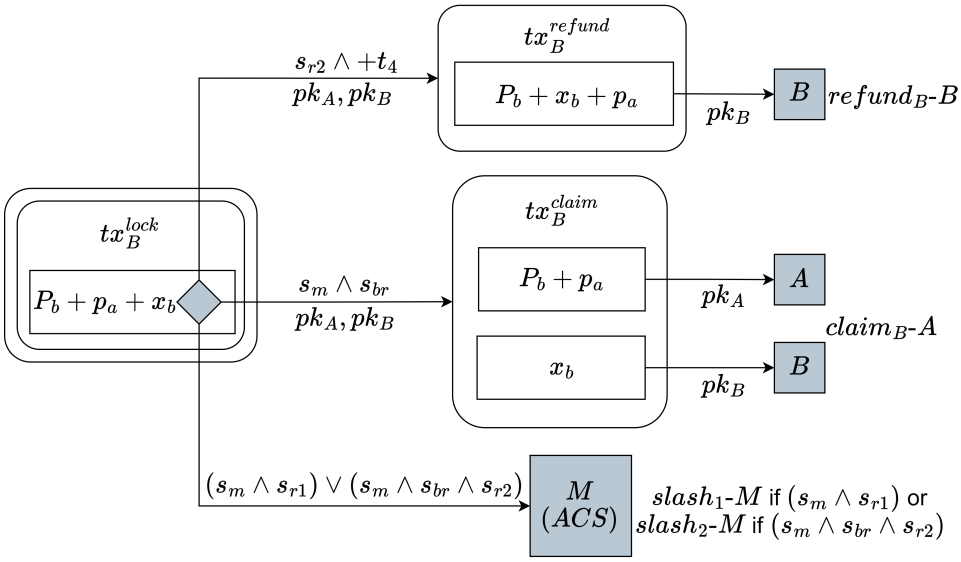


Figure 8 Transaction Schema of $4S$ for B.

The protocol introduces two bribery premiums to mitigate the abovementioned risks: x_a for party A and x_b for party B. These premiums are not cross-locked, meaning each party must contribute an additional amount alongside their principal to be a bribery premium. Moreover, we incorporate static claim transactions in the swap. In the setup phase, both parties jointly create their claim transactions, which cannot be altered during the protocol's runtime. These static claim transactions facilitate the exchange of principal amounts while returning the bribery premiums within the same transaction. This approach addresses the bribery risks in the abovementioned scenarios. If a party attempts bribery, they risk losing their bribery premium if the counterparty cannot claim it due to the bribery attempt. We also create static early refund and refund transactions for party A on $Chain$-A and static refunds for B on $Chain$-B to reduce the overhead of multiple redeem paths. $slash_1$ and $slash_2$ in Figure 7 and Figure 8 represent the redeem paths available to miners on individual locking transactions in case of bribery. The protocol flow is the same as Figure 6, but just in off-chain interactions, the claim, refund and early refund transactions are created and signed before signing the locking transactions. When a party(miner) uses s_m, s_{r1} to slash, it comes under $slash_1$, otherwise if it slashes by s_m, s_{br}, s_{r2} it comes under $slash_2$. We now present the $4S$ protocol in three phases.

Protocol $4S$ defines an atomic swap between participants A and B, initiated by B. Principal amounts are P_a for A and P_b for B, with griefing premiums p_a and p_b, and bribery premiums x_a and x_b, respectively. Figure 7 and Figure 8 shows the detailed construction of the two locking transactions and how the subsequent transactions will be created. The following are the required steps:

1. **Setup Phase**
   - **a.** B generates secrets s_m, s_{r2}, finds $\mathcal{H}(s_m), \mathcal{H}(s_{r2})$ and sends $\mathcal{H}(s_m)$, UTXO p_b to A.
   - **b.** A generates secrets s_{r1}, s_e, s_{br}, finds $\mathcal{H}(s_{r1})$, $\mathcal{H}(s_e)$, $\mathcal{H}(s_{br})$, creates multi-sig transaction tx_A^{lock} that locks $P_a + p_b$ with all hashes with different conditions and sets up two refund timelocks t_1(early refund) and t_2. s_{r1} is required if A wants to refund the locked tokens early. A also creates multi-signature transactions tx_A^{claim}, $tx_A^{erefund}$,

tx_A^{refund} using tx_A^{lock} and signs it. The tx_A^{claim}, $tx_A^{erefund}$ and tx_A^{refund} are completely signed before signing tx_A^{lock}. A then sends tx_A^{lock} (only signed by A), tx_A^{claim}, $\mathcal{H}(s_{r1})$, $\mathcal{H}(s_e)$, $\mathcal{H}(s_{br})$ and UTXO p_a to B.

c. B creates a multi-sig transaction tx_B^{lock} that locks $P_b + p_a$ with all hashes with different conditions and sets up refund timelock t_4. B also creates a multi-signature transaction tx_B^{claim} and tx_B^{refund} using tx_B^{lock} and signs it. B then sends the tx_B^{lock} and tx_B^{claim} to A. The tx_B^{claim} and tx_B^{refund} are completely signed before signing tx_B^{lock}. B then sends $tx_{lock,B}$ (signed by B) to A. Nothing has been published in either chain until now.

2. **Initiation Phase**
   a. B publishes tx_A^{lock} to *Chain-A*.
   b. A then publishes tx_B^{lock} to *Chain-B*.
3. **Redeeming Phase**
   a. If A and publishes $tx_A^{erefund}$ for early refunding $P_a + x_a$ using s_{r1}, B also gets back p_b.
   b. If $tx_A^{erefund}$ is not published, B waits for t_1 to expire and publishes tx_A^{claim} to claim P_a and p_b. This also refunds x_a to A.
   c. B can publish without waiting if A releases the early execution secret s_e to B.
   d. If B publishes tx_A^{claim}, then A also creates and publishes tx_B^{claim} to claim P_b and p_a. this also refunds x_b to B.
   e. If B does not claim before t_2, A creates and publishes tx_A^{refund} to get P_a, p_b and x_a.
   f. If A does not claim before t_4, B creates and publishes tx_B^{refund} to get P_b, p_a and x_b.

Reducing the on-chain transactions to four helps in reducing the overall time taken by the exchange in both the worst case as well as for the best case. For the best case, the parties follow the protocol and the early execution secret s_e is also shared. The protocol thus just requires four transactions, each followed by some confirmation delay. For the worst case, if party B leaves after both the locking transactions have been published, both parties have to wait until the refund timelocks expire in both the chains to refund their tokens. The refund timelocks entered will be smaller than the refund timelocks in the earlier works [21, 13] as it requires fewer on-chain transactions.

The security analysis of the protocol is detailed in the extended version of the paper [17]. We now present a game-theoretical analysis of the 4-Swap protocol to ensure that the protocol remains in the equilibrium state in the presence of rational parties and miners

6 Game Theoretical Analysis

The protocol is modelled as an Extensive Form Perfect Information Game, where all parties know the latest states of blockchains and their mempools [1]. The game consists of three phases: Setup, Claim, and Refund. The Setup phase covers the protocol's setup and locking steps, while the Claim and Refund phases focus on claiming and refunding locked funds.

The game involves three rational players: A, B, and the miner M. The miner M is initially assumed to be the strongest and only miner on both blockchains, and is responsible for mining all blocks. Since M controls the blockchain, it can always accept bribes to censor transactions. However, we demonstrate that the protocol effectively incentivizes honest behaviour, ensuring that bribery is not rational for any party.

The game progresses in rounds. In each round, A and B choose their actions, after which the miner M confirms transactions and advances the game to the next round. The complete game tree is constructed by combining the trees for the Base, Claim, and Refund trees in Figure 9, Figure 10 and Figure 11.

Initially, we analyze the protocol under the assumption of a single powerful miner, M. However, we assume this miner M does not intentionally delay the confirmation of transactions without bribery. We later show that this is true in the case of a multi-miner setup because of opportunity cost. We then generalize the model to a multi-miner setup across both chains. The analysis shows that the actions and utilities of A and B remain consistent even in the multi-miner setting, illustrating the protocol's robustness and ability to maintain incentives for honest participation under varying conditions.

▶ **Definition 2** (Perfect Information Extensive Form Game). *A game with perfect information in extensive form can be described as a tree structure and is formally defined by the tuple* (N, H, P, A_i, u_i) *where* $i \in N$*:*

- N*: A finite set of* n *players, labeled as* $N = \{1, 2, \ldots, n\}$*. Each decision point in the game that is not terminal is associated with a specific player* $I \in N$*, responsible for choosing that stage.*
- H*: The collection of all possible action sequences, called histories, where each sequence* h *leads to a specific node within the game tree. A subset* $Z \subseteq H$ *represents terminal histories, which correspond to the endpoints of every possible playthrough, also known as the tree's leaf nodes.*
- P*: A mapping called the player function, which determines the player assigned to decide each non-terminal history* $h \in H \setminus Z$*.*
- A_i*: A function that specifies the set of actions available to a player* i *at a given history* h*, denoted as* $A_i(h)$*. This set represents the choices available to player* i *at that point in the game. Each branch extending from a node in the tree corresponds to one of these actions.*
- u_i*: A function representing the payoffs for each player* i*, defined as* $u_i : Z \to \mathbb{R}$*. This function assigns a real number to each terminal history* $z \in Z$*, which reflects the utility that player* i *receives if the game ends at that particular terminal state.*

The parties A, B, M will resort to different actions at different game histories based on the utilities they will get from the actions. A strategy for a party is the sequence of actions it will take as the game proceeds based on the actions already taken by all the parties. The set of all such strategies of all the players is known as the strategy profile. We now formally present the definition of strategy profile.

▶ **Definition 3** (Strategy Profile). *In an extensive form game with perfect information, a strategy profile specifies the action* $a \in A_i(h)$ *chosen by each player* $i \in N$ *at every history* h *when required to act. Specifically, for each player* $i \in N$*, a strategy* s_i *is a function mapping the set of histories* $H_i = \{h \in H : P(h) = i\}$ *to the set of actions* A_i*. This mapping satisfies* $s_i(h) \in A_i(h)$ *for all* $h \in H_i$*. A strategy profile is a collection of strategies for all players,* $s = (s_1, s_2, \ldots, s_n)$*.*

Since the game is a perfect information game, all the parties know the best action for the other parties. Since parties are rational, they will try to increase their utility in each subgame of the main game. Subgame Perfect Nash Equilibrium gives the path of the game from which the parties will not deviate, as it gives them the best utility. We now define the Subgame-Perfect Nash Equilibrium.

▶ **Definition 4** (Subgame Perfect Nash Equlibrium). *A strategy profile* $s^* = (s_1^*, s_2^*, \ldots, s_n^*)$ *qualifies as a Subgame Perfect Nash Equilibrium (SPNE) if, in every subgame* G' *of the game* G*, each player's strategy* s_i^* *is the optimal response to the strategies of all other players.*

More specifically, let H' *represent the set of all histories within the subgame* G'*. For each player* i*, the strategy* s_i^* *is deemed optimal in* G' *if the following condition holds:* $u_i(s_i^*, s_{-i}^*; h) \geq u_i(s_i, s_{-i}^*; h)$*, for all strategies* s_i *available to player* i *in* G'*, and for every*

history $h \in H'$. Here, s^*_{-i} represents the strategies of all players other than i in the SPNE, and $u_i(s_i, s^*_{-i}; h)$ denotes player i's payoff when the strategy profile (s_i, s^*_{-i}) is followed in the subgame starting at history h. A strategy profile is an SPNE if it ensures a Nash Equilibrium in every subgame.

We first define the slashing lemma and then prove the game-theoretic safety of 4-Swap.

▶ **Lemma 5** (Slashing Lemma)**.** *Slashing is an action available to the miner M, which can take away all the locked tokens of the parties A and B. For any $h \in H_M$ of subgame G' of our game where M is miner, if $\exists\ a = slash \in A_M(h)$, then $u_M(h, slash) > u_M(h, !slash)$.*

Proof. The miner only gets the option to slash the locked tokens if there is a bribery scenario of censoring a current transaction by a conflicting transaction that becomes valid after a certain timelock. The miners will only censor the transaction if they get enough bribes from the bribing party to censor it. Assuming the lemma does not hold, at least one other action exists for the miner M, which gives it a better utility than slashing. Therefore, the bribe amount should be more than the tokens locked in the chains. Assuming the parties are rational, such a bribe will not be published. Thus, no action can provide a better utility than slashing, and the assumption that the lemma does not hold is false. Hence, rational miners will always choose to slash both parties' locked tokens. ◀

6.1 Game Trees

The game proceeds in rounds with parties putting all their available choices before a miner comes to confirm some transaction, and then the next round starts. The game is divided into three parts for better representation. The base tree is the starting point of the tree and starts with A sharing the tx^A_{lock} with B or not. This tree gives information about different scenarios of early refunds and reflects that the best strategy is not to deviate from the honest path. The claim tree assumes both parties have locked their tokens in their chains. The claim tree starts with B, choosing whether to publish the claim transaction or not. The refund tree focuses on parties wanting to return their tokens without claiming. In any of the trees, wherever a miner gets the option to slash, it chooses that, and thus, all the other available options are not shown in the tree to reduce its size. The nodes in the tree represent the party that takes the action, and the action taken by that party is represented by the arrow that takes it to another node, which takes the next action. After all the transitions are finished, the leaf represents the utilities of the parties A, B and M. In Figure 9, Figure 10 and Figure 11, the utilities of all the parties in different game paths are represented at the leaf nodes. The utilities in *Chain-A* and *Chain-B* are represented with a superscript of 1 and 2, respectively, and f represents the transaction fees the miner gets on confirming some transaction. The claim tree and refund tree are together called as redeem tree. The game trees do not directly represent the opportunity cost, but if two paths give the same utility, the path with the least delay is preferred because of the lesser loss of opportunity cost. Ideally, B would prefer P_a over P_b and vice-versa for A as they are interested in the exchange. In case of refund this interest of parties changes and thus the difference in griefing premium $(p_b - p_a)$ helps in dis-incentivizing the parties from leaving.

6.1.1 Base Tree

The base tree begins with A deciding whether to send her locking transaction tx^{lock}_A to B. If not, the protocol ends without affecting either party's tokens. If sent, B can either publish tx^{lock}_A or respond with tx^{lock}_B. If B only publishes tx^{lock}_A, miners (M) confirm it and earn a

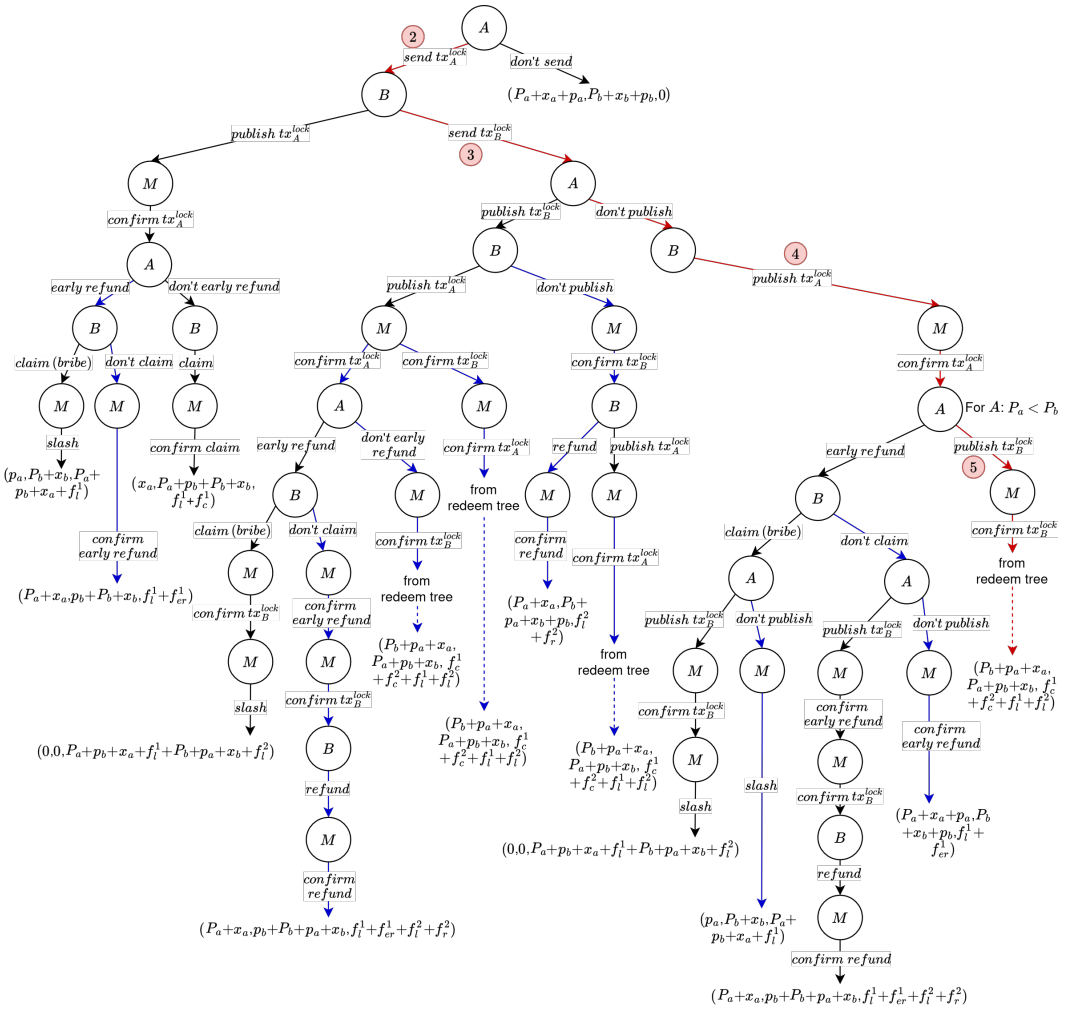


Figure 9 SPNE – Base Tree: Blue lines show sub-game optimal strategies; red lines show main game optimal strategies. Red path numbers correspond to transaction steps in Figure 6.

fee f_l^1, and A then chooses whether to publish an early refund. If A does so and B bribes miners to censor it, M slashes all locked tokens. Otherwise, A is refunded and M earns f_{er}^1. If A doesn't refund early, B can claim the funds after a timelock, gaining P_a, while A loses.

If B sends tx_B^{lock}, A then decides whether to publish it. If she does and B withholds tx_A^{lock}, M confirms tx_B^{lock} and gains f_l^2. B can now either refund, gaining both his funds and A's griefing premium, or proceed with the exchange. By backward induction, refunding yields higher utility. If both publish their locking transactions, miners confirm one first, and parties act accordingly. In any bribery attempt, miners benefit from slashing, and both parties incur losses.

Transaction fees f_{er}^1, f_r^1 are for early and regular refunds on *Chain-A*, and f_l^2, f_r^2 are for locking and refunding on *Chain-B*. The redeem tree shows equilibrium utilities after locking, further explored in the claim tree. If parties exchange locking transactions and A publishes first, backward induction shows B prefers refunding to gain the griefing premium.

Finally, if both parties exchange transactions and B publishes tx_A^{lock} first (confirmed with fee f_l^2), A can either early refund, publish tx_B^{lock} or do nothing(not shown in Figure 9). Since early refunding results in fee loss, backward induction favours continuing the exchange by publishing tx_B^{lock}. The path where A does not do anything will lead to B claiming the locked assets as it knows the main secret.

6.1.2 Claim Tree

The claim tree represents the subgame where both parties have published and locked their tokens on the chain. It starts with B having the option to publish a claim transaction or leave the exchange. Leaving the exchange means eventual refining by both parties, which we will discuss in the refund tree. If B publishes a claim transaction revealing the secret s_m, A can publish the claim transaction. In subsequent sub-games, whenever a party A or B gets an option to bribe by submitting the refund transaction on the chain, the best strategy for the miners is to slash; thus, all rational parties will refrain from doing so. Thus, the best strategy is to claim each other's tokens without refunding after publishing the claim transactions. f_c^1 represents the transaction fees of claim transaction on *Chain-A* and f_c^2, f_r^2 represents claim and refund transaction fees on *Chain-B*.

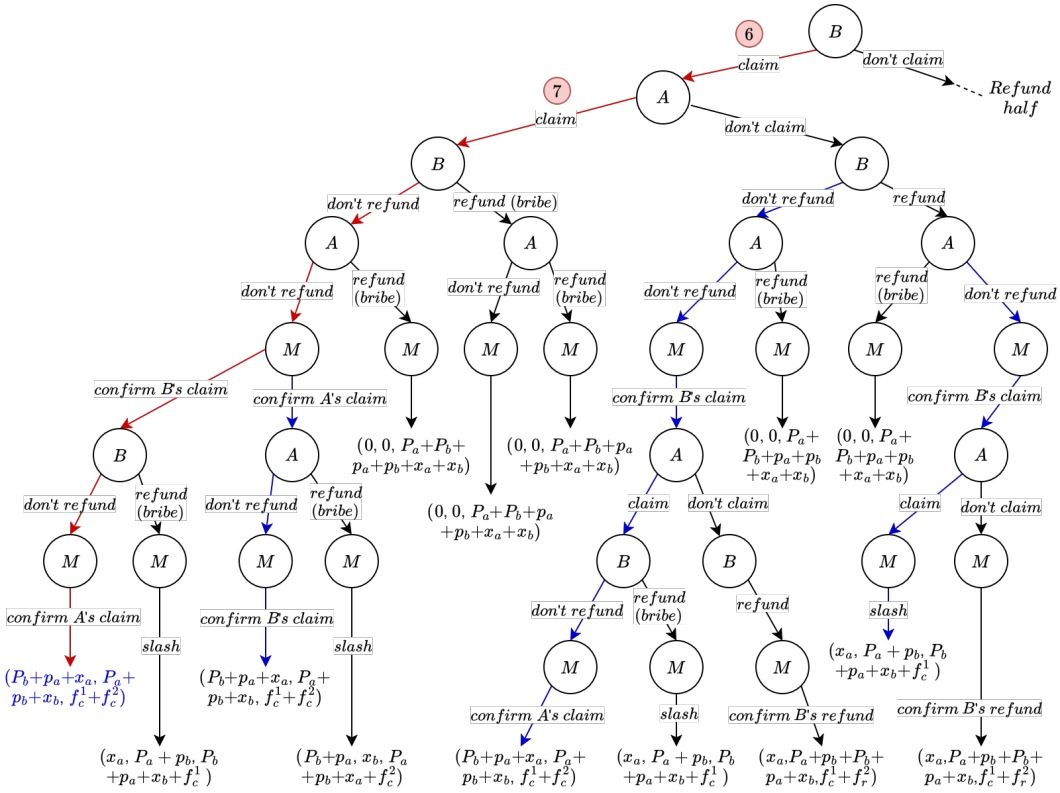


Figure 10 SPNE – Claim Half.

6.1.3 Refund Tree

When B does not publish a claim transaction, A either publishes a refund transaction or does not. If she does not publish the refund transaction, B will publish his refund transaction, which is eventually confirmed by miners by gaining f_r^2. After this, B can still try to claim

A's transaction by publishing a claim, which can be contested by A by publishing a refund, but the miners' best strategy would be to slash. If A publishes a refund after B does not publish a claim transaction, the best strategy for B is to also submit a refund transaction and avoid slashing actions for the miner. Thus, using backward induction, we can see from the refund tree that the best strategy is when both parties publish refund transactions without publishing their claim transactions.

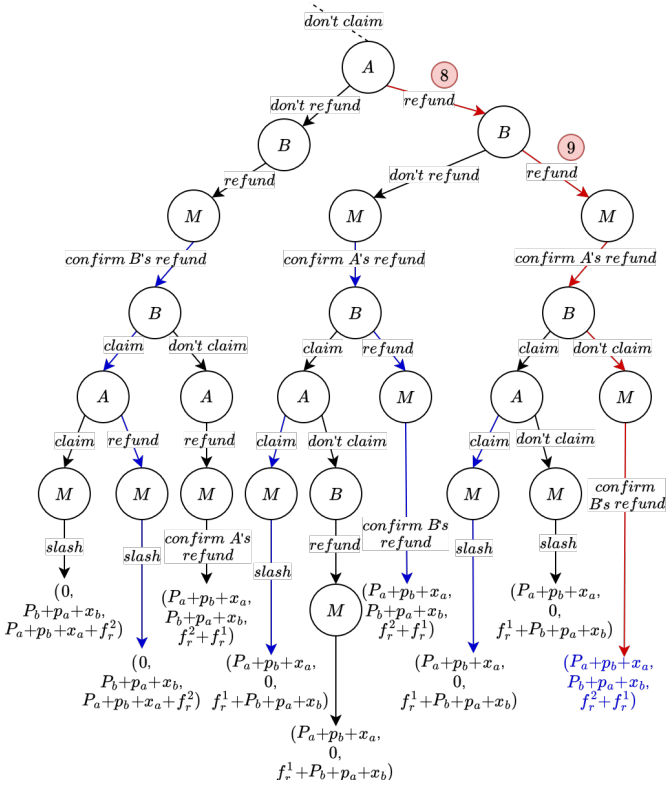


Figure 11 SPNE – Refund Half.

To prove that our protocol is game-theoretically safe, we need to show that the protocol holds the Subgame Perfect Nash Equilibrium at the honest(expected) path in the game tree.

▶ **Theorem 6** (SPNE). *The Subgame Perfect Nash Equilibrium for our game is the strategy profile* $s^*(A,B,M) = (send\ tx_A^{lock},\ don't\ publish,\ publish\ tx_B^{lock})$, $(send\ tx_B^{lock},\ publish\ tx_A^{lock},\ claim,\ don't\ refund,\ don't\ refund)$, $((confirm\ tx_A^{lock},\ confirm\ tx_B^{lock},\ confirm\ B's\ claim,\ confirm\ A's\ claim)\ or\ (\ confirm\ tx_A^{lock},\ confirm\ tx_B^{lock},\ confirm\ A's\ claim,\ confirm\ B's\ claim))$.

Proof. Lemma 5 ensures parties will never try to bribe the miners because of slashing. We apply backward induction at every subgame of the setup, claim, and refund phases of the game tree until we get the SPNE for the entire game. Figure 9, Figure 10 and Figure 11 show the SPNE of the entire 4-Swap game. Thus, parties can never get more utility than the prescribed(honest) strategy. ◀

Theorem 6 shows that the honest course of action is the SPNE of our game. We now show that the strongest miner M can be generalized to a multi-miner setup in both chains while ensuring the protocol's safety.

▶ **Lemma 7** (Miner's Dilemma). *In the case of a multi-miner setup, a miner does not wait for the parties to publish conflicting transactions for slashing opportunities.*

Proof. Lemma 5 proves that slashing gives the maximum utility to the miners. However, in the presence of rational parties, parties know that slashing is the best action for the miners, and thus, they will refrain from publishing conflicting transactions. Thus, the miner will lose the opportunity cost to some other miner by waiting for the conflicting transaction and hence does not wait for the same. ◀

▶ **Theorem 8.** *In the 4-Swap protocol, rational parties A and B maximize their utilities by following honest actions, even in a multi-miner setup.*

Proof. Replacing a single all-powerful miner with arbitrary miners in a multi-miner setup yields identical utilities for parties A and B at all game tree leaf nodes. Lemma 5 shows that the slashing mechanism deters any miner, regardless of hashing power, from delaying transactions due to bribery. Rational miners thus enforce the slashing mechanism when applicable. In non-bribery scenarios, miners can increase their rewards by confirming the only valid transaction (on any particular chain). Thus, arbitrary miners make the same decisions as an all-powerful miner. While the rewards in the game tree may be distributed among multiple miners in a multi-miner setup, this distribution does not affect the utilities of A and B. Consequently, the rational strategies for A and B remain aligned with honest behaviour, preserving the protocol's game-theoretic safety in a multi-miner environment. ◀

7 Discussion

To make the 4-Swap protocol reverse bribery safe along with griefing safety, we can leverage the idea of splitting the refund transaction into two steps, such that whenever a refund transaction is published, there is sufficient time for miners to claim a portion and burn the remaining tokens [20, 5]. Splitting the transaction in two steps helps in providing a punishing window, thereby solving the reverse bribery. In 4-Swap there are three bribery cases: early refund of A getting censored by claim of B, claim of B getting censored by refund of A and claim of A getting censored by refund of B. To handle all these cases, all the transactions that cause conflicts have to be split up. Thus, the claim of B and the two main refund transactions also have to be split up into two transactions. Thus for the worst case where both parties refund, we will have six transactions and in best case where parties exchange their tokens, we will have five transactions. This still leads to an open question whether we can extend this construction to be reverse bribery safe in just four transactions.

8 Conclusion

We propose 4-Swap, the first atomic swap protocol to achieve both grief-free and bribery-resistant atomic swaps using only four steps. To our knowledge, this is the first work that combines principals and premiums in both chains while matching the number of transactions in TN-Swap, leading to faster execution of the protocol compared to previous grief-free atomic swaps. A game-theoretical analysis of the protocol shows that the honest execution path constitutes a stable equilibrium, and thus satisfies the goals of being grief-free and safe against bribery attacks.

References

1 Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, and Subhra Mazumdar. Securing lightning channels against rational miners. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 393–407, 2024. doi:10.1145/3658644.3670373.

2 Lukas Aumayr, Matteo Maffei, Oğuzhan Ersoy, Andreas Erwig, Sebastian Faust, Siavash Riahi, Kristina Hostáková, and Pedro Moreno-Sanchez. Bitcoin-compatible virtual channels. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 901–918, 2021. doi:10.1109/SP40001.2021.00097.

3 Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. A survey on blockchain interoperability: Past, present, and future trends. *ACM Comput. Surv.*, 54(8):168:1–168:41, 2022. doi:10.1145/3471140.

4 Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007. doi:10.1007/978-3-540-70936-7_4.

5 Hao Chung, Elisaweta Masserova, Elaine Shi, and Sri AravindaKrishnan Thyagarajan. Rapidash: Atomic swaps secure under user-miner collusion. *Cryptology ePrint Archive*, 2022.

6 Yuanjun Ding and Weili Chen. Probing the mystery of cryptocurrency exchange: The case study based on mt.gox. In *International Conference on Service Science, ICSS 2022, Zhuhai, China, May 13-15, 2022*, pages 297–304. IEEE, 2022. doi:10.1109/ICSS55994.2022.00053.

7 Shange Fu, Qin Wang, Jiangshan Yu, and Shiping Chen. FTX collapse: A ponzi story. In Aleksander Essex, Shin'ichiro Matsuo, Oksana Kulyk, Lewis Gudgeon, Ariah Klages-Mundt, Daniel Perez, Sam Werner, Andrea Bracciali, and Geoff Goodell, editors, *Financial Cryptography and Data Security. FC 2023 International Workshops - Voting, CoDecFin, DeFi, WTSC, Bol, Brač, Croatia, May 5, 2023, Revised Selected Papers*, volume 13953 of *Lecture Notes in Computer Science*, pages 208–215. Springer, 2023. doi:10.1007/978-3-031-48806-1_14.

8 Maurice Herlihy. Atomic cross-chain swaps. In Calvin Newport and Idit Keidar, editors, *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, pages 245–254. ACM, 2018. URL: https://dl.acm.org/citation.cfm?id=3212736.

9 Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 245–254, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3212734.3212736.

10 Philipp Hoenisch, Subhra Mazumdar, Pedro Moreno-Sanchez, and Sushmita Ruj. Lightswap: An atomic swap does not require timeouts at both blockchains. In Joaquín García-Alfaro, Guillermo Navarro-Arribas, and Nicola Dragoni, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2022 International Workshops, DPM 2022 and CBT 2022, Copenhagen, Denmark, September 26-30, 2022, Revised Selected Papers*, volume 13619 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2022. doi:10.1007/978-3-031-25734-6_14.

11 Subhra Mazumdar. Towards faster settlement in htlc-based cross-chain atomic swaps. In *4th IEEE International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications, TPS-ISA 2022, Atlanta, GA, USA, December 14-17, 2022*, pages 295–304. IEEE, 2022. doi:10.1109/TPS-ISA56441.2022.00043.

12 Tejaswi Nadahalli, Majid Khabbazian, and Roger Wattenhofer. Timelocked bribing. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I*, pages 53–72, Berlin, Heidelberg, 2021. Springer-Verlag. doi:10.1007/978-3-662-64322-8_3.

13 Tejaswi Nadahalli, Majid Khabbazian, and Roger Wattenhofer. Grief-free atomic swaps. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2022, Shanghai, China, May 2-5, 2022*, pages 1–9. IEEE, 2022. doi:10.1109/ICBC54727.2022.9805490.

14 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, 2008.

15 Tier Nolan. Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic=193281.msg2224949, 2013. Accessed: 2024-05-20.

16 Sandeep Rao and Corbet Shaen. Mt. gox–the fall of a giant. *Understanding Cryptocurrency Fraud, edited by Shaen Corbet*, pages 71–82, 2022.

17 Kirti Singh, Vinay J. Ribeiro, and Susmita Mandal. 4-swap: Achieving grief-free and bribery-safe atomic swaps using four transactions, 2025. doi:10.48550/arXiv.2508.04641.

18 Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. Universal atomic swaps. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 1299–1316. IEEE, 2022. doi:10.1109/SP46214.2022.9833731.

19 Itay Tsabary, Matan Yechieli, Alex Manuskin, and Ittay Eyal. MAD-HTLC: because HTLC is crazy-cheap to attack. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1230–1248. IEEE, 2021. doi:10.1109/SP40001.2021.00080.

20 Sarisht Wadhwa, Jannis Stoeter, Fan Zhang, and Kartik Nayak. He-htlc: Revisiting incentives in HTLC. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023. URL: https://www.ndss-symposium.org/ndss-paper/he-htlc-revisiting-incentives-in-htlc/.

21 Yingjie Xue and Maurice Herlihy. Hedging against sore loser attacks in cross-chain transactions. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 155–164. ACM, 2021. doi:10.1145/3465084.3467904.

22 Ryan Zarick, Bryan Pellegrino, and Caleb Banister. Layerzero: Trustless omnichain interoperability protocol. *CoRR*, abs/2110.13871, 2021. arXiv:2110.13871.

Cuttlefish: A Fair, Predictable Execution Environment for Cloud-Hosted Financial Exchanges

Liangcheng Yu
Microsoft Research, Redmond, WA, USA
University of Pennsylvania, Philadelphia, PA, USA

Prateesh Goyal
Microsoft Reserach, Redmond, WA, USA

Ilias Marinos
Nvidia, Redmond, WA, USA

Vincent Liu
University of Pennsylvania, Philadelphia, PA, USA

Abstract

Recent years have seen a rising interest in cloud-hosted financial exchanges. While the public cloud platforms promise a cost-effective and more accessible option to traders, unfortunately, achieving fairness in cloud environments is challenging due to non-deterministic network latencies and execution times.

This work presents Cuttlefish, a fair-by-design cloud execution environment for algorithmic trading. The idea behind Cuttlefish is the efficient and robust mapping of real operations to a novel formulation of "virtual time". With it, Cuttlefish abstracts out the variances of the underlying network communication and computation hardware. Our implementation and evaluation not only validate the practicality of Cuttlefish, but also show its operational efficiency on public cloud platforms.

2012 ACM Subject Classification Networks → Cloud computing; Networks → Network dynamics; Applied computing → Online auctions

Keywords and phrases Cloud-hosted exchanges, Financial exchanges, Computation and communication variances, Virtual time overlay

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.33

Related Version *Full Version*: `https://www.microsoft.com/en-us/research/publication/cuttlefish` [57]

1 Introduction

Low-latency algorithmic trading – a subset of the broader securities trading taxonomy – is pivotal to the efficiency of modern financial markets, promoting accurate and timely pricing of securities, higher liquidity, and lower trade costs for all investors [31]. The goal of low-latency algorithmic trading is to create an ecosystem within each exchange where, based on incoming market data, traders can issue buy and sell orders as quickly as possible to take advantage of ephemeral market-making and arbitrage opportunities [56]. A sizable fraction of activity in today's exchanges is the result of this class of trades [2, 24].

In recent years, exchanges like NASDAQ, CME, LSE, and B3, alongside cloud providers like Microsoft, Amazon, and Google, have expressed growing interest in exploring the viability of hosting this type of trading in the cloud [22, 48, 41, 40]. Their interest is rooted in a variety of factors including better scaling, fewer outages, improved cost savings, and a potentially

© Liangcheng Yu, Prateesh Goyal, Ilias Marinos, and Vincent Liu;
licensed under Creative Commons License CC-BY 4.0

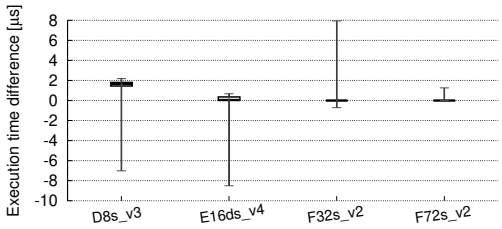


Figure 1 Execution time differences (candlesticks showing $p0.1$, $Q1$, $Q2$, $Q3$, and $p99.9$, with some quantiles visually converging to a single bar) measured with `rdtsc` for 1M parallel invocations of two MPs running on separate VM instances within the same region, despite an identical program (performing a moving average crossover strategy, with each invocation taking $\approx 14\mu s$ on D8s_v3), input market data stream, performance optimizations, and VM type.

broader customer base as, in this model, any Market Participant (MP)[1] can rent a machine in the same region as the Central Exchange Server (CES) and participate, bypassing the logistical hurdles involved in installing and maintaining on-premise hardware [24, 17, 19, 18]. Smaller and newer exchanges (e.g., for cryptocurrency) are particularly interested, as the cloud can also lower their barriers to entry by eliminating the need for custom infrastructure.

Unfortunately, in addition to the above advantages, cloud-hosted exchanges also face significant challenges in ensuring fairness for MPs, a primary requirement of exchange services [47, 24, 17, 18]. Recent work has noted these exchanges' issues with unpredictable network latencies [24, 37, 19], but the sources of unfairness extend beyond the network and include practicalities like noisy neighbors, thermal fluctuations, or (un)scheduled maintenance [38, 52, 54]. For example, Figure 1 shows the differences between local execution time of two identical trading programs running on parallel instances in the same Azure region. The experiment is for a simple program, single-tenancy (in the case of the `F72_v2` instance type), and strict scheduling policies (core pinning, highest kernel scheduling priority, etc.). It further omits network variability and clock drift effects – two important and difficult-to-control sources of unfairness. Even so, we can observe significant variability and bias between the two executions.

Crucially, the magnitude of the variability is immaterial – *any* difference, no matter how small, may alter the order in which trades are processed by the CES. We note that some of these effects also exist in traditional on-premise exchanges; however, cloud-hosting (besides increasing the sources and magnitude of variability/bias) presents a qualitative change in how these effects must be handled. In an on-premise deployment, customers are in full control of their infrastructure – when a machine is slow, it is the customer's fault. In the cloud, infrastructure can be slow through no fault of the customer, and the responsibility falls upon the cloud provider.

In this paper, we tackle an ambitious goal: a cloud execution environment where the outcome of races between different MPs' buys/sells is based *only* on the design of the MPs rather than (un)lucky performance fluctuations of their underlying cloud infrastructure. Our approach is similarly ambitious: to change the execution model from one where users have unfettered access to (virtualized) hardware to one where users provide bytecode-level programs (closer to a Functions-as-a-Service interface) and cloud providers control their rate of execution to ensure fairness.

[1] In low-latency algorithmic trading, MP also refers to the computer program executing the trading algorithm, terms we use interchangeably.

The resulting system, Cuttlefish[2], is an execution environment for a cloud-hosted exchange that ensures fair, predictable end-to-end execution. Cuttlefish is the first to address execution variation/bias in cloud-hosted exchanges, but its focus on end-to-end guarantees means that it also handles deficiencies in the communication guarantees of prior work that specialize in communication [17, 24, 19]. Further, not only is Cuttlefish able to guarantee these strong properties, but it is able to do so while offering low latency and high throughput.

Guaranteeing this level of fairness is fundamentally challenging as, in the end, simultaneous data delivery and synchronous execution is a classic (and under some assumptions impossible) challenge in distributed systems [20, 13, 55, 24]. Moreover, modern hardware performance is increasingly unpredictable, complex to reason about, and difficult to verify [38, 52, 54, 60, 12, 4, 32, 3, 9].

Cuttlefish achieves the above using an efficient mapping of real operations to "virtual time" from a platform-agnostic intermediate representation (IR). This *indirection* through virtual time allows us to quantify computation and, critically, to control its advancement deterministically through the rate-limiting of MP operations – a level of control that is not possible with real time. This approach enables deterministic and fair operations in both simultaneous market data release and MP exeuction processes and guarantees fairness, regardless of the varying latencies in the communication of market data and trade responses or variations in execution platforms.

This concept of virtual time mirrors that of other applications such as co-simulation [4, 32, 3, 9], which coordinates virtual time for concurrent emulation and simulation processes. Cuttlefish takes a step further by extending the concept to real-time cloud systems and developing an end-to-end trading platform. Cuttlefish's architecture is, thus, a combination of (1) a platform-agnostic Intermediate Representation (IR) instantiating virtual time per virtual machine instruction cycle count, along with its expressive programming interface and lightweight instrumentation for virtual cycle tracking, (2) a runtime execution environment optimized for co-located MPs, and (3) a protocol to control inflight virtual cycles and handle variations in the underlying network or compute.

While Cuttlefish represents an extreme point in the design space, our prototype demonstrates the feasibility and efficiency of its design, deployable to commercial cloud virtual machines. This paper makes the following contributions:

- We propose Cuttlefish, the first fair execution environment for cloud-hosted exchange that abstracts out the differences in the underlying cloud primitives, tackling execution fairness and simultaneously addressing persistent gaps in the communication fairness of existing systems.
- We introduce an efficient mapping strategy of the virtual time overlay to real-time operations while maintaining low latency and high execution throughput to MPs.
- We evaluate Cuttlefish using an end-to-end implementation on a real cloud platform. When serving 100 MPs, Cuttlefish guarantees fairness in both communication and execution, while introducing low overhead and achieving latency and execution throughput close to the limits of the underlying cloud hardware.

[2] The animal renowned for its ability to see invisible polarized light to discern subtle changes in murky waters for navigation and communication.

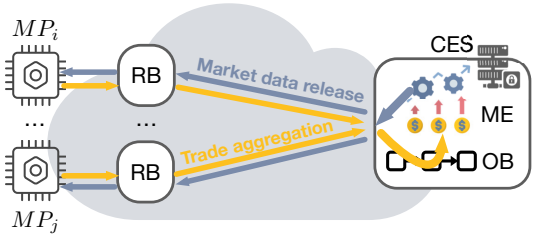


Figure 2 Basic structure of cloud-hosted exchanges. All system components are controlled by the trusted cloud provider.

2 Background

Historically, financial exchanges were bustling places where people would shout orders, negotiate prices, and physically exchange papers representing ownership of stocks or other assets. In today's financial markets, however, the vast majority of trades are executed by computers rather than by humans, opening up the possibility of so-called algorithmic trading techniques, now the cornerstone of modern financial markets.

Algorithmic trading refers to the process of making trade decisions with the help of computer programs. Under this umbrella, low-latency algorithmic trading, which focuses on fast (e.g., <1 ms) reactions to real-time market data with minimal human intervention, has become critical to market efficiency, price discovery, liquidity, and low transaction costs. These algorithms account for a significant portion of the trading volume in financial markets [2, 24, 31, 35, 56]. Compared to the broader set of algorithmic trading strategies, the logic of MPs who participate in low-latency algorithmic trading is relatively simple (designed for quick reactions) and features highly optimized data path logic [36, 33].

Exchanges that support low-latency algorithmic trading expose a simple top-level abstraction: (1) the exchange broadcasts market data to all MPs within the exchange, delivered at time $\{D\}$, (2) the MPs analyze the data, and (3) they return buy/sell orders to the exchange at time $\{S\}$, processed in the order of submission.

In the most cases, the market data feed is the sole input into the system, and orders are handled entirely within the exchange. However, MPs sometimes also integrate outside data into their strategies. Exchanges can also provide alternative trading interfaces [27], although these are typically three to four orders of magnitude higher latency and used by MPs for less latency-critical trading (e.g., those leveraging complex machine learning models). In this work, we assume that all such external interactions are funneled through the CES (in the case of alternative trading interfaces) or a special gateway node (Section 7; in the case of outside data). Clearly, Cuttlefish cannot control the outside world or make it fair, rather, our focus is only on the variability and bias of communication and execution after it enters the system through these nodes. We discuss the interaction of Cuttlefish with the broader ecosystem in Section 11.

Cloud-hosted exchanges. As mentioned, there has been recent interest in cloud-hosted exchanges for reasons including better scaling, fewer outages, improved cost savings, and broader access to the financial markets [24, 19, 17, 35].

Figure 2 depicts the main components of these exchanges. At the core of these systems is the CES, which disseminates market data to all MPs through Release Buffers (RBs) or equivalents. The communication between CES and MPs typically leverages a reliable message transport [24, 19]. Then, the MPs – hosted by proxy cloud instances in the same cloud

Table 1 Summary of notations. Each refers to a scalar timestamp in real or virtual time.

Notation	Definition
$G(x)$	Wall-clock time when market data x is generated.
$D(i,x)$	Wall-clock time when data x is delivered to MP_i.
$S(i,y)$	Wall-clock time trade y is submitted by MP_i.
$\widetilde{D}(x)$	Virtual time assigned by CES for delivering data x.
$\widetilde{S}(i,y)$	Virtual time when trade y is submitted by MP_i.
$\widetilde{CES}(t)$	Virtual time of CES at a wall clock time t.
$\widetilde{OB}(t)$	Virtual time of OB at a wall clock time t.
$\widetilde{MP_i}(t)$	Virtual time of MP_i at a wall clock time t.
$\widetilde{CES}^{-1}(vt)$	Wall clock time of CES at a virtual time vt.
$\widetilde{OB}^{-1}(vt)$	Wall clock time of OB at a virtual time vt.
$\widetilde{MP_i}^{-1}(vt)$	Wall clock time of MP_i at a virtual time vt.

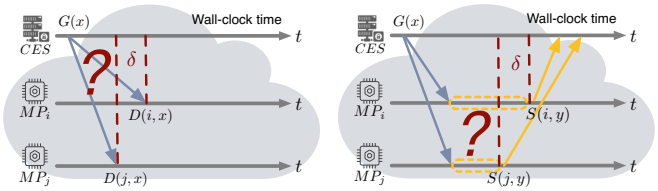
region as the CES – compute their trading decisions and forward them to the CES.[3] There, trades are first enqueued and sorted at the Ordering Buffer (OB) and then processed by the Matching Engine (ME). The ME finally updates the limit order book and generates a new batch of market data.

Performance variations of cloud primitives. Unfortunately, despite their attractive properties, in our conversations with major cloud providers, financial exchanges, and trading firms, there is still a fundamental distrust of the performance properties of the underlying infrastructure. In particular, for both exeuction and communication, performance variations have the potential to invalidate the benefits of careful design, creating a world where MPs win and lose not on the strength of their algorithms but on purely external factors (e.g., temporal variation or provider monitoring/maintenance).

To illustrate, consider the simple scenario of delivering market data x for MP_i and MP_j in Figure 3a. Coordinating delivery to ensure $D(i,x) = D(j,x)$ is difficult due to unpredictable and unbounded path latencies. More importantly, even with simultaneous data delivery, the same hardware substrate (e.g., same server SKUs), the same software stack (OS and the MP's trading algorithm), and the same algorithm, the exeuction time can still vary (e.g., due to different thermal state). This leads to non-deterministic submission times $S(i,y) \neq S(j,y)$, as shown in Figure 3b. In both cases, *any* disparity, even at nanosecond time scales, can advantage/disadvantage an MP.

Recent measurements on cloud-hosted exchanges have quantified the danger of latency spikes in modern clouds [24, 19]. Equally important, however, is the bias and variability in execution performance, which can also occur for any number of reasons, including everything from non-deterministic software operations to machine-specific hardware wear and thermal effects, with prior measurement citing the Coefficient of Variation (CoV) of performance in bare-metal infrastructure of up to 30% [54, 38]. These are on top of noisy neighbor and hypervisor effects introduced in cloud deployments. Our benchmarks on Azure cloud, shown in Figure 1, validate the presence of these biases and variances. Even for single-tenant `F72_v2` instances, with all physical cores of the machine reserved, the interquartile range (IQR) is still 20 ns with disparities exceeding $> 1\,\mu s$ at $p99.9$.

[3] Cross-region deployment of VMs is feasible (e.g., through virtual network (VNet) peering [44]), but not suitable for low-latency communication [24, 19].



(a) Communication unfairness. **(b)** Execution unfairness.

Figure 3 Simultaneous data delivery and execution fairness are difficult, even under the same market data, MP algorithm, and execution platform.

While in both cases it may be possible for cloud providers to try to tame this effect, for instance, by removing all management and telemetry infrastructure or carefully controlling temperature and wear, resulting in much tighter SLOs, (1) doing so would substantially cut into the cost and scalability advantages of cloud-hosted exchanges, and (2) this still does not eliminate disparities that can result in unfairness.

3 Goals and Related Work

Our goal in this work is to create a fair and predictable execution environment where the outcome of competitions is based on only the properties of MPs' algorithms rather than luck. More specifically, we target:

R1 *Communication fairness:* (*a*) All MPs should get access to the market data points at the same time and (*b*) in the other direction, MP_a's trade should execute before MP_b iff MP_a submits a trade before MP_b.

R2 *Execution fairness:* Given any execution of the algorithm on the platform, the submission time of the generated trades is *completely* defined by the delivery time of the input data to the algorithm. Consequently, given *R1*, for a given trading algorithm, the execution time to generate the trades should not vary across MPs.

To be practical, MPs should be able to sustain the above requirements while maintaining low latency and high execution throughput for their trades and algorithms. As previously noted, we explicitly do not consider the fairness or feasibility of cross-exchange arbitrage, which is likely impossible in a partial deployment scenario.

Prior work on cloud exchange fairness. Existing cloud-hosted exchanges, despite focusing exclusively on communication [24, 17, 19], provide incomplete guarantees, even for *R1*. For example, CloudEx [17] and Octopus [18] enforce high-resolution clock synchronization among all RBs and the CES. The CES, upon generating market data at time t, assigns a future release timestamp $t + \Delta t_r$ with a predefined threshold Δt_r, allowing RBs to forward the data simultaneously. Similarly, when an inbound trade arrives at an RB at time t, the CES enqueues it to the OB until $t + \Delta_d$ where the delay Δ_d allows earlier trades to arrive within this headroom. Unfortunately, even with perfect clock synchronization – a strong assumption in distributed systems [37] – the guarantees break whenever latency spikes exceed the threshold. Such latency spikes can occur unpredictably in cloud environments [24, 19, 34]. Configuring conservative threshold values can help (at the cost of performance) but are not a complete remedy [24, 19, 13].

More recent work, DBO [24, 19], relaxes the requirement of clock synchronization and proposes logical clocks based on MP response time. Briefly, DBO offloads RBs to local SmartNICs that measure and tag each MP's response time, while trades are being ordered

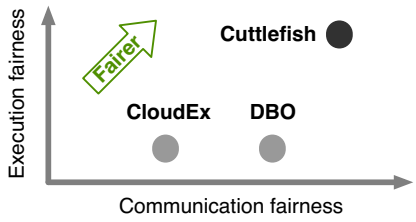


Figure 4 Recent cloud-hosted exchanges [17, 24, 19, 18] target only communication fairness, but still struggle to achieve it. We discuss other related works in [57].

accordingly at the OB/CES. This method corrects inaccuracies in simultaneous data delivery post hoc and provably guarantees Limited Horizon Response Time Fairness for of *R1*. However, DBO's guarantees are limited to a specific trading pattern, namely trigger-point-based high-speed trades. Trades that do not fit this model (e.g., trades triggered using two or more data points) are not necessarily fair.

4 Virtual Time in Cuttlefish

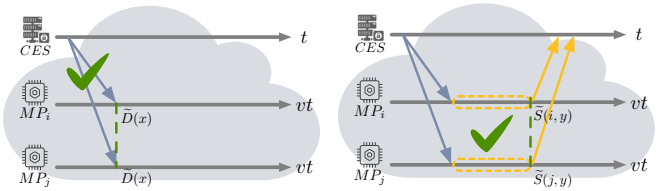
Cuttlefish tackles both execution and communication fairness simultaneously. As depicted in Figure 4, we find that including execution fairness not only serves to present a more predictable execution platform, it also naturally addresses the fundamental limitations of existing work on communication.

To guarantee both *R1* and *R2*, our system, Cuttlefish, adopts a "virtual time" abstraction. Virtual time, as a general concept, is not new – there are many instances where it is beneficial to have a global and fine-grained notion of dependencies independent of wall-clock time.

Of particular relevance is the use of virtual time in high-fidelity emulation of processes interacting over a network [3, 32, 9, 4, 23]. In these frameworks, all processes keep a virtual clock for use in coordinating per-process progress and cross-process events, e.g., network communication. Unlike wall-clock time, virtual time is controllable: a process's virtual clock advances only when it is scheduled. Virtual time is, thus, a stand-in for the expected behavior of the emulated network. The framework exploits the ability to pause and resume processes to ensure that all processes are synchronized and events are sequenced correctly according to their virtual time.

Network emulation vs. low-latency algorithmic trading. Cuttlefish takes an analogous approach by assigning virtual time to all communication and execution – down to an instruction level. Like emulation, Cuttlefish benefits from the ability to control the fine-grained progression of virtual time for each MP (pausing and skipping forward as necessary). Unlike emulation, however, low-latency algorithmic trading presents a substantially different set of goals and knobs.

Soft real-time constraints on virtual time progression. Generally, the primary concern of network emulation is fidelity to a target emulated network. The relationship between the emulator's virtual time and wall-clock time is of secondary importance, with the most important impact being its effect on the end-to-end execution time of emulation. In contrast, Cuttlefish is a platform for trading real-world financial instruments, so consistent timeliness is critical, especially in the presence of alternative trading interfaces and external data.



(a) Communication fairness. **(b)** Execution fairness.

Figure 5 Communication and execution fairness are achievable in the virtual time domain through the deterministic control of virtual time passage and the quantification of execution.

Control over input frequency. Emulation's focus on fidelity also generally assumes a "correct" emulation target. In contrast, the CES in Cuttlefish has significant control over market data delivery times – what matters is the fairness of the delivery, not fidelity to any particular execution. Cuttlefish uses this control to adjust the market data delivery rate in response to the current load and to allow lagging nodes to catch up.

Cuttlefish virtual time, illustrated. Figure 5 depicts the operation of virtual time in Cuttlefish. The Cuttlefish CES operates unrestricted in *wall-clock time*, while the MPs track and adhere to *virtual time*. For simplicity, we will ignore component failures but discuss how Cuttlefish can be extended to handle them in Section 8. The notation used in the figure and the remainder of the paper are summarized in Table 1.

1. [Figure 5a] For fair market data delivery (*R1a*), the CES picks and tags a virtual time $\widetilde{D}(x)$ for the release of each data point x. Each MP's local execution runtime controller ensures the release accordingly.
2. [Figure 5b] For execution fairness (*R2*), Cuttlefish provides a deterministic accounting of compute (in virtual time). Specifically, it counts the instruction cycles executed by its platform-agnostic virtual machine substrate. This ensures that, for any MP_i consuming an input x at $\widetilde{D}(x)$ and producing a trade y, the resulting $\widetilde{S}(i, y)$ is deterministic, regardless of execution performance variations.
3. [Figure 5b] For fair trade ordering (*R1b*), trades from an MP are marked with the virtual time at which they are generated, $\widetilde{S}(i, y_1)$ and $\widetilde{S}(j, y_2)$. Similar to DBO, Cuttlefish features an ordering buffer that forwards these trades to the CES based on their generation virtual time in monotonically increasing order. Thus, the ME processes y_1 before y_2 if and only if $\widetilde{S}(i, y_1) < \widetilde{S}(j, y_2)$.

Strict adherence to virtual time on all MPs ensures both (R1) and (R2). Further, the CES's ability to bridge wall-clock and virtual time ensures minimal delta between the two (Section 8).

5 Design Overview

Instantiating virtual time abstraction end-to-end necessitates several building blocks: How can we express computation in the virtual time domain (§6.1)? How do we execute the programs efficiently (§7)? How can we track and control the virtual time advancement (§8)?

Architecture. Similar to other cloud-hosted exchanges [24, 19], VMs executing MPs are co-located in the CES region for low-latency services. Unlike existing systems, however, users of Cuttlefish build on a platform-agnostic virtual machine substrate clocked by virtual time cycles, mediating all MPs' computational and I/O operations through this abstraction.

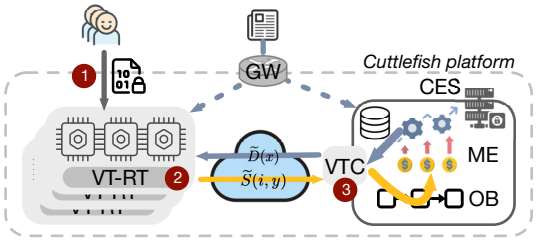


Figure 6 Overview of the Cuttlefish platform.

Figure 6 depicts Cuttlefish's high-level architecture. For simplicity, most of our discussion will assume homogeneous, single-threaded market participants and the market data stream as the only input to the ecosystem. We describe the integration of external data in Section 7, extension to heterogeneous compute in Section 11, and multi-threading support in [57].

Workflow. Building on the typical cloud-hosted exchange architecture, Cuttlefish introduces:

❶ *MP algorithm representation via eBPF VM bytecode [Section 6].* To account for the amount of execution deterministically, Cuttlefish leverages a platform-agnostic IR that is based on the eBPF Virtual Machine (VM)[4] instruction set. Cuttlefish advances virtual time based on the consumed number of VM instruction cycles, allowing it to abstract out potential variances in the underlying infrastructure. Cuttlefish also adapts eBPF user-space libraries to support a simple but expressive programming interface. Cuttlefish verifies, instruments, and translates this code from MPs to native assembly for the underlying computation platform for efficient execution.

❷ *Virtual time execution runtime (VT-RT) [Section 7].* Cuttlefish develops a runtime execution environment that can efficiently utilize all available cores to execute the binaries for MPs allocated to the same cloud VM. It also manages a range of real-time operations for the responsible MPs, including tracking and advancing the virtual time, data delivery, and local batching of trades and heartbeat to the central CES for ordering based on the submitted virtual time.

❸ *Virtual time control (VTC) [Section 8].* Cuttlefish integrates a virtual time control algorithm for the CES to assign virtual market data release times. By controlling virtual time assignment, the CES controls how much compute throughput is available to each MP. This is crucial to mitigating underlying network latency spikes or slowdown in execution behaviors.

The cost of fairness. Cuttlefish prioritizes fairness, which is the primary concern for exchanges [47, 24, 19]. This trade-off is intentional, as a slower exchange operating within the higher latency bounds of public clouds can still meet market needs, provided it delivers market data and accepts orders uniformly across all participants [24, 47, 39].

In exchange for fairness, Cuttlefish incurs modest overheads on MP execution. Some of this is due to the extra instrumentation to track and control virtual time (∼2–20%). More fundamentally, guaranteeing that all MPs have equal opportunity means that system-wide progress is gated on the slowest node. This limitation is intrinsic to any fair system. Prior work like DBO and CloudEx noted similar limitations when equalizing network delay; Cuttlefish incurs the same for execution.

[4] A virtual machinery abstraction, not to be confused with physical cloud VMs.

Despite these trade-offs, in our evaluation on a public cloud and 100 MPs (approximately the maximum scale of most existing exchanges [24, 27]), Cuttlefish incurs low overhead in both latency and throughput under real-world performance variation, approximating the limits of the underlying cloud hardware (Section 10.3). If higher throughput is needed, better hardware or multi-threaded execution ([57]) can help. Also, the observed latency is well within the requirements of many major exchanges.[5] Regardless, Cuttlefish still guarantees fairness and predictability in all cases.

Trust assumptions and threat model. Moving to a cloud-hosted solution fundamentally requires MPs to trust the cloud provider and trading platform to not steal or manipulate the MP trading code and execution [17, 19, 24]. Such requirements are typically enforced through contracts, log auditing, privacy laws, and regulatory bodies such as the SEC, and the same prohibitions apply here. Recent advances in hardware (e.g., Intel SGX [10]) as well as efforts in cloud confidential computing [46, 1, 26] could also benefit Cuttlefish by providing cryptographic attestation and secure enclaves during both instrumentation and execution. An exploration of these mechanisms is out of the scope of this work.

For the provider's side of the provider-MP relationship, following prior work [24, 19, 17], we assume that all core system components – including release and ordering buffers – are trusted and protected against tampering. As we discuss in Sections 6.2 and 8, Cuttlefish's abstractions naturally defend against problematic instructions or program structures. Otherwise, MPs can utilize the system in arbitrary ways, including trying to slow down virtual time advancement through interaction with the CES and execution engine.

For all other interactions (e.g., MP-to-MP or to/from malicious third parties), we assume secure and exclusive communication channels between MPs and Cuttlefish components using standard public cloud features, such as security groups [24, 19] to prohibit outside communication with any of these entities. While it is possible that malicious actors may attempt to indirectly influence these connections, e.g., through datacenter-level DDoS attacks, cloud providers themselves are already quite capable of mitigating, defending against, and provisioning for these types of attacks.

6 MP Algorithm Representation

This section elaborates on Cuttlefish's abstractions and platform-agnostic IR, taking a top-down approach.

6.1 Programming Interface

Recall from Section 2, MP algorithms consist of processing CES data feeds to make trading decisions that aim to optimize profit from price disparities, bid-ask spreads, or liquidity subsidies. To allow users to easily program MP algorithms, Cuttlefish utilizes a simple event-driven programming interface.

MP handler abstraction. Figure 7 shows a simplified example of how users may express trading logic with Cuttlefish's *mp_handler* interface.

[5] A major exchange, IEX, prides itself on fairness with a 700μs latency [27, 24].

```
1  #include <cuttlefish_user.h> /* Single include of whitelist APIs */
2  int mp_handler (subscribed_context_t* ctx):
3      if (ctx->price) > 100 then
4          trade_t trade = 1; /* Sell */
5          submit_trade(&trade); /* Just-in-time trade submission */
6      else if (ctx->price) < 10 then
7          trade_t trade = 2; /* Buy */
8          submit_trade(&trade);
9      update_map(0, &ctx->price); /* Save the history price  */
10     return 0;
```

Figure 7 An example MP pseudocode in high-level language using **Cuttlefish service APIs**, which includes a narrow interface to a KV store for stateful invocations.

An MP's handlers are invoked serially on each subscribed market data point. Virtual time advances on every new invocation (in accordance with *R1a*) and on every execution of an IR instruction (with a fixed virtual time cost per instruction).

More specifically, for each market data x, the virtual time of MP_i is updated according to the rule $\widetilde{MP_i} = \max(\widetilde{D}(x), \widetilde{MP_i})$. This involves two scenarios: (1) If the prior invocation finishes before $\widetilde{D}(x)$, Cuttlefish advances $\widetilde{MP_i}$ to the target virtual time and releases the data; and (2) if the MP handler chose to consume more cycles that ends up overshooting $\widetilde{D}(x)$, $\widetilde{MP_i}$ remains unchanged. MPs can submit trades at any point in this process. Each trade's ordering is determined by the exact virtual time of the associated `submit_trade` call.

The cost of each instruction is fixed and public knowledge (for details on the map between individual IR instruction types to its virtual time cost, see Section 9).

eBPF IR. Although any platform-agnostic runtime could serve as a virtual hardware substrate, Cuttlefish chooses an IR based on the eBPF VM instruction set. This IR is compelling for many reasons: it is simple (a 64-bit RISC register machine), it has a mature ecosystem including support for various language frontends, and it is widely supported in multiple target architectures including specialized hardware accelerators (e.g., FPGAs, smartNICs) [5, 30]. More importantly, the simple eBPF Instruction set architecture (ISA) allows us to easily enforce a constrained memory access model and reason about safety by verifying MPs' eBPF bytecode accordingly before execution through static analysis [53, 16, 30].

We note that using the eBPF IR does *not* mean that we are using the kernel-based eBPF VM. While the kernel-based VM [16] imposes restrictions that limit expressiveness, e.g., loop bounds, Cuttlefish does not impose such constraints. Thus, it provides a *Turing-complete* interface [16, 53] for trading strategies, and we show examples of these in Table 4.

Usability. Users can write their trading programs directly in eBPF bytecode (and will likely do so for performance reasons), or they can use more accessible toolchains (such as `llvm`'s eBPF backend) to compile the MP expressed in a high-level language like `C` to the bytecode and then sent (e.g., as an `elf` file) as input to Cuttlefish.

Service APIs. To enable user access to Cuttlefish's trading services, Cuttlefish provides a single header file that contains main data structures and a *whitelist* of shared service APIs. These include: (1) primitive service APIs for trade submission and virtual time facilities, as well as a runtime context object for accessing real-time market data, current virtual time $\widetilde{MP_i}(t)$, and the release virtual time $\widetilde{D}(x)$ for the current invocation, (2) a narrow interface for KV store interactions (e.g., update, lookup) for stateful invocations, and (3) extensible built-in computational helpers like FFT – which users can optionally leverage for convenience – although users can also write their own implementations in the MP handler.

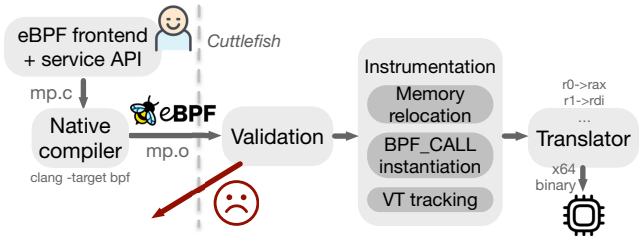


Figure 8 MP bytecode processing workflow.

6.2 MP Bytecode Processing Lifecycle

Figure 8 illustrates the subsequent processing pipeline of MP bytecode: Cuttlefish first validates and instruments the bytecode before final JIT compilation to native hardware binary for safe and efficient execution.

Validation. Cuttlefish ensures the safety of input bytecode through a validation process similar to that of kernel space eBPF VMs [16, 30]. It rejects programs that attempt memory interactions beyond the allowed indirect KV store access, such as through dynamic memory allocation. Additionally, the use of `BPF_CALL` instructions is restricted to the predefined set of service APIs in Section 6.1, blocking any attempts to invoke unsupported functions through illegal opcodes. Further security checks are described in Section 8.

Note that this step requires that MPs trust the cloud provider and the trading platform operator to not tamper with the ordering of trades or MP code; however, as noted by prior work, this trust is fundamental to a cloud-hosted paradigm [17, 19, 24]. We discuss enforcement mechanisms further in Section 11.

Memory relocation and service API instantiation. As MP programs operate within a constrained memory access environment, Cuttlefish performs memory relocation for those requesting access to KV maps. In particular, it dynamically resolves and replaces symbolic references in the KV map API's `BPF_CALL` instructions with the appropriate memory address during eBPF bytecode loading. This indirection process and the dynamically assigned addresses are invisible and inaccessible to the MPs.

Virtual time tracking instrumentation. To track the virtual time efficiently, Cuttlefish takes a passive, non-intrusive approach via binary rewriting [4, 58], shown in Figure 9:

(1) *Basic block segmentation:* The bytecode is split into basic blocks (BBs) – straight-line sequences of instructions without branches – to facilitate batched virtual time increments Δvt. In addition to `BPF_JMP` call sites, trade submission calls also serve as instrumentation points for capturing the most recent virtual time as the trade needs to be tagged accordingly. For large blocks, Cuttlefish inserts dummy trade submission calls for timely updates of virtual time progress of the MP.

(2) *Virtual time increment instruction:* Cuttlefish emits native machine code (two instructions for x64) at the epilogue of each block to update MP_i's virtual time by addressing the memory location storing $\widetilde{MP_i}$ during JIT translation.

(3) *Offset correction:* The instrumentation also updates the offsets for the JMP instructions. The absence of indirect jumps in the eBPF assembly simplifies this step.

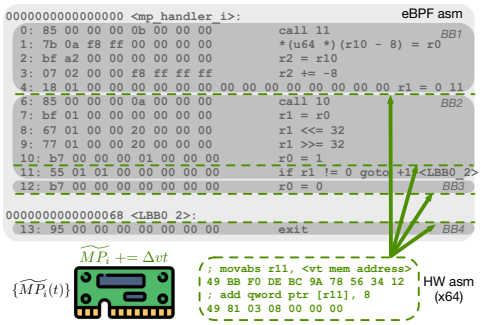


Figure 9 Virtual time update instrumentation.

Finally, instead of the slower eBPF interpreter, Cuttlefish adapts eBPF JIT translator to compile IR bytecode into native binaries (e.g., x86_64). This two-tier compilation ensures fair, platform-agnostic virtual time tracking and efficient execution on native hardware.

7 Cuttlefish Execution Runtime

Cuttlefish features a practical runtime architecture efficiently implementable on modern clouds. We describe a single VM's execution runtime (Figure 10); extending to multiple is straightforward.

Communication with CES. Cuttlefish runtime interfaces with CES via the data dispatcher and the trade aggregator, both exchanging data streams over a reliable transport layer.

Data dispatcher. The dispatcher manages inbound market data, each coming with an assigned virtual release time.

Trade Aggregator. The trade aggregator gathers tuples from MPs, each comprising a trade decision y and its virtual submission time $\widetilde{S}(i, y)$. These tuples are locally sorted by submission time, batched, and sent to the OB for global sorting. The aggregator is additionally responsible for sending heartbeats to the CES to indicate the latest virtual time reached by all local MPs. The OB uses these heartbeats to decide when it can forward the trade with the lowest virtual submission time in its buffer to the CES safely (i.e., there are no in-flight trades with lower virtual time). This localized handling of sorting and heartbeat calculations enhances the CES's scalability.

Local execution workflow. Cuttlefish's runtime allows consolidating multiple MPs into multi-core VMs for efficiency. It also maximizes CPU utilization (for execution-throughput) and eliminates blocking operations along the data path (for latency). Central to its workflow are the worker threads that execute MPs in parallel, each affined to a dedicated CPU core and configured to run a loaded MP binary. Interaction with the dispatcher and aggregator is streamlined using lock-free, cache-efficient Single-Produce-Single-Consumer (SPSC) rings to minimize processing latency.

The worker threads operate in a busy loop with minimal stalls (e.g., context switching) during the execution. It first polls a batch of command items that contain market data from the command ring. For each market data x processed by MP_i, the worker thread updates the virtual time and invokes the binary with the new market data immediately. Each worker then runs hardware binaries instrumented for uninterrupted virtual time advancement, as

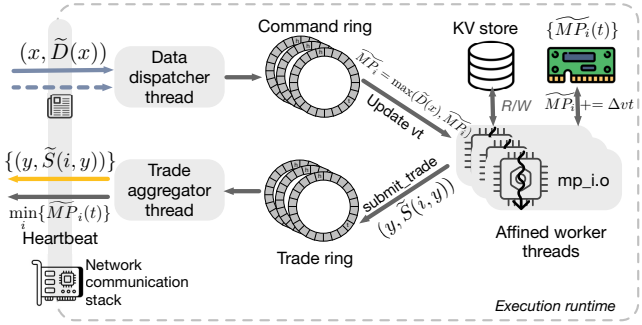


Figure 10 Overview of Cuttlefish's execution runtime.

outlined in Section 6.2. During execution, MP handlers access the KV store in a thread-safe manner and invoke the redirected function that enqueues trades to the ring with a virtual submission time of the `submit_trade` calls.

External data handling. In addition to supporting internal data feeds from the CES, Cuttlefish also accommodates external data sources. These interactions are done through a designated gateway (GW) node, as shown in Figure 6, which enforces an additional virtual time latency comparable to that of the public Internet.[6] Just as with current exchanges, we expect that MPs can and will leverage a variety of optimization for these messages, e.g., microwave [7, 6] and satellite [25, 51, 28] networks to surpass wireline c limitations, and/or employ human oversight to tweak parameters in response to changing conditions or unexpected events. Cuttlefish's goal is only to ensure that the post-GW transmission and delivery of the data is fair and predictable. MPs can access the data through the subscribed data context input to the handler, just as they do with normal CES data. We detail the mechanism in [57].

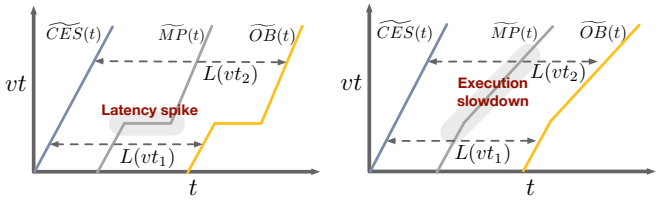
8 Virtual Time Control

So far, we have discussed how the CES broadcasts data and processes aggregated trades from MPs in the order of their virtual submission times at the OB. Another key role of the CES in Cuttlefish is to assign the virtual release time, $\widetilde{D}(x)$, tagged to each market data x for delivery by runtime engines. Intuitively, this virtual time assignment process resembles congestion control but concerns regulating the rate of virtual time progression rather than bytes on a wire.

Objectives. Cuttlefish's virtual time control targets two goals:

1. Minimizing *latency* ($L(vt) = \widetilde{OB}^{-1}(vt) - \widetilde{CES}^{-1}(vt)$), defined as the time difference between the CES releasing market data with virtual time vt ($\widetilde{CES}^{-1}(vt)$) and when the OB hears from all MPs until vt ($\widetilde{OB}^{-1}(vt)$). In simple terms, latency here refers to the minimum time between when market data is produced at the CES and when a trade from an MP using this data can be executed. This definition extends previous end-to-end latency concepts [24] for trades ordered per virtual time.

[6] A modern commodity switch can provide Tbps capacity [49, 50], which is sufficient to handle typical quantities of external data to exchanges. The GW can scale beyond a single node with a shared clock.



(a) Increased network latency. **(b)** Slower execution.

Figure 11 Examples when the latency gets impacted by environmental conditions. For the purpose of illustration, the diagram simplifies the discrete steps on events of market data release, invocation, and trade response receipt.

2. Maximizing *overlay execution-throughput* $\theta = \Delta vt/\Delta t$. This represents the rate of virtual time advancement at the CES or, equivalently, the number of IR instruction cycles available to each MP (per unit of wall-clock time) to process the incoming market data.

Note that neither of these goals affects correctness, fairness, or predictability. Rather, a good virtual time assignment is important for purely performance reasons. Specifically, virtual time assignment that is too slow can limit the execution-throughput of the exchange, even when the underlying MPs are capable of supporting a higher virtual-time throughput. Conversely, virtual time assignment that is too fast can increase worst-case $L(vt)$ due to MPs that are lagging behind.

Methodology. As Cuttlefish seeks to guarantee fairness across all MPs, stragglers (depicted in Figure 11) can influence virtual time control. Stragglers can arise for two reasons. First, increased network latencies delay market data delivery to MPs, slowing virtual time progression. For example, in Figure 11a, a spike in latency from CES to an MP results in latency growth from $L(vt_1)$ to $L(vt_2)$. In reality, network latencies for both paths (CES-to-MP or MP-to-OB) can affect $L(vt)$. Virtual time assignment should try to mitigate the effect of such spikes. Second, execution slowdowns at an MP (e.g., due to change in processor frequency) can reduce the rate of virtual time progression, cumulatively affecting latencies if these slowdowns are prolonged. Figure 11b illustrates a simplified example of this effect.

To account for them, Cuttlefish presents an easy-to-reason-about approach by coupling its virtual time assignment with real-time evolution. We detail the assignment algorithm in [57]. The key idea of our algorithm is similar to that of BBR congestion control protocol [8]. In particular, Cuttlefish measures τ_i, which is the estimated computational capability of MP_i (i.e., how many virtual cycles can the executing engine of MP_i process per unit wall-clock time). We then control the virtual time progression rate based on the bottleneck compute capacity, $\min(\tau_i)$. Similar to BBR, Cuttlefish applys a window cap on the max "in-flight" virtual time as a guardrail to prevent excessive virtual time assignment in worst-case scenarios.

Failure handling. As described, any MP failure will halt the progress of virtual time. Cuttlefish incorporates timeouts at the CES to detect such events.[7] Because of the determinism of virtual time, with periodic check-pointing of the MP state, the CES can restart and/or relocate such failed components.

[7] Spurious timeouts may degrade performance, but will not affect fairness.

We further note that eBPF uses predefined key-value stores for managing state, which makes identifying state and replicating it straightforward. Cuttlefish is also amenable to replication of the MPs themselves. This approach can potentially mask the impact of failures on virtual time progression (§10.4).

An alternate choice would be to remove the failed MP from the VTC assignment and trade-forwarding logic, and then do a clean restart. The failed MP incurs unfairness in this case. In some abstract sense, Cuttlefish is subject to a CAP-theorem-like limitation: here, the choice is between fairness and progression of virtual time progress in case of failures.

CES failure. CES fault tolerance is beyond the scope of this work; to the best of our knowledge, existing CESes also rely on state replication (e.g., of the order book) for fault tolerance.

Security. While MP code will run alongside other MPs and Cuttlefish components, Cuttlefish benefits greatly from its IR abstraction, basis in virtual time, and validation process (Section 6.2). Potentially problematic instructions or program structures (e.g., attempts to access system memory or cycle counters for creating side channels) are explicitly disallowed. Communication is similarly restricted. As mentioned in Section 5, Cuttlefish adopts a model that forbids MP-to-MP communication similar to prior proposals [24, 19]. However, whereas prior work punts on enforcement, Cuttlefish's constrained runtime provides natural avenues to guarantee it.

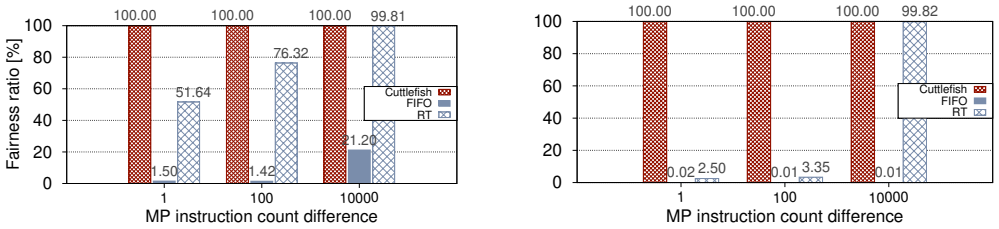
Within the confines of the system, the main potential avenue of attack for MPs is the manipulation of virtual time. Note that MPs with fewer instructions per invocation do not affect system-wide progress, as virtual time advances independently. Instead, MPs might try to deliberately slow their virtual time advancement speed for a chance to influence $\min(\tau_i)$ (if they are the slowest MP in the system) by picking sequences of instructions that have the largest ratio of real-time cost to virtual-time cost. Conversely, if they are already the slowest MP, they could speed up to try to accelerate virtual time advancement. Within Cuttlefish, there is no advantage to manipulating the advancement speed; instead, differences only arise in interactions with external communication and events. We note that the impact is limited: Cuttlefish's GW ensures that external coordination is inefficient (Section 7) and the extent of manipulation is constrained to the gap between the "slowest" virtual program and the second slowest MP. In the end, however, the critical guarantee that Cuttlefish provides is that any variability in virtual time advancement speed is experienced equally by all MPs.

9 Implementation

To demonstrate Cuttlefish's practicality, we developed and ran a prototype on standard public cloud VMs [43].

Processing MP handler programs. Cuttlefish supports the end-to-end workflow for MP handlers described in Section 6, exposing a single-header interface for Cuttlefish's virtual time services. For verified eBPF bytecodes provided by users, Cuttlefish embeds virtual time tracking transparently through binary writing, while achieving high performance by leveraging the existing eBPF JIT compiler to emit native code [30].

Virtual time cost of instructions. By default, Cuttlefish assigns the virtual time for eBPF instructions based on the equivalent hardware instructions on standard CPU models (e.g., x86_64) and leveraging previous studies that have extensively quantified their costs per



(a) Same VM. **(b)** Different VM types.

Figure 12 Cuttlefish guarantees 100% fairness ratio, whereas FIFO and RT-based ordering can only approximate even when MP_b spends non-negligible number of instructions more than MP_a.

machine cycles or reciprocal throughput in modern hardware architecture [14]. For example, `BPF_ADD` is assigned one virtual time unit, with other operations, including handler invocation and `BPF_CALL` instructions for built-in service APIs such as the access to the KV store, scaled accordingly. In Cuttlefish, the relatively small eBPF instruction set [29] simplifies the process. While actual instruction cycle times (e.g., memory accesses) can vary in wall-clock time due to microarchitectural effects such as caching, these variances are abstracted away in the virtual time domain. The idea behind such an assignment is to, as much as possible, reduce the dependency of τ on the IR instructions used by the MP in its code. However, exchanges can customize their cost model according to their needs – as long as the models are transparent, the system is fair and predictable.

Supporting efficient execution in virtual time. Cuttlefish employs a reliable, message-based transport [42] for its dispatcher and trade aggregator thread, interfacing with the CES/OB. Worker threads are affined to dedicated cores and are responsible for invoking MP handlers. To facilitate communication between workers and the data dispatcher, Cuttlefish uses a cache-efficient, lock-free Single Producer/Single Consumer circular buffer implementation to instantiate the market and trade rings respectively as detailed in Section 7.

10 Evaluation

10.1 Unfairness in the Cloud

Ordering mechanisms. In addition to Cuttlefish, we examine:

- Response Time (RT) based ordering, which ranks trades by MP processing time. Since CloudEx and DBO either require high-resolution clock synchronization or SmartNIC support – hard to replicate in our environment – we use (1) as a proxy to evaluate behaviors (potential execution unfairness) of CloudEx under perfect network communication fairness and that of DBO's logical clock based on response time. Under identical execution conditions, RT-based ordering should yield 100% fairness ratio.
- FIFO ordering, which processes trades by OB arrival time. Under ideal network and execution conditions, FIFO should also achieve 100% fairness.

Setup. We consider two MPs. MP_b executes N additional primitive IR instructions than MP_a, with all other aspects being identical. In theory, an identical network and execution environment must always prioritize the trades of MP_a over those of MP_b triggered by the same market data. To quantify fairness, we define fairness ratio as the fraction of MP_a trades were (correctly) ordered ahead of corresponding trades from MP_b.

Table 2 $L(vt)$ for 100 MPs: Cuttlefish achieves $\theta = 3279$M vt/s with avg. $\tau_{min} = 3702$M vt/s.

	Latency (µs)				
	avg.	**p50**	**p90**	**p99**	**p99.9**
MaxRTT	52.04	47.74	49.95	55.85	144.2
Cuttlefish	54.19	50.82	53.49	68.46	166.3

We run experiments on Azure, with one VM operating as CES to generate market data messages at ≈100 µs intervals. We examine two scenarios: (a) MP_a and MP_b run on identical VMs with `Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz`; (b) MP_b uses `Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz` while MP_a running on a different CPU `Intel(R) Xeon(R) CPU E5-2673 v4 @ 2.30GHz`. In approach (2), we use lightweight, high-resolution `rdtsc` counters – leveraging constant tsc support across modern processor cores – to measure CPU time at fine granularity and piggyback the measurement with the data for OB ordering. A caveat of this experiment is that the fairness ratio for (1) and (2) can be impacted by the specifics of how the MP algorithm is executed. To reduce this impact, we execute the algorithm using the Cuttlefish runtime environment, which uses strategies like core pinning, CPU isolation, and DPDK to minimize the impact of the OS on execution and network. Further and alternative optimizations (e.g., IPU-based network offload) could further reduce this impact, though other variability would still remain.

Observation. Figure 12a shows the fairness ratios of different ordering mechanisms when two MPs operate on the same type of VMs. In this case, FIFO ordering exhibits significant unfairness as the VM of MP_a experiences a higher RTT compared to that of MP_b. When MP_b executes $N = 10000$ more instructions than MP_a per invocation, the fairness ratio only improves to 78.8%. On the other hand, RT-based ordering shows resilience to network latency disparities. Nevertheless, it incurs about 48% unfairness for $N = 1$, attributable to execution time variances. Despite the mitigation effect of increasing N, RT-based ordering still incurs 0.18% unfairness rate, even at $N = 10000$.

Figure 12b presents a different scenario (b) where MP_a operates on a VM with a slower processor and significantly higher RTT from the CES. FIFO's unfairness remains pronounced at higher N values. RT-based ordering, in turn, experiences a substantially reduced fairness ratio due to the disparity in processor frequencies. E.g., with $N = 1$ or $N = 100$, RT-based ordering's fairness ratio drops below 5%. Throughout these scenarios, Cuttlefish consistently maintains deterministic and 100% fairness ratio. Cuttlefish's fairness guarantee by design remains unaffected by variations in underlying computational power and network latency.

10.2 Performance of Cuttlefish

We evaluate the performance cost of Cuttlefish for fairness, focusing on end-to-end latency $L(vt)$ and execution throughput θ (Section 8). We run Cuttlefish in the Azure cloud environment using `Standard_F16s_v2` instances. Our setup includes a CES VM with a ConnectX-4 NIC and `Intel Xeon Platinum 8272CL CPU @ 2.60GHz`. We use 10 VMs to host 100 MPs and another for the gateway, all in the same VNet/region. The CES broadcasts market data at ≈ 100µs intervals.

We also compare the performance of Cuttlefish against its limits (max network latency and minimum compute capability τ_{min} across MPs). To ensure a fair comparison, we measured both network latencies of messages and the computational capabilities of the cores for each MP's core under identical environmental conditions. We record timestamps when market

Table 3 Under higher RTT and background noise: avg. τ_{min} = 2488M vt/s and θ = 2373M vt/s.

	Latency (µs)				
	avg.	**p50**	**p90**	**p99**	**p99.9**
MaxRTT	112.0	101.0	113.7	640.3	2984
Cuttlefish	115.5	104.2	116.8	674.5	2996

Table 4 Example textbook algorithmic trading technical analysis indicators that we have expressed with Cuttlefish's MP handlers.

Abbreviation	**Algorithm**
bbs	Bollinger Bands Strategy
bmm	Basic Market Making
ema	EMA Mean Reversion
macd	Moving Average Convergence Divergence
macs	Moving Average Crossover Strategy
mmacs	Multiple Moving Average Crossover Strategy
obv	On Balance Volume (OBV) + EMA
psar	Parabolic SAR
rsi	Relative Strength Index
sma	SMA Mean Reversion

data arrive at the VM (t_1) and when the corresponding trade response leaves the VM (t_2), as well as when they leave (t_0) and arrive at the CES machine (t_3). We then calculate the RTT per VM based on $(t_3 - t_0) - (t_2 - t_1)$ without needing high-resolution clock synchronization. MaxRTT across VMs is the highest latency across VMs corresponding to the same market data release. Similarly, we measure τ_{min} using `rdtsc` counters.

Table 2 compares Cuttlefish's end-to-end latencies against the MaxRTT across various percentiles. The observed latency discrepancies are attributed to the data flow operations of Cuttlefish, including ring management, batching, and virtual time assignment. Despite trading off some latency for fairness, Cuttlefish shows a commendably low p99.9 tail latency within a typical public cloud setting, in part due to minimal barriers in data release and MP execution. Further, Cuttlefish exhibits a execution-throughput of 3279 M vt/s, which is close to τ_{min} = 3702M vt/s.[8] In a separate experimental setup, shown in Table 3, we incorporated a VM with a comparatively slower processor at 2.3 GHz frequency and a higher RTT. Cuttlefish adapts well, incurring a modest latency overhead and approximating latency bounds even under stressed conditions.

10.3 Instrumentation Cost

Section 10.2 evaluates the performance of Cuttlefish against latency and throughput limits, showing the end-to-end costs associated with online operations of the virtual time overlay. This subsection investigates the static overhead due to virtual time tracking instrumentation. In particular, we evaluate the impact of this instrumentation on compute capacity.

To quantify the cost, we conducted a stress test under a worst-case scenario: invoking the handler at maximum rate through market data in-memory. Our tests covered various programs shown in Table 4, each with different computational logic and memory access patterns. While these algorithms are simple versions of what is actually used in practice, they demonstrate that, as expected, the Turing completeness of our language is expressive

[8] This number should not be directly compared with native cycles/s on a superscalar processor [14, 3].

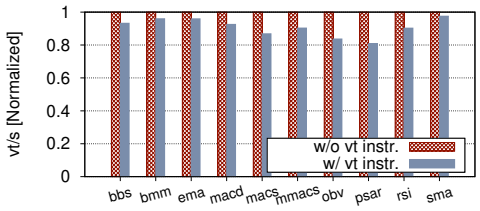


Figure 13 Overhead of virtual time instrumentation across MP handlers in Table 4.

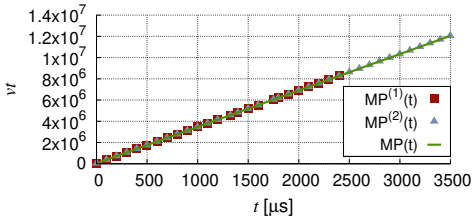


Figure 14 Virtual time series upon a failed MP replica.

enough to support a wide range of trading algorithms, such as those for statistical arbitrage or directional trading. Interestingly, the interface is also sufficiently intuitive to allow GPT-4 to generate the core algorithmic trading programs that are fully compatible.

As shown in Figure 13, the tracking instructions incur 2–20% overhead in throughput as compared to the vanilla executable, depending on the basic block and branch patterns in the program. In particular, those with thinner loop blocks exhibited a higher virtual time tracking cost. Note the evaluations involve raw algorithms. Cuttlefish can also mitigate the virtual time tracking overhead by providing common computational blocks (often involving intensive loops) as a `BPF_CALL` helper, thus reducing the virtual time tracking breakpoints.

10.4 Exploiting Determinism in Virtual Time For Fault Tolerance

To speed up recovery in the event of failures (§8), Cuttlefish can replicate the execution of an MP handler across different machines by exploiting the *deterministic* virtual time progression across replicas. A physical replica gets integrated as usual with Cuttlefish's virtual time overlay, however, the OB processing the trade that arrives earliest in real-time across the replicas to advance the virtual time for the logical MP. This helps improving the fault-tolerance of Cuttlefish. Figure 14 illustrates a case where Cuttlefish replicates a single MP across 2 different VMs. When a replica $MP^{(1)}$ fails (by killing the worker thread), the virtual time of the logical MP still proceeds as the other replicas $MP^{(2)}$ keeps updating its virtual time. With Cuttlefish, the service provider may trade the cost of replication (and the associated traffic overhead) for lower latency and reliability. Beyond fault tolerance, such replication can also help reduce the latency of Cuttlefish [11].

11 Discussion and Future Work

Cuttlefish explores an admittedly extreme design point, but one that provides strong guarantees, presents an attractive and general interface, and is amenable to efficient implementation. It also happens to fit with existing calls from the financial community and SEC to reduce the barrier to entry for retail traders [35, 6, 7]. The entrenchment of existing exchange architectures means that it is unlikely Cuttlefish will supplant traditional exchanges; however, only time can tell if its tradeoffs allow it to carve out a small, sustainable segment of the market. Even so, as computer scientists, we argue that exploring the challenges and opportunities of this design is worthwhile. This section briefly discuss how Cuttlefish potentially fits into the broader cloud/financial ecosystem.

Heterogenous hardware backends. While Cuttlefish currently focuses on a CPU-based interface and runs on common x64-based cloud VMs, the choice of eBPF IR allows operators to benefit from a broader ecosystem, including mature toolchains that compile eBPF bytecode to ARM64 [30] or offload it to accelerators such as FPGAs [5, 15].

In the extreme case, virtual time is flexible enough to support differentiated instance types and price points. For example, clouds could charge more money for more memory or faster processing (via scaled-up virtual time throughput); they could also sell instances with, for example, 8 CPU cores and 4 FPGAs (by leveraging alternative eBPF backends and the techniques in [57]). While this type of support is well out of scope for this paper, one could imagine an IaaS abstraction as rich as traditional cloud offerings, but where each compute device is metered using virtual time.

Interaction with the broader market ecosystem. Securities represent and interact with global assets, and Cuttlefish is only designed to mediate the low-latency algorithmic trading API of the specific exchanges that adopt it; financial activity can occur outside the exchange and even through the slower alternative APIs of an adopting exchange.

In all cases, Cuttlefish eliminates variability and bias between MPs in the exchange. While it potentially also slows them compared to the rest of the world, for securities listed solely at the adopting exchange, Cuttlefish will still be the fastest way to issue buy/sell orders (Section 7). Securities cross-listed at multiple exchanges (Cuttlefish or otherwise) may present opportunities for arbitrage, but that is no different than today, and crucially, the Cuttlefish-enabled cloud platform will never be the determiner of which MP wins the race. As others have noted [47, 24, 7, 19, 39], this prioritization – fairness over pure latency – is a desirable one.

Scalable deployment. Our prototype currently runs with 100 MPs, with the CES broadcasting market data at approximately 100µs intervals. In this setup, the primary bottleneck lies in the CES, which must serialize inbound data ingestion and outbound market data dissemination. One can mitigate bottlenecks with faster processors and high-bandwidth NICs for the CES, which translate to scalability directly. "Beefier" VM SKUs on the MP side can also help, as Cuttlefish's execution runtime is also designed to scale up with the number of cores, which allows greater MP co-location per machine. To that end, modern cloud providers can offer "beefy" VM SKUs with over 100 vCPUs [45].

As the number of MPs increases further, components like OB can be scaled out using a distributed architecture: multiple OB instances can be deployed, each responsible for a disjoint subset of RBs (e.g., sharded by trading symbol). These distributed OBs can safely dequeue and batch pending trades, forwarding them to a final merge layer colocated with the matching engine (ME) for order finalization [24].

In parallel, Cuttlefish can benefit from ongoing advances in cloud networking infrastructure. Techniques such as traffic admission control and prioritization [21, 59] can reduce or bound end-to-end latencies. Additionally, scalable multicast engines [26, 47, 18] can help scale CES's market data dissemination, enabling high-rate, low-latency delivery to a larger number of participants.

12 Conclusion

This paper presents Cuttlefish, a fair-by-design execution environment for low-latency algorithmic trading that can run on commercial public clouds. With its virtual time overlay, Cuttlefish abstracts out the variances in the underlying communication and computation hardware, while maintaining low latency and high execution throughput.

References

1 Amazon Web Services. Cryptographic attestation, 2024. URL: https://docs.aws.amazon.com/enclaves/latest/user/set-up-attestation.html.

2 Matteo Aquilina, Eric B Budish, and Peter O'Neill. Quantifying the high-frequency trading" arms race": A simple new methodology and estimates. Technical report, Working Paper, 2020.

3 Vignesh Babu and David Nicol. Precise virtual time advancement for network emulation. In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 175–186, 2020. doi:10.1145/3384441.3395978.

4 Vignesh Babu and David Nicol. Temporally synchronized emulation of devices with simulation of networks. In *Proceedings of the 2022 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 1–12, 2022. doi:10.1145/3518997.3531020.

5 Marco Spaziani Brunella, Giacomo Belocchi, Marco Bonola, Salvatore Pontarelli, Giuseppe Siracusano, Giuseppe Bianchi, Aniello Cammarano, Alessandro Palumbo, Luca Petrucci, and Roberto Bifulco. hXDP: Efficient software packet processing on FPGA NICs. *Communications of the ACM*, 65(8):92–100, 2022. doi:10.1145/3543668.

6 Eric Budish. High-frequency trading and the design of financial markets, 2023. URL: https://www.youtube.com/watch?v=OwQjTedWSUM.

7 Eric Budish, Peter Cramton, and John Shim. The high-frequency trading arms race: Frequent batch auctions as a market design response. *The Quarterly Journal of Economics*, 130(4):1547–1621, 2015.

8 Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, 2017. doi:10.1145/3009824.

9 Gong Chen, Zheng Hu, and Dong Jin. Integrating I/O time to virtual time system for high fidelity container-based network emulation. In Kalyan Perumalla, Margaret Loper, Dong (Kevin) Jin, and Christopher D. Carothers, editors, *SIGSIM-PADS '22: SIGSIM Conference on Principles of Advanced Discrete Simulation, Atlanta, GA, USA, June 8 - 10, 2022*, pages 37–48. ACM, 2022. doi:10.1145/3518997.3531023.

10 Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, page 86, 2016. URL: http://eprint.iacr.org/2016/086.

11 Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013. doi:10.1145/2408776.2408794.

12 Dmitry Duplyakin, Alexandru Uta, Aleksander Maricq, and Robert Ricci. On studying CPU performance of CloudLab hardware. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–2. IEEE, 2019. doi:10.1109/ICNP.2019.8888128.

13 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

14 ewasm. Determining wasm gas costs, 2023. URL: https://github.com/ewasm/design/blob/master/determining_wasm_gas_costs.md.

15 Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking: SmartNICs in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, 2018.

16 Elazar Gershuni, Nadav Amit, Arie Gurfinkel, Nina Narodytska, Jorge A Navas, Noam Rinetzky, Leonid Ryzhyk, and Mooly Sagiv. Simple and precise static analysis of untrusted linux kernel extensions. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1069–1084, 2019. doi:10.1145/3314221.3314590.

17 Ahmad Ghalayini, Jinkun Geng, Vighnesh Sachidananda, Vinay Sriram, Yilong Geng, Balaji Prabhakar, Mendel Rosenblum, and Anirudh Sivaraman. CloudEx: A fair-access financial exchange in the cloud. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 96–103, 2021. doi:10.1145/3458336.3465278.

18 Junzhi Gong, Yuliang Li, Devdeep Ray, KK Yap, and Nandita Dukkipati. Octopus: A fair packet delivery service. *arXiv preprint arXiv:2401.08126*, 2024. doi:10.48550/arXiv.2401.08126.

19 Prateesh Goyal, Ilias Marinos, Eashan Gupta, Chaitanya Bandi, Alan Ross, and Ranveer Chandra. Rethinking cloud-hosted financial exchanges for response time fairness. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 108–114, 2022. doi:10.1145/3563766.3564098.

20 James N Gray. Notes on data base operating systems. *Operating systems: An advanced course*, pages 393–481, 2005.

21 Matthew P Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert NM Watson, Andrew W Moore, Steven Hand, and Jon Crowcroft. Queues don't matter when you can JUMP them! In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 1–14, 2015.

22 CME Group. CME group signs 10-year partnership with Google cloud to transform global derivatives markets through cloud adoption, 2019. URL: https://www.cmegroup.com/media-room/press-releases/2021/11/04/cme_group_signs_10-yearpartnershipwithgooglecloudtotransformglob.html.

23 Diwaker Gupta, Ken Yocum, Marvin McNett, Alex C. Snoeren, Amin Vahdat, and Geoffrey M. Voelker. To infinity and beyond: Time-warped network emulation. In Larry L. Peterson and Timothy Roscoe, editors, *3rd Symposium on Networked Systems Design and Implementation (NSDI 2006), May 8-10, 2007, San Jose, California, USA, Proceedings*. USENIX, 2006. URL: http://www.usenix.org/events/nsdi06/tech/gupta.html.

24 Eashan Gupta, Prateesh Goyal, Ilias Marinos, Chenxingyu Zhao, Radhika Mittal, and Ranveer Chandra. DBO: Fairness for cloud-hosted financial exchanges. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 550–563, 2023. doi:10.1145/3603269.3604871.

25 Mark Handley. Delay is not an option: Low latency routing in space. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, HotNets '18, pages 85–91, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3286062.3286075.

26 Muhammad Haseeb, Jinkun Geng, Ulysses Butler, Xiyu Hao, Daniel Duclos-Cavalcanti, and Anirudh Sivaraman. Poster: Jasper, a scalable and fair multicast for financial exchanges in the cloud. In *Proceedings of the ACM SIGCOMM 2024 Conference: Posters and Demos*, pages 36–38, 2024. doi:10.1145/3672202.3673728.

27 IEX. The cost of exchange services, 2019. URL: https://finansdanmark.dk/media/mstbpq23/iex-and-market-data-cost-2019.pdf.

28 CFA Institute. SpaceX is opening up the next frontier for HFT, 2019. URL: https://blogs.cfainstitute.org/marketintegrity/2019/06/25/spacex-is-opening-up-the-next-frontier-for-hft/.

29 Internet Engineering Task Force (IETF). BPF instruction set specification, v1.0, 2023. URL: https://datatracker.ietf.org/doc/draft-ietf-bpf-isa/.

30 iovisor. uBPF: User space eBPF vm, 2023. URL: https://github.com/iovisor/ubpf.

31 Andrei A. Kirilenko and Andrew W. Lo. Moore's law versus murphy's law: Algorithmic trading and its discontents. *The Journal of Economic Perspectives*, 27(2):51–72, 2013. URL: http://www.jstor.org/stable/23391690.

32 Jereme Lamps, David M Nicol, and Matthew Caesar. Timekeeper: A lightweight virtual time system for linux. In *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 179–186, 2014. doi:10.1145/2601381.2601395.

33 Christian Leber, Benjamin Geib, and Heiner Litz. High frequency trading acceleration using FPGAs. In *2011 21st International Conference on Field Programmable Logic and Applications*, pages 317–322. IEEE, 2011. doi:10.1109/FPL.2011.64.

34 Yiran Lei, Liangcheng Yu, Vincent Liu, and Mingwei Xu. PrintQueue: Performance diagnosis via queue measurement in the data plane. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 516–529, 2022. doi:10.1145/3544216.3544257.

35 Jaime Lizárraga. Increasing competition and improving transparency in U.S. equity markets, 2022. URL: https://www.sec.gov/news/statement/lizarraga-rule-605-20221214.

36 John W Lockwood, Adwait Gupte, Nishit Mehta, Michaela Blott, Tom English, and Kees Vissers. A low-latency library in FPGA hardware for high-frequency trading (HFT). In *2012 IEEE 20th annual symposium on high-performance interconnects*, pages 9–16. IEEE, 2012. doi:10.1109/HOTI.2012.15.

37 Jennifer Lundelius and Nancy A. Lynch. An upper and lower bound for clock synchronization. *Inf. Control.*, 62(2/3):190–204, 1984. doi:10.1016/S0019-9958(84)80033-9.

38 Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. Taming performance variability. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2018. URL: https://www.flux.utah.edu/paper/maricq-osdi18.

39 Vasilios Mavroudis and Hayden Melton. Libra: Fair order-matching for electronic financial exchanges. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 156–168, 2019. doi:10.1145/3318041.3355468.

40 Microsoft. B3 partners with microsoft and oracle for systems migration to the cloud, 2022. URL: https://news.microsoft.com/es-xl/b3-partners-with-microsoft-and-oracle-for-systems-migration-to-the-cloud.

41 Microsoft. Empowering the future of financial markets with London Stock Exchange Group, 2022. URL: https://blogs.microsoft.com/blog/2022/12/11/empowering-the-future-of-financial-markets-with-london-stock-exchange-group/.

42 Microsoft. Machnet, 2023. URL: https://github.com/microsoft/machnet/tree/main.

43 Microsoft. Microsoft azure: Cloud computing services, 2023. URL: https://azure.microsoft.com/en-us/.

44 Microsoft. Azure virtual network peering, 2024. URL: https://learn.microsoft.com/en-us/azure/virtual-network/virtual-network-peering-overview.

45 Microsoft. Sizes for virtual machines in Azure, 2025. Accessed July 2025. URL: https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/overview.

46 Microsoft Azure. Confidential computing, 2024. URL: https://azure.microsoft.com/en-us/solutions/confidential-compute.

47 Andy Myers, Brian Nigito, and Nate Foster. Network design considerations for trading systems. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*, pages 282–289, 2024. doi:10.1145/3696348.3696890.

48 NASDAQ. Nasdaq and aws partner to transform capital markets, 2021. URL: https://www.nasdaq.com/press-release/nasdaq-and-aws-partner-to-transform-capital-markets-2021-12-01.

49 Arista Networks. Arista 7132lb datasheet. URL: https://www.arista.com/assets/data/pdf/Datasheets/7132LB-Datasheet.pdf.

50 Arista Networks. Arista 7135lb datasheet. URL: https://www.arista.com/assets/data/pdf/Datasheets/7135LB-Datasheet.pdf.

51 John Osborne. High-frequency trading over LEO, 2022. URL: https://josborne.ca/high-frequency-trading-over-leo/.

52 Reinhard Schwarz and Friedemann Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed computing*, 7(3):149–174, 1994. doi:10.1007/BF02277859.

53 SeaHorn Project. Seahorn: Extending eBPF verification with static analysis, 2019. URL: https://seahorn.github.io/seahorn/crab/static%20analysis/linux%20extensions/ebpf/2019/07/04/seahorn-ebpf.html.

54 Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. Is big data performance reproducible in modern cloud networks? In *17th USENIX symposium on networked systems design and implementation (NSDI 20)*, pages 513–527, 2020. URL: https://www.usenix.org/conference/nsdi20/presentation/uta.

55 Maarten van Steen and Andrew S. Tanenbaum. *Distributed Systems*. distributed-systems.net, 3 edition, 2017.

56 Wikipedia Contributor. Algorithmic trading, 2024. URL: `https://en.wikipedia.org/wiki/Algorithmic_trading`.

57 Liangcheng Yu, Prateesh Goyal, Ilias Marinos, and Vincent Liu. Cuttlefish: A fair, predictable execution environment for cloud-hosted financial exchanges, November 2024. URL: `https://www.microsoft.com/en-us/research/publication/cuttlefish-a-fair-predictable-execution-environment-for-cloud-hosted-financial-exchanges/`.

58 Fan Zhang, Ittay Eyal, Robert Escriva, Ari Juels, and Robbert Van Renesse. REM: Resource-efficient mining for blockchains. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1427–1444, 2017. URL: `https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/zhang`.

59 Yiwen Zhang, Gautam Kumar, Nandita Dukkipati, Xian Wu, Priyaranjan Jha, Mosharaf Chowdhury, and Amin Vahdat. Aequitas: Admission control for performance-critical RPCs in datacenters. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 1–18. ACM, 2022. `doi:10.1145/3544216.3544271`.

60 Yuxuan Zhao, Dmitry Duplyakin, Robert Ricci, and Alexandru Uta. Cloud performance variability prediction. In *Companion of the ACM/SPEC International Conference on Performance Engineering*, pages 35–40, 2021. `doi:10.1145/3447545.3451182`.

PvpAMM: A Perpetual Market for Unbalanced Long-Short Positions

Zhenhang Shang
Hong Kong University of Science and Technology, Hong Kong SAR, China

Zhenyu Zhao
Hong Kong University of Science and Technology, Hong Kong SAR, China

Kani Chen
Hong Kong University of Science and Technology, Hong Kong SAR, China

Abstract

Perpetual futures – swap contracts without expiration dates – are the most widely traded derivatives in cryptocurrency markets. Traditional perpetual trading relies on order books, which require substantial bilateral liquidity and face challenges in high-volatility environments. In this paper, we introduce pvpAMM, a peer-to-peer perpetual trading protocol based on automated market maker (AMM) principles. The protocol enables efficient settlement of long-short mismatched markets and drives positions toward equilibrium: when the minority leveraged side wins, their returns are amplified compared to conventional perpetual contracts, while the opposite occurs when the majority side prevails. We also propose arbitrage mechanisms to maintain economic equilibrium within the pvpAMM system. By incorporating liquidity providers (LPs), the protocol aligns more closely with traditional order book trading. Numerical experiments validate our theoretical findings.

2012 ACM Subject Classification Applied computing → Digital cash; Mathematics of computing → Stochastic processes

Keywords and phrases Perpetuals, Decentralized Finance, Auto Market Making, Blockchain

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.34

Funding We acknowledge the support from the Hong Kong RGC through GRF Grants 16310222, 16308221 and 16308421.

1 Introduction

1.1 Background

Perpetual contracts have emerged as the cornerstone of cryptocurrency derivatives trading, enabling leveraged speculation and hedging without requiring holders to roll over the underlying asset. Dominating digital asset markets with daily volumes exceeding $100 billion, they surpass traditional futures in both liquidity and accessibility [3]. However, their unique structure – continuous funding payments, no expiration, and 24/7 settlement – introduces critical challenges for maintaining balanced markets during extreme volatility or unilateral trading pressure.

Traditional implementations rely on centralized limit order books (CLOBs), where execution requires matching long and short order volumes. While straightforward, this mechanism proves fragile in skewed markets: liquidity providers face asymmetric risks that widen spreads, increase fees, and may cause market failure [5]. The zero-sum nature of derivatives exacerbates these issues, as directional exposure imbalances cannot be resolved without substantial trader costs.

We present a paradigm-shifting solution through the first complete **automated market maker (AMM) framework specifically engineered for perpetual futures**, which systematically addresses the persistent position imbalance problem while simultaneously

© Zhenhang Shang, Zhenyu Zhao, and Kani Chen;
licensed under Creative Commons License CC-BY 4.0

enhancing capital efficiency and trader profitability. Unlike traditional AMM designs originally conceived for prediction markets [6] or simple spot trading [9, 12], our protocol introduces **leveraged, state-aware liquidity provisioning** that dynamically adjusts pricing curves and exposure limits in response to real-time market conditions. This innovative approach enables the autonomous absorption of unilateral order flow without dependence on external arbitrageurs or centralized liquidity pools, representing a fundamental advancement in decentralized finance infrastructure. Our solution finally bridges the critical architectural gap that has forced perpetual contracts to remain tethered to centralized order book models despite their inherent incompatibility with DeFi's core principles.

Our research makes three foundational contributions to decentralized derivatives markets:

- A leveraged trading protocol resolving position imbalances, with profitability comparisons against traditional order books.
- Novel arbitrage mechanisms that sustain economic equilibrium, detailing liquidity providers' role in maintaining protocol stability.
- Comprehensive agent-based simulations evaluating protocol efficacy through diverse participant interactions.

1.2 Related Work

Perpetual contracts were first suggested by Shiller in [24] as a means of hedging illiquid assets, and they later gained widespread popularity as a method for taking leveraged positions in cryptocurrency markets [11]. The existing literature on perpetual futures primarily focuses on descriptive evidence. For instance, Alexander et al. [3] found that BitMEX derivatives lead the price discovery process across major Bitcoin spot exchanges. Hung et al. [23] identified significant pricing effects and breakpoints in market efficiency. Additionally, some researchers have addressed the challenges of perpetual trading by proposing derivatives with alternative return structures. For example, D. White proposed a perpetual contract with returns proportional to some power of the spot price in [27], leading to the creation of a decentralized perpetual protocol on the Ethereum blockchain called Squeeth [21]. However, this protocol splits total liquidity and does not resolve the issue of unequal positions.

AMMs have been utilized in various applications within the DeFi ecosystem, primarily popularized by token swap protocols [2, 4, 13]. They have also been adapted for several DeFi applications, including crypto options (e.g., Hegic [28]), rate swaps (e.g., Voltz [16]), and NFT exchanges (e.g., Caviar [10]). There is emerging research on the economic implications of AMMs; however, much of it focuses on the incentives for liquidity provision, particularly the relationship between transaction fees and impermanent loss [15, 17, 8].

In the context of derivatives, a **virtual AMM (vAMM)** model has been implemented, which differs slightly from the conventional AMM. In a vAMM, there is no proper asset pool that supports the counterparty risk, resulting in under-collateralization. Due to this under-collateralization, many vAMMs maintain an insurance pool to cover potential losses. However, this can lead to adverse selection by traders when liquidations do not occur in a timely manner. Some researchers have even argued that perpetual contracts designed through vAMMs resemble a Ponzi scheme that cannot be sustained [2]. In [14], GMX X4 introduced a peer vs. peer (pvp) mechanism for executing perpetual contract transactions via AMMs. LionDex later referenced this design in [18], but their analysis was restricted to narrow use cases – primarily liquidity pool initialization and single-price updates, without accounting for multi-period price dynamics. Their model also suffers from mathematical inconsistencies, particularly in modeling the effect of leverage on returns (see Section 2.3 for a detailed critique).

2 Preliminaries

2.1 Notations

Let P_t denote the spot price of the underlying asset at time t. Contract positions are indexed by i, j, k. For position j, let t_j denote its creation time and T_j its clearing time, with the ordering $t_1 \le t_2 \le t_3 \le \dots$ assumed without loss of generality. The collateral for the position j is $m_j > 0$, and the leverage is b_j, where $b_j > 0$ indicates a long position, $b_j < 0$ a short position, and $b_j = 0$ designates a liquidity provider(LP) role.

At time t_j, the user borrows an amount $m_j|b_j|$ to acquire $n_j = m_j b_j / P_{t_j}$ units of the underlying asset. Ignoring liquidations, the conventional value of position j at time t, where $t_j \le t < T_j$, is given by:

$$w_j^{rough}(t) = \underbrace{m_j}_{collateral} \quad \underbrace{- m_j b_j}_{cash\ borrow} \quad \underbrace{+ n_j P_t}_{spot\ value} = m_j \cdot (1 - b_j + b_j \frac{P_t}{P_{t_j}}).$$

Position j will be *liquidated* if $w_j^{rough}(t)$ falls below a specified threshold. Specifically, the bust time for j is defined as:

$$\tau_j = \inf\{t \in [t_j, T_j] : 1 - b_j + b_j \frac{P_t}{P_{t_j}} \le \epsilon\},$$

where inf denotes the infimum, and $\epsilon \in [0, 1)$ is typically set to 0. When the set is empty, we adopt the convention $\inf \varnothing = \infty$, which implies no liquidation occurs before T_j. Let $I(\cdot)$ be the indicator function, with $I(x) = 1$ if x is true and $I(x) = 0$ otherwise. The conventional value of the j-th perpetual contract over $[t_j, T_j]$ is:

$$w_j(t) = w_j^{rough}(t) \cdot I(\tau_j \ge t) = m_j \cdot (1 - b_j + b_j \frac{P_t}{P_{t_j}}) \cdot I(\tau_j \ge t),$$

Note that τ_j depends only on b_j and $\{P_s : s \ge t_j\}$, so the process $\{w_j(t) : t \ge t_j\}$ is adapted to the filtration generated by $(t_j, m_j, b_j, \{P_s : s \le t\})$.

2.2 pvpAMM Dynamics

In an AMM-based protocol, a liquidity pool serves as the sole counterparty for each transaction, utilizing a conservation function to algorithmically price positions and restrict price movements to predefined trajectories [29]. A perpetual AMM protocol typically involves two types of interaction mechanisms: (1) the establishment, clearing, and liquidation of leveraged positions; (2) the provision and withdrawal of liquidity. These interactions must be specified in a way such that desired invariant properties are upheld.

To illustrate the problem addressed by the pvpAMM protocol, we consider the simplest scenario where all positions remain active post-creation. Similar analysis in more complex scenarios leads to analogous conclusions (see B.2). For time t where:

$$t_1, t_2, t_3, \dots \le t \le T_1, T_2, T_3, \dots \tau_1, \tau_2, \tau_3 \dots$$

the total assets in the liquidity pool are:

$$m_{\text{pool}}(t) = \sum_{j : t_j \le t \le T_j} m_j,$$

and the total value of all positions is:

$$w_{\text{pool}}(t) = \sum_{j:t_j \le t \le T_j} w_j(t).$$

The primary challenge for pvpAMM is to distribute the available collateral $m_{\text{pool}}(t)$ to individual position values $w_j^{\text{pvp}}(t)$ when $m_{\text{pool}}(t) \neq w_{\text{pool}}(t)$, preserving protocol solvency and economic incentives. A straightforward method is to distribute funds in proportion to their conventional worth:

$$w_j^{natural}(t) = \frac{w_j(t)}{w_{pool}(t)} \cdot m_{pool}(t).$$

▶ **Proposition 1.** $\lim_{t \to t_j^+} \left(w_j^{natural}(t) - m_j\right) < 0$ *if* $w_{pool}(t_j) > m_{pool}(t_j)$*, reducing the incentive to create new positions.*

▶ **Proposition 2.** $\lim_{t \to t_j^+} \left(w_j^{natural}(t) - m_j\right) > 0$ *if* $w_{pool}(t_j) < m_{pool}(t_j)$*, leading to a flash-loan attack where an attacker profits by borrowing* m_j *at* t_j *and exiting immediately.*

The proofs of these propositions (Appendix A.1) demonstrate the insufficiency of this natural approach.

2.3 GMX's Solution

To rectify the cash-wealth imbalance, GMX introduced an intermediate token, LPT. Let $\lambda^{gmx}(t)$ represent the price of LPT, and $\mathscr{B}_j(t)$ the LPT balance for position j at time t. The protocol functions as follows:

- *Establishment*: $\mathscr{B}_j(t_j) = m_j/\lambda^{gmx}(t_j)$, effectively purchasing LPT with m_j.
- *Price Updates*: for $\Delta P_t = P_{t+1} - P_t$, $\mathscr{B}_j(t+1) = \mathscr{B}_j(t) \cdot (1 + b_j \Delta P_t / P_t)$.
- *Liquidation*: position j is liquidated if $\mathscr{B}_j(t) \le 0$ at any time $t_j \le t \le T_j$
- *Clearing*: at $t = T_j < \tau_j$, the position holder receives $\mathscr{B}_j(T_j) \cdot \lambda^{gmx}(T_j)$ back.

Some specific numerical examples are given in B.1. Actually we have:

▶ **Proposition 3.** $\mathscr{B}_j(t) \to (m_j/\lambda^{gmx}(t_j)) \left(P_t/P_{t_j}\right)^{b_j}$ *as the time interval for price updates* $\Delta t \to 0$.

The proof is provided in Appendix A.2. In DeFi implementations, spot prices update with each on-chain operation. Consequently, frequent operations compound leverage's effect on LPT balances exponentially, resulting in exponential position value growth.

3 Mechanisms

In this section, we present the mathematical framework governing our pvpAMM protocol. We begin by defining the PLT token system and establishing the fundamental value conservation properties. The dynamics of leveraged positions are then analyzed, with particular attention to the evolution of the scaling parameter $\psi(t)$. Finally, we derive the stochastic differential equation for $\psi(t)$ and examine its probabilistic characteristics. The results provide a complete specification of the core mechanisms of the protocol.

3.1 PLT Token

The defining idea of our pvpAMM is "concentrated collateral and proportional worth": all collateral m_j and worth $w_j(t)$ for leveraged positions is pooled together, and the actual worth $\tilde{w}_j(t)$ of each contract j is realized in proportion to its conventional value. We incorporate a strategy from the GMX model, where m_j is divided by the current (cash)/(position value) scaling factor upon entering the pool. For better understanding, we introduce an intermediate token named PLT(**P**vpAMM **L**iquidity **T**oken), priced at $\lambda^{pvp}(t)$. When a position j is established, an amount of $m_j/\lambda^{pvp}(t_j)$ PLT is recorded. As time progresses, the position's PLT balance is adjusted to $w_j(t)/\lambda^{pvp}(t_j)$ at time t, and its value in the pvpAMM pool is calculated as:

$$\tilde{w}_j(t) = \left(\frac{w_j(t)}{\lambda^{pvp}(t_j)}\right) \cdot \lambda^{pvp}(t).$$

The price of PLT, $\lambda^{pvp}(t)$, is determined by the ratio of total collateral to total PLT balance:

$$\lambda^{pvp}(t) = \frac{\sum_j m_j}{\sum_j \left(w_j(t)/\lambda^{pvp}(t_j)\right)}.$$

$\tilde{w}_j(t)$ measures the amount of numéraire that the j-th contract can retrieve if cleared at time $t_j \le t < \tau_j, T_j$. The PLT price $\lambda^{pvp}(t)$ acts as a normalization factor, ensuring that the total value of all contracts equals the remaining cash amount, the conservation function for our pvpAMM protocol can be written as:

$$\sum_j m_j = \sum_j \left(\frac{w_j(t)}{\lambda^{pvp}(t_j)}\right) \cdot \lambda^{pvp}(t) = \sum_j \tilde{w}_j(t), \quad for\ t > t_j.$$

The PLT mechanism operates similarly to a casino's chip system. Just as a casino maintains a fixed cash reserve backing all chips in circulation, the protocol pools all collateral to back PLT tokens. When traders open positions, their collateral converts to PLTs at the current exchange rate, much like purchasing casino chips with cash. The PLT price fluctuates based on the aggregate performance of all open positions - when collective gains exceed losses, the PLT appreciates, and vice versa. Traders closing positions redeem their PLTs at the prevailing rate, ensuring the system remains fully collateralized at all times. This creates a self-balancing ecosystem where value flows naturally between participants while maintaining systemic solvency.

For computational convenience, we set $\psi(t) = 1/\lambda^{pvp}(t)$, and get:

$$\sum_j m_j = \sum_j \frac{w_j(t)\psi(t_j)}{\psi(t)}, \quad for\ t > t_j. \tag{1}$$

▶ **Proposition 4.** *The scaler function $\psi(t)$ satisfies:*

$$\psi(t) = \frac{\sum_{j:t_j<t\le T_j} w_j(t)\psi(t_j)}{\sum_{j:t_j<t} m_j} \quad for\ t > t_j. \tag{2}$$

By setting the initial value $\psi(0) = 1$, we have one definite and unique solution for $\psi(\cdot)$. It is continuous, and $\psi(t) > 0$ unless no live contracts exist at time t.

Proof. The expression for $\psi(t)$ can be derived directly from equation (1). We assume at any time there is at least one contract live, this can be easily satisfied by setting up a long contract at time 0 with leverage $b \leq 1$, which will never bust. From (2) we know $\psi(\cdot)$ is left-continuous. Since $w_j(t)$ is continuous for $t > t_j$, it is seen from (2) that $\psi(t)$ is continous over t that is not the event times $t_1, t_2, \ ...$

For the position creation time t_k, we may find that

$$\lim_{t \to t_k^+} \psi(t_k) = \frac{\sum_{j:t_j<t_k\leq T_j} w_j(t_k)\psi(t_j) + w_k(t_k)\psi(t_k)}{\sum_{j:t_j<t_k} m_j + m_k} = \frac{\left(\sum_{j:t_j<t_k} m_j\right)\psi(t_k) + w_k(t_k)\psi(t_k)}{\left(\sum_{j:t_j<t_k} m_j\right) + w_k(t_k)}$$
$$= \psi(t_k),$$

the continuity of $\psi(t)$ is thus proved.

To show $\psi(t) > 0$ with at least one live contract at time t, we define $t^* = \inf\{t : \psi(t) = 0\}$, with the convention of $\inf \varnothing = \infty$, then $\psi(t) > 0$ for all $t < t^*$. If t^* is finite, the numerator in (2) will converge to 0 as $t \uparrow t^*$. This implies there are live contracts right before t^*, and they all bust at t^*, which cannot happen as we assumed there is always one live contract at any time. ◀

Note that the calculation of $\psi(t)$ in (2) only involves $\psi(s)$ for $s < t$, indicating an iterative algorithm for computing $\psi(t)$. We can iteratively determine $\psi(t)$ based on all previously computed values of $\psi(\cdot)$ at the event times t_j, T_j that occur before t. Additionally, if there are no events for establishing or clearing positions in the time interval $[a, b]$, the sets $\{j : t_j < a \leq T_j\}$ and $\{j : t_j < b \leq T_j\}$ are identical. Under this condition, we have:

$$\frac{\psi(b)}{\psi(a)} = \frac{\sum_{j:t_j<b\leq T_j} w_j(b)\psi(t_j)}{\sum_{j:t_j<a\leq T_j} w_j(a)\psi(t_j)}. \tag{3}$$

with the continuity of $\psi(\cdot)$, this formula may be used to efficiently compute ψ.

3.2 Leveraged Positions

In this section, we clarify the state transitions of the defined pvpAMM market when operating with leveraged positions. Consider two trading positions, $i = 1$ for Alice and $j = 2$ for Bob. To demonstrate the system's behavior across different stages, we specifically consider a scenario where $t_1 < t_2 < T_1 < T_2 < \tau_1, \tau_2$. The trading activities in our protocol function as follows:

- *Initialization.* we initialize $\psi(t) = 1$ for $t \leq t_1$.
- *Establishment.* for $t_1 < t \leq t_2$,

  $$\psi(t) = \frac{w_1(t)\psi(t_1)}{m_1} = \frac{w_1(t)}{m_1}, \quad \tilde{w}_1(t) = \frac{w_1(t)}{\psi(t)} \equiv m_1.$$

  Alice can close her position at any time during this period to reclaim her collateral m_1, it occurs without any profit or loss, as there is no counterparty involved.

  For $t_2 < t \leq T_1$:

  $$\psi(t) = \frac{w_1(t)\psi(t_1) + w_2(t)\psi(t_2)}{m_1 + m_2} = \frac{w_1(t) + w_2(t)(w_1(t_2)/m_1)}{m_1 + m_2}.$$

  $$\begin{cases} \tilde{w}_1(t) = w_1(t)\psi(t_1)/\psi(t) = w_1(t)\frac{m_1+m_2}{w_1(t)+w_2(t)(w_1(t_2)/m_1)} \\ \tilde{w}_2(t) = w_2(t)\psi(t_2)/\psi(t) = w_2(t)(w_1(t_2)/m_1)\frac{m_1+m_2}{w_1(t)+w_2(t)(w_1(t_2)/m_1)} \end{cases}$$

  where $\tilde{w}_1(t)$ and $\tilde{w}_2(t)$ represent the amounts each position would receive if closed at time t, $\tilde{w}_1(t) + \tilde{w}_2(t) \equiv m_1 + m_2$.

- *Clearing.* Alice closed her position at time T_1, withdrew $\tilde{w}_1(T_1) = w_1(T_1)/\psi(T_1)$ from the pool. for $T_1 < t \leq T_2$:

$$\psi(t) = \frac{w_2(t)\psi(t_2)}{m_1 + m_2 - w_1(T_1)\psi(t_1)/\psi(T_1)},\ \tilde{w}_2(t) = w_2(t)\psi(t_2)/\psi(t) \equiv m_1 + m_2 - \tilde{w}_1(T_1).$$

The above discussion illustrates how $\psi(\cdot)$ is updated and its role in the trading process. For any leveraged position, the presence of a counterparty is not strictly necessary. Even with only positions in the same direction, such as all long positions, trading can still proceed. In this case, the less leveraged positions effectively act as short positions("relatively short").

Numerical examples can be found in Appendix B.2.

3.3 Pricing Equation

For the interval $t_j < t < s < \tau_j$, we have:

$$\frac{\tilde{w}_j(s)}{\tilde{w}_j(t)} = \frac{w_j(s)\psi(t_j)/\psi(s)}{w_j(t)\psi(t_j)/\psi(t)} = \left(\frac{\psi(t)}{\psi(s)}\right)\frac{w_j(s)}{w_j(t)}.$$

This suggests that, from the traders' perspective, the ratio $\psi(t)/\psi(s)$ determines whether their profit(or loss) is greater or less than those in a conventional perpetual setup.

For notational simplicity, let

$$\xi(t) = \frac{\sum_{j:t_j<t\leq T_j} P_t m_j b_j \psi(t_j)/P_{t_j} I(\tau_j \geq t)}{\sum_{j:t_j<t\leq T_j} w_j(t)\psi(t_j)} = \frac{\sum_{j:t_j<t\leq T_j} P_t n_j \psi(t_j)}{\sum_{j:t_j<t\leq T_j} w_j(t)\psi(t_j)}. \tag{4}$$

Observe that $\xi(t)$ is left-continuous, it represents the balance of long and short positions in the pool: $\text{sign}\left(\xi(t)\right) = \text{sign}\left(\sum_{j:t_j<t<T_j} n_j(t)\psi(t_j)\right)$. $\xi(t) > 0$ means that the weighted long shares, adjusted by $\psi(t_j)$, are heavier than the weighted short shares, and vice versus.

▶ **Theorem 5.** *The scalar function $\psi(t)$ satisfies the stochastic differential equation:*

$$\frac{\mathrm{d}\psi(t)}{\psi(t)} = \frac{\xi(t)}{P_t}\mathrm{d}P_t. \tag{5}$$

Suppose the price process satisfies

$$\frac{\mathrm{d}P_t}{P_t} = \mu(t)\mathrm{d}t + \sigma(t)\mathrm{d}B_t$$

with B_t a standard Brownian motion, and $\mu(t)$, $\sigma(t)$ continuous functions. If $\mu(t) \equiv 0$, namely, the price process is a martingale, then $\psi(t)$ is a positive martingale. In this case, $\mathbb{E}[\psi(t)] = 1$ for all t, and $\psi(t)$ converges almost surely as $t \to \infty$.

Proof. Equation (5) directly follows from differentiating both sides of equation (2). If P_t follows geometric Brownian motion, we have

$$\psi(t) = \exp\left\{\int_0^t \xi(s)[\sigma(s)\mathrm{d}B_s + \mu(s)\mathrm{d}s] - \frac{1}{2}\int_0^t [\xi(s)\sigma(s)]^2\mathrm{d}s\right\}$$

when $\mu(t) \equiv 0$, $\mathrm{d}\psi(t) = \psi(t)\xi(t)\sigma(t)\mathrm{d}B_t$, $\psi(t)$ is a local martingale, given that $\psi(t)$ is positive, it is also a true martingale by the positive martingale property [19]. Then, $\mathbb{E}[\psi(t)] = \psi(0) = 1$, $\psi(t)$ is bounded in expectation. By the Martingale Convergence Theorem, $\psi(t)$ converges almost surely to some limit as $t \to \infty$. ◀

Suppose $\xi(t) > 0$, i.e. the weighted long positions are heavier than short positions, as seen from an interpretation of equation (5), $\psi(t)$ increases (or decreases) along with the increase (or decrease) of P_t. If $P_s > P_t$, $\psi(t)$ increases, and all long positions, the majority of the pool, get their profit from the difference $(P_s - P_t)$. Note that the increase in $\psi(t)$ suggests a diminished return compared to the conventional perpetual. Specifically, if Alice establishes the same position in a conventional setup and Bob does so in our protocol, Bob's instantaneous return will be lower than Alice's.

Conversely, if the pool (majority) is incorrect and P_t decreases, $\psi(t)$ also decreases. The decrease in $\psi(t)$ implies a better return compared with a conventional perpetual. The same interpretation holds for the case where $\xi(t) < 0$.

In summary, our protocol favors the overall performance of the pool. If the pool is correct, the established contracts yield worse returns than those in conventional perpetuals. If the pool is incorrect, the contracts yield better returns. For an individual position, the best return occurs when its direction is correct while the pool is wrong, whereas the worst return happens when the individual contract direction is incorrect but the pool is right.

4 Security Analysis

Security is paramount in DeFi protocols, particularly for innovative systems like pvpAMM. This section examines the security of our protocol from four aspects: (1) arbitrage maintains market equilibrium through price-stabilizing flows, (2) liquidity providers ensure stability via deposit/withdrawal dynamics, (3) flash loan resistance emerges from bounded profit potential, and (4) oracle reliability depends on feed accuracy and update frequency. These properties collectively ensure robustness against smart contract exploits, strategic manipulations, and data integrity threats.

4.1 Arbitrage Mechanisms for System Balance

In traditional perpetual contract trading markets such as centralized exchanges, prices of perpetual contracts are determined by secondary market transactions, and quantitative analysis between the spot and perpetual markets can identify arbitrage-free prices [1]. In contrast, our pvpAMM model determines the value of a position primarily through the price of PLT, denoted as $\psi(t)$. This value influences the strategies for position entry and exit, and arbitrageurs effectively trade over$\psi(t)$. Unlike in CEX, $\psi(t)$ in pvpAMM is unaffected by secondary market transactions or market tendencies towards long or short positions; the determination of $\psi(t)$ follows exclusively from the stochastic differential equation defined in Section 5. The pvpAMM protocol introduces an arbitrage mechanism that attracts minority orders, automatically balancing the long-short ratio by driving $\xi(t)$, as defined in 4, toward zero.

▶ **Theorem 6.** *Arbitrage opportunities exist when* $\xi(t) \neq 0$.

Proof. For simplicity, assume $P_t = 1$, and let $s = t + \Delta t$, where Δt is small. At time t, user j borrows $(\xi(t) + 2)$ dollars to: (1) purchase $\xi(t)$ spot assets, (2) use 1 dollar as collateral to open a position in the conventional market with leverage $(-b)$, and (3) use another 1 dollar as collateral to open a position in our pvpAMM with leverage b. The arbitrage behavior here occurs in a short period of time, so the funding fee in the conventional perpetual market is negligible. The portfolio consists of the following four components:

- Conventional perpetuals: $CP(s) = 1 + b - bP_s$
- pvpAMM perpetuals: $PVP(s) = (1 - b + bP_s)\,(\psi(t)/\psi(s))$

- Spot: $SP(s) = P_s\xi(t)$
- Cash: $CASH(s) = -(\xi(t) + 2)$

From (5) and $P_t = 1$, we have:

$$\frac{\psi(s) - \psi(t)}{\psi(s)} = \xi(t)\Delta P_t - \xi(t)^2(\Delta P_t)^2 + o((\Delta P_t)^2).$$

The complete derivation appears in Appendix A.4. Note that $portfolio(t) = 0$, and the portfolio value at time s is given by:

$$\begin{aligned} portfolio(s) &= CP(s) + PVP(s) + SP(s) + CASH(s) \\ &= [1 + b - bP_s] + \left[(1 - b + bP_s)\left(\frac{\psi(t)}{\psi(s)}\right)\right] + [P_s\xi(t)] + [-(\xi(t) + 2)] \\ &= -[b(P_s - 1) + 1]\left(\frac{\psi(s) - \psi(t)}{\psi(s)}\right) + [P_s\xi(t) + 2] - (\xi(t) + 2) \\ &= -(b\Delta P_t + 1)\left(\xi(t)\Delta P_t - (\xi(t)\Delta P_t)^2 + o((\Delta P_t)^2)\right) \\ &\qquad + (P_s\xi(t) + 2) - (\xi(t) + 2) \\ &= \xi(t)(\xi(t) - b)\left((\Delta P_t)^2 + o((\Delta P_t)^2)\right). \end{aligned}$$

The portfolio is positive when arbitrageur set:

$$b \begin{cases} < \xi(t), & \text{if } \xi(t) > 0 \\ > \xi(t), & \text{if } \xi(t) < 0 \end{cases}$$

When $\xi(t) > 0$, open a position with leverage $b < \xi(t)$ implies relatively short position in pvpAMM, thereby driving $\xi(t)$ downward. Vice versa when $\xi(t) < 0$, open a position with leverage $b > \xi(t)$ implies relatively long position in pvpAMM, thereby driving $\xi(t)$ upward. The arbitrage opportunities exist when $\xi(t) \neq 0$. ◀

4.2 Liquidity Provision and Market Stability

Liquidity provision is a key mechanism for enhancing stability in decentralized finance (DeFi) systems – a critical factor in ensuring protocol security. In pvpAMM, liquidity providers act as counterweights to dominant pool positions, mirroring the stabilizing role of market makers in traditional perpetual markets.

By letting $b_j = 0$ for the j-th contract, we get a liquidity position:

$$w_j(t) = m_j, \quad \tilde{w}_j(t) = w_j(t)\frac{\psi(t_j)}{\psi(t)} = m_j\frac{\psi(t_j)}{\psi(t)}.$$

this position will profit from a decrease in $\psi(t)$. Under the condition that $\xi(t) > 0$, a decline in $\psi(t)$ indicates a fall in P_t, meaning the liquidity position is opposite to the direction of the majority of the pool's positions. This is precisely the role of the LP, as it partially acts as a counterparty to the pool majority. Let $b_1 = 0$, i.e. the first position be an LP position, actually, we may regard the first LP position establishment point t_k as the process origin. We have:

▶ **Theorem 7.** *Liquidity position makes the worth of our pvpAMM position closer to a conventional perpetual. Specially,*

$$\lim_{m_1 \to \infty} \frac{\tilde{w}_j(t)}{w_j(t)} = 1, \ \forall\ j \neq 1, t_1 < t_j < t < T_j < T_1.$$

Proof. for $t_1 < t < T_1$,

$$\lim_{m_1\to\infty} \psi(t) = \lim_{m_1\to\infty} \frac{m_1 + \sum_{j\neq 1: t_j < t \leq T_j} w_j(t_j)\psi(t_j)}{m_1 + \sum_{j\neq 1: t_j<t} m_j - \sum_{j:T_j<t} w_j(T_j)\psi(t_j)/\psi(T_j)} = 1,$$

then

$$\lim_{m_1\to\infty} \frac{\tilde{w}_j(t)}{w_j(t)} = \lim_{m_1\to\infty} \frac{\psi(t_j)}{\psi(t)} = \frac{\lim_{m_1\to\infty}\psi(t_j)}{\lim_{m_1\to\infty}\psi(t)} = 1. \quad \blacktriangleleft$$

Generally, traders interacting with the pool for leveraged positions have to reimburse LPs for supplying assets and living with the volitality of $\psi(t)$. This compensation comes in the form of swap fees that are charged on each position, and then distributed to liquidity pool shareholders [20]. Let δ be a small percentage of the trading fee, the pvpAMM protocol with fees is designed as follows:

$$\begin{aligned} \tilde{w}_j(t) &= (1-\delta) I(b_j \neq 0) w_j(t) \frac{\psi(t_j)}{\psi(t)}, \\ \psi(t) &= \frac{\sum_{j: t_j < t \leq T_j} w_j(t)\psi(t_j)}{\sum_{j:t_j<t} m_j - \sum_{j:T_j<t}(1-\delta I(b_j\neq 0)) w_j(T_j)\psi(t_j)/\psi(T_j)}. \end{aligned} \tag{8}$$

4.3 Resistance to Flash Loan Attacks

Flash loan attacks – a prevalent threat in DeFi, as outlined in Proposition 2 – leverage rapid, uncollateralized borrowing to artificially distort market conditions. While earlier pvpAMM iterations were susceptible to such exploits, our protocol eliminates this vulnerability by preventing unanticipated profits from immediate position entry and exit, as formalized below.

▶ **Theorem 8.** *The proposed pvpAMM protocol as 3.1 is resistant to flash loan attacks, i.e., there is no unanticipated profit from an immediate exit after entry.*

Proof. This follows directly from the continuity of $\psi(t)$ 2: $\lim_{t\to t_j^+}\psi(t) = \psi(t_j)$, thus

$$\lim_{t\to t_j^+} \tilde{w}_j(t) = \lim_{t\to t_j^+} w_j(t)\frac{\psi(t_j)}{\psi(t)} = \left(\lim_{t\to t_j^+} w_j(t)\right)\left(\lim_{t\to t_j^+}\frac{\psi(t_j)}{\psi(t)}\right) = m_j. \quad \blacktriangleleft$$

4.4 Oracle Price Feed Security

In decentralized finance (DeFi), accurate and timely price updates are critical to the functionality of the protocol, particularly for perpetual decentralized exchanges such as GMX and Jupiter. These platforms depend on oracles to provide asset price data, which underpins position valuation and overall protocol stability. However, oracles introduce potential attack vectors, including off-chain data manipulation, update latency, and centralization risks, all of which may compromise system integrity [26].

To mitigate these risks, modern DeFi protocols employ advanced oracle mechanisms. For example, Jupiter Perp integrates the Signal Oracle, a collaborative solution with Jupiter that enables compute-efficient multi-asset price updates in a single transaction, ensuring low latency and high reliability. Additionally, it supplements security with the Pyth Oracle for redundancy [25].

Decentralized oracle networks like Chainlink further enhance robustness by aggregating data from multiple sources, reducing manipulation risks. These systems use cryptographic attestations to verify data providers and employ aggregation methods to derive consensus prices – often accompanied by confidence intervals to indicate reliability [7].

5 Simulation

To validate the properties of our proposed pvpAMM protocol, we conducted numerical simulations. These simulations define a simulated underlying price process and explore how different types of agents interact with the protocol. The simulation runs for a predetermined number of time steps, updating prices and evaluating policies for introducing new agents at each step.

5.1 Market and Agents

The underlying market price is updated at each time step (after all agents have completed their actions) using the formula:

$$P_t \to P_t \cdot e^{\sigma X + \mu}$$

where $X \sim \mathcal{N}(0,1)$ is drawn from a normal distribution, and $\mu, \sigma \in \mathbb{R}$ represent the mean returns and market volatility. In our simulation, we set $\mu = 0$ and $\sigma = 0.01$. The initial price P_0 is set to 1.0, with the timestep defined as $\mathrm{d}t = \frac{1}{100000}$.

We analyze the performance of positions in a simple market with three different types of users, where the leverage factor b_j and the collateral m_j are defined as in Section 2.1. The users and their default values are set to be:

- Alice, who joins at $t_1 = 1$, uses no leverage, i.e. $b_1 = 0$, with a $m_1 = 1000$ collateral.
- Bob, who joins at $t_2 = 1000$, uses $3\times$ leverage long, i.e. $b_2 = 3$, with a $m_2 = 1500$ collateral.
- Chris, who joins at $t_3 = 2000$, uses $2\times$ leverage short, i.e. $b_3 = -2$, with a $m_3 = 1000$ collateral.

We assume that price fluctuations are relatively small and users have a low leverage ratio, ensuring no position will be liquidated. The variables $\psi(t), w_j(t), \tilde{w}_j(t)$ for $j = 1, 2, 3$ are updated through equations 2, 2.1, 3.1, respectively.

5.2 Results

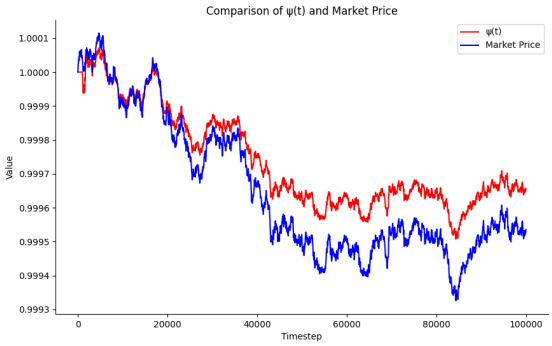


Figure 1 $\psi(t)$ and P_t are derived by averaging data from 500 experimental sets. It can be seen that $\psi(t)$ is steady and exhibits lower volatility compared to P_t.

To investigate the property of $\psi(t)$, multiple experiments with different price series P_t are conducted, we calculate $\psi(t)$ and take their mean as our final value. The results, shown in Figure 1, indicate that $\psi(t)$ consistently fluctuates around its expected value of 1.0, with lower volatility compared to P_t.

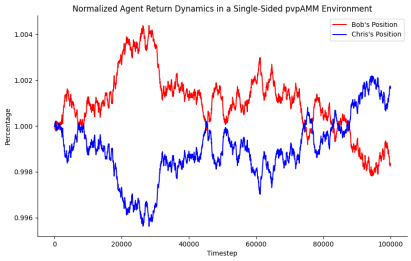
The unique properties of our proposed pvp model under various market conditions are experimentally verified as follows:

5.2.1 No Counterparty Needed

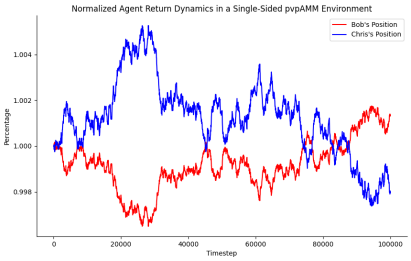
Extreme market conditions often exhibit a clear one-sided sentiment, such as a bull market where all participants expect prices to rise, or a bear market where everyone anticipates a decline. In such scenarios, market makers are not necessary in our model to allow trader activities. For instance, in a bullish market where all traders anticipate an upward movement, their expectation of rising prices is reflected in the leverage of their trades: the stronger their belief in the price increase, the higher their leverage will be.

For example, if Bob takes a $b_2 = 3\times$ long position, while Chris opts for a $b_3 = 2\times$ long position, both entering at $t_2 = t_3 = 1$ with collateral amounts of $m_2 = m_3 = 1000$. Results shown in Figure 2a indicate that when the price rises, Bob achieves a positive return while Chris incurs a loss. Conversely, if the price drops – indicating a misjudgment by all traders, the outcomes reverse. Despite both being long positions, Chris is effectively at a disadvantage compared to Bob, whose smaller position yields returns akin to being short.

To further illustrate the impact of leverage, we conducted additional experiments where $b_2 = 2, b_3 = 3, m_2 = 3000$, and $m_3 = 2000$, ensuring that $m_2 \cdot b_2 = m_3 \cdot b_3$. Even under these conditions, Bob's position beats Chris's when the market price increases, and vice versa when price drops, see 2b. This reinforces the conclusion that leverage is the critical factor: for long positions, less leveraged traders effectively act as relative short against more leveraged traders. A similar conclusion applies to short positions.



(a) Long positions with different contract positions.



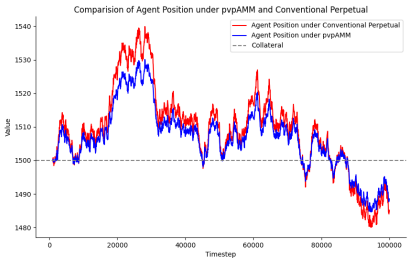
(b) Long positions with same positions but different leverage.

Figure 2 Normalized agent return dynamics in a single-sided pvpAMM environment. The left figure depicts return dynamics for varying contract values, where $m_2 \cdot b_2 > m_3 \cdot b_3$, $b_2 \cdot b_3 > 0$, the right figure shows dynamics for equal contract values but differing leverage, with $m_2 \cdot b_2 = m_3 \cdot b_3$, and $b_3 > b_2 > 0$. This demonstrates that the agent with higher leverage and a contract position aligned with the market direction achieves greater profits.

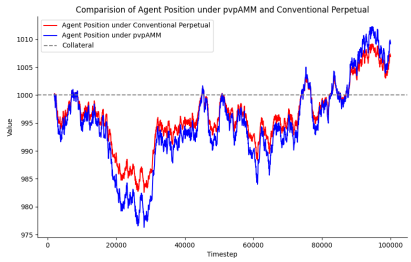
5.2.2 Auto Rebalancing in Unbalanced Positions

In a normal market where users hold differing directional views, the pvp protocol naturally encourages a balance between the long and short sides through economic incentives. It offers higher returns for the minority side while providing the majority side with lower but more

stable returns. Figure 3 illustrates the dynamics of agent positions in this context. Bob, representing the minority, achieves a return that exceeds the conventional return when the price series P_t trends upwards. Conversely, Chris, as the majority, incurs a negative return, however, this loss is less severe than what would be experienced in a conventional perpetual market.



(a) Position values for Bob, the majority party.



(b) Position values for Chris, the minority party.

Figure 3 Comparison of pvpAMM position value $\tilde{w}_j(t)$ (in blue) and traditional perpetual position value $w_j(t)$ (in red), with the horizontal dashed line representing the initial collateral amount. Bob represents the majority party in the protocol, while Chris is the minority party. It is evident that when Bob wins, i.e., when the curves are above the dashed line, $w_1(t) > \tilde{w}_1(t)$, indicating that the return on the pvp position is lower than the traditional return. Conversely, for Chris, $w_2(t) < \tilde{w}_2(t)$. This demonstrates that the pvp protocol can maintain equilibrium between long and short positions through economic incentives.

For liquidity providers in the protocol, Figure 4 illustrates the impact of the LP's principal on $\tilde{w}_{Bob}$ at $t = 50000$. As the LP position m_1 increases, Bob's returns in the pvp protocol, $\tilde{w}_2(t)$, gradually converge to those of the traditional perpetual contract $w_2(t)$. This observation aligns with our derivations presented in 4.2.

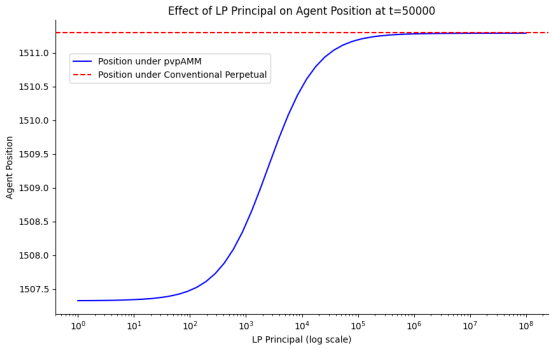


Figure 4 $t = 50000$, effect of LP principal m_1 on agent Bob's position value $\tilde{w}_2(50000)$. As the LP principal m_1 increases, $\tilde{w}_2(50000)$ converges to the conventional perpetual worth $w_2(50000)$.

6 Conclusion

We have detailed the design and implementation of pvpAMM, a peer-to-peer automated market maker (AMM) for perpetual contract transactions. The system proves particularly effective in markets with asymmetrical long and short positions, enabling a robust secondary

market without traditional market makers or liquidity providers. The elimination of liquidity providers allows pvpAMM to extend derivative trading into smaller or markedly skewed market conditions, offering customized trading opportunities. Additionally, pvpAMM applies to prediction markets, where it can utilize trading behaviors to extract price insights and forecasts.

In deploying pvpAMM via blockchain smart contracts, careful attention is required for price determination to ensure the smooth handling of leveraged positions. The industry often adopts a multi-source oracle strategy to ensure reliability and accuracy, as elaborated in [30]. We include Solidity-based code for the critical modules of pvpAMM here, such as position opening, closing, and liquidation, to serve as a practical reference for readers [22].

Perpetual contracts are pivotal in finance, particularly within the cryptocurrency sector, where they traditionally depend on order books or liquidity pools with liquidity providers as counterparties. Our pvpAMM design addresses this reliance by introducing a new asset type, PLT. Analyzing PLT's behavior under various spot price processes are left as future work.

References

1 Damien Ackerer, Julien Hugonnier, and Urban Jermann. Perpetual futures pricing. Technical report, National Bureau of Economic Research, 2024.

2 Hayden Adams and Noah Zinsmeister. Moody salem moody, uniswaporg river keefer, and dan robinson. uniswap v3 core. Technical report, Uniswap Foundation, 2021.

3 Carol Alexander, Jaehyuk Choi, Heungju Park, and Sungbin Sohn. Bitmex bitcoin derivatives: Price discovery, informational efficiency, and hedging effectiveness. *Journal of Futures Markets*, 40(1):23–43, 2020.

4 Guillermo Angeris and Tarun Chitra. Improved price oracles: Constant function market makers. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 80–91, 2020. doi:10.1145/3419614.3423251.

5 Guillermo Angeris, Tarun Chitra, Alex Evans, and Matthew Lorig. A primer on perpetuals. *SIAM Journal on Financial Mathematics*, 14(1):SC17–SC30, 2023.

6 Dmitriy Berenzon. Constant function market makers: Defi's "zero to one" innovation. URL: https://medium.com/bollinger-investment-group/constant-function-market-makersdefis-zero-to-one-innovation-968f77022159, 2020. Accessed: 2025-08-08.

7 Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, Sergey Nazarov, Alexandru Topliceanu, Florian Tramèr, and Fan Zhang. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks, 2021. URL: https://research.chain.link/whitepaper-v2.pdf.

8 Agostino Capponi and Ruizhe Jia. The adoption of blockchain-based decentralized exchanges. *arXiv preprint arXiv:2103.08842*, 2021.

9 Athos VC Carvalho, Douglas Silveira, Regis A Ely, and Daniel O Cajueiro. A logarithmic market scoring rule agent-based model to evaluate prediction markets. *Journal of Evolutionary Economics*, 33(4):1303–1343, 2023.

10 Caviar. Caviar. https://www.caviar.sh/, 2023. Accessed: 2025-08-08.

11 CoinGecko. Cryptocurrency derivatives (perpetual contract) by volume. https://assets.coingecko.com/reports/2022/CoinGecko-2022-Annual-Crypto-Industry-Report.pdf, 2022. Annual Crypto Industry Report. Accessed: 2025-08-08.

12 Richard Dewey and Craig Newbold. The pricing and hedging of constant function market makers. *arXiv preprint arXiv:2306.11580*, 2023. doi:10.48550/arXiv.2306.11580.

13 Michael Egorov. Stableswap-efficient mechanism for stablecoin liquidity. *Retrieved Feb*, 24:2021, 2019.

14 GMX. Gmx. https://gmxio.substack.com/p/x4-protocol-controlled-exchange, 2022. Accessed: 2025-08-08.

15 Joel Hasbrouck, Thomas J Rivera, and Fahad Saleh. An economic model of a decentralized exchange with concentrated liquidity. *Available at SSRN 4529513*, 2023.

16 Simon Jones and Artur Begyan. Voltz v2: A non-custodial clearinghouse. Technical report, Voltz, April 2023. Working Draft. URL: https://assets.website-files.com/61389177f5eab433827f7185/64353ecfebb1137bea5355d7_Voltz_v2_11Apr23.pdf.

17 Alfred Lehar and Christine A Parlour. Decentralized exchange: The uniswap automated market maker. *Available at SSRN 3905316*, 2021.

18 LionDEX. Liondex. https://liondex-official.gitbook.io/liondex/pvp-amm-protocol, 2023. Accessed: 2025-08-08.

19 Aleksandar Mijatović and Mikhail Urusov. On the martingale property of certain local martingales. *Probability Theory and Related Fields*, 152(1):1–30, 2012.

20 Allan Niemerg, Dan Robinson, and Lev Livnev. Yieldspace: An automated liquidity provider for fixed yield tokens. *Retrieved Feb*, 24:2021, 2020.

21 Opyn. Squeeth. https://squeeth.opyn.co/, 2021. Accessed: 2025-08-08.

22 pvpAMM. pvpamm codes. https://github.com/zhshang1221/pvpAMM, 2025. Accessed: 2025-08-08.

23 Qihong Ruan and Artem Streltsov. Perpetual contracts and market quality: Evidence from cryptocurrencies. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4218907, 2025. SSRN working paper. Originally posted: Sep 25, 2022. Last revised: Jun 22, 2025. Accessed: Aug 8, 2025. doi:10.2139/ssrn.4218907.

24 Robert J Shiller. Measuring asset values for cash settlement in derivative markets: hedonic repeated measures indices and perpetual futures. *The Journal of Finance*, 48(3):911–931, 1993.

25 Jupiter Team. Understanding how jupiter perps works: Complete guide, 2023. URL: https://station.jup.ag/guides/perpetual-exchange/how-it-works.

26 Sergey Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. Oracles in decentralized finance: Attack costs, profits and mitigation measures. *Entropy*, 25(1):60, 2023. doi:10.3390/E25010060.

27 D White, D Robinson, Z Koticha, A Leone, A Gauba, and A Krishnan. Power perpetuals, 2021.

28 Molly Wintermute. Hegic: On-chain options trading protocol on ethereum powered by hedge contracts and liquidity pools. Technical report, Hegic Protocol (via GitHub), February 2020.

29 Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *ACM Computing Surveys*, 55(11):1–50, 2023. doi:10.1145/3570639.

30 Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. Sok: Decentralized finance (defi) attacks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2444–2461. IEEE, 2023. doi:10.1109/SP46215.2023.10179435.

A Proofs

A.1 Proofs for Natural Worth Allocation in pvpAMM

We provide proofs for propositions within Section 2.2, which demonstrate that the natural worth allocation $w_j^{\text{natural}}(t) = w_j(t)/w_{\text{pool}}(t) \cdot m_{\text{pool}}(t)$ leads to disincentives for creating new positions when $w_{\text{pool}}(t_j) > m_{\text{pool}}(t_j)$ or vulnerabilities to flash-loan attacks when $w_{\text{pool}}(t_j) < m_{\text{pool}}(t_j)$.

Proof.

$$\begin{aligned}\lim_{t\to t_j^+} w_j^{natural}(t) - m_j &= \left(\lim_{t\to t_j^+} w_j(t)\right)\cdot\left(\frac{\lim_{t\to t_j^+} m_{pool}(t)}{\lim_{t\to t_j^+} w_{pool}(t)}\right) - m_j \\ &= m_j\cdot\left(\frac{m_{pool}(t_j)+m_j}{w_{pool}(t_j)+w_j(t_j)}\right) - m_j \\ &= m_j\cdot\left(\frac{m_{pool}(t_j)+m_j}{w_{pool}(t_j)+m_j}\right) - m_j \\ &= m_j\cdot\left(\frac{m_{pool}(t_j)-w_{pool}(t_j)}{w_{pool}(t_j)+m_j}\right).\end{aligned}$$

◀

A.2 Proof for LPT Balance in GMX's Solution

We prove Proposition 3, which states that the LPT balance for position j in the GMX protocol converges to

$$\mathscr{B}_j(t) \to \left(\frac{m_j}{\lambda^{\text{gmx}}(t_j)}\right)\left(\frac{P_t}{P_{t_j}}\right)^{b_j}$$

as the time interval for price updates $\Delta t \to 0$.

Proof. The price of LPT remains consistent across all positions and follows the equation:

$$\lambda^{gmx}(t) = \frac{\sum_j m_j}{\sum_j \mathscr{B}_j(t)} = \frac{\sum_j m_j}{\sum_j \underbrace{\prod_{s=t_j}^{t}\left(\frac{m_j}{\lambda^{gmx}(t_j)}\right)\left(1+b_j(\frac{\Delta P_s}{P_s})\right)}_{LPT\ balance\ for\ position\ j}}.$$

then

$$\begin{aligned}\prod_{s=t_j}^{t}\left(1+b_j\left(\frac{\Delta P_s}{P_s}\right)\right) &= e^{\sum_s \log\left(1+b_j(\frac{\Delta P_s}{P_s})\right)} \sim e^{\sum_s b_j(\frac{\Delta P_s}{P_s})} \\ &= e^{b_j\int_{t_j}^{t}\frac{dP}{P}} = e^{b_j\log\left(\frac{P_t}{P_{t_j}}\right)} = \left(\frac{P_t}{P_{t_j}}\right)^{b_j}.\end{aligned}$$

◀

A.3 Conservation and Continuity in pvpAMM with Position Clearing

In Section 3.1, we derived the conservation function and the scaler function $\psi(t)$ for the pvpAMM protocol assuming all positions remain active after creation (see Proposition 2). Here, we extend the analysis to include position clearing at times T_j, showing that the conservation function holds and that $\psi(t)$ remains continuous at position creation (t_k) and clearing (T_k) times.

Conservation Function

$$\sum_{j:t_j<t} m_j = \sum_{j:t_j<t\le T_j} w_j(t)\lambda(t)/\lambda(t_j) + \sum_{j:T_j<t} w_j(T_j)\lambda(T_j)/\lambda(t_j)$$

and

$$\sum_{j:t_j<t} m_j = \sum_{j:t_j<t\le T_j} w_j(t)\psi(t_j)/\psi(t) + \sum_{j:T_j<t} w_j(T_j)\psi(t_j)/\psi(T_j).$$

Continuity of $\psi(t)$

Proof. For the position creation time t_k, we may find that

$$\begin{aligned}\lim_{t\to t_k^+}\psi(t_k) &= \frac{\sum_{j:t_j<t_k\le T_j} w_j(t_k)\psi(t_j)+w_k(t_k)\psi(t_k)}{\sum_{j:t_j<t_k} m_j-\sum_{j:T_j<t_k} w_j(T_j)\psi(t_j)/\psi(T_j)+m_k}\\ &= \frac{\left(\sum_{j:t_j<t_k} m_j-\sum_{j:T_j<t_k} w_j(T_j)\psi(t_j)/\psi(T_j)\right)\psi(t_k)+w_k(t_k)\psi(t_k)}{\left(\sum_{j:t_j<t_k} m_j-\sum_{j:T_j<t_k} w_j(T_j)\psi(t_j)/\psi(T_j)\right)+w_k(t_k)}\\ &= \psi(t_k),\end{aligned}$$

and for the clearing time T_k, we have:

$$\begin{aligned}\lim_{t\to T_k^+}\psi(T_k) &= \frac{\sum_{j:t_j<T_k\le T_j} w_j(T_k)\psi(t_j)-w_k(T_k)\psi(t_k)}{\sum_{j:t_j<T_k} m_j-\sum_{j:T_j<T_k} w_j(T_j)\psi(t_j)/\psi(T_j)-w_k(T_k)\psi(t_k)/\psi(T_k)}\\ &= \frac{\left(\sum_{j:t_j<T_k} m_j-\sum_{j:T_j<T_k} w_j(T_j)\psi(t_j)/\psi(T_j)\right)\psi(T_k)-w_k(T_k)\psi(t_k)}{\left(\sum_{j:t_j<t_k} m_j-\sum_{j:T_j<t_k} w_j(T_j)\psi(t_j)/\psi(T_j)\right)-w_k(T_k)\psi(t_k)/\psi(T_k)}\\ &= \psi(T_k).\end{aligned}$$

◀

A.4 Derivation of $\psi(t)$ Dynamics for Theorem 6

We provide the detailed derivation for the expression of $\frac{\psi(s)-\psi(t)}{\psi(s)}$ used in the proof of Theorem 6.

Proof.

$$\begin{aligned}\frac{\psi(s)-\psi(t)}{\psi(s)} &= \frac{\psi(s)-\psi(t)}{\psi(t)+\psi(s)-\psi(t)} = \frac{(\psi(s)-\psi(t))/\psi(t)}{1+(\psi(s)-\psi(t))/\psi(t)} \qquad (1)\\ &= \frac{\mathrm{d}\psi(t)/\psi(t)}{1+\mathrm{d}\psi(t)/\psi(t)} = \frac{\mathrm{d}\psi(t)}{\psi(t)}-\left(\frac{\mathrm{d}\psi(t)}{\psi(t)}\right)^2+o\left(\left(\frac{\mathrm{d}\psi(t)}{\psi(t)}\right)^2\right)\\ &= \xi(t)\frac{\mathrm{d}P_t}{P_t}-\xi(t)^2\left(\frac{\mathrm{d}P_t}{P_t}\right)^2+o\left(\left(\frac{\mathrm{d}P_t}{P_t}\right)^2\right)\\ &= \xi(t)\Delta P_t-\xi(t)^2(\Delta P_t)^2+o((\Delta P_t)^2).\end{aligned}$$

◀

B Numerical Examples

B.1 GMX's LPT Solution

Use USD as the default numéraire. At time t_0, Alice places a long order with 200 USD, 2× leverage. Bob places a short order with 100 USD, 3× leverage.

1. $t=t_0$, the initial spot price P_{t_0}=100, the initial LPT price $\lambda^{gmx}(t_0)=1$.
   - Alice's LPT amount, $\mathscr{B}_1(t_0)=m_1/\lambda^{gmx}(t_0)=200$.
   - Bob's LPT amount, $\mathscr{B}_2(t_0)=m_2/\lambda^{gmx}(t_0)=100$.
   - Total collateral is $m_{pool}=m_1+m_2=300$.
   - Total LPT amount is $\mathscr{B}(t_0)=\mathscr{B}_1(t_0)+\mathscr{B}_2(t_0)=300$.
   - New LPT price $\lambda^{gmx}(t_0)=m_{pool}/\mathscr{B}(t_0)=1$.
   - Alice's position value $w_1^{gmx}(t_0)=\mathscr{B}_1(t_0)\cdot\lambda^{gmx}(t_0)=200$.
   - Bob's position value $w_2^{gmx}(t_0)=\mathscr{B}_2(t_0)\cdot\lambda^{gmx}(t_0)=100$.

2. $t = t_1$, assume the spot price rises 10% to $P_{t_1} = 110$.
   - Alice's LPT amount, $\mathscr{B}_1(t_1) = \mathscr{B}_1(t_0)(1 + b_1 \times 10\%) = 200 \times (1 + 2 \times 10\%) = 240$.
   - Bob's LPT amount, $\mathscr{B}_2(t_1) = \mathscr{B}_2(t_0)(1 + b_2 \times 10\%) = 100 \times (1 - 3 \times 10\%) = 70$.
   - Total collateral is $m_{pool} = m_1 + m_2 = 300$.
   - Total LPT amount is $\mathscr{B}(t_1) = \mathscr{B}_1(t_1) + \mathscr{B}_2(t_1) = 310$.
   - New LPT price $\lambda^{gmx}(t_1) = m_{pool}/\mathscr{B}(t_1) = 0.9677$.
   - Alice's position value $w_1^{gmx}(t_1) = \mathscr{B}_1(t_1) \cdot \lambda^{gmx}(t_1) = 232.2581$.
   - Bob's position value $w_2^{gmx}(t_1) = \mathscr{B}_2(t_1) \cdot \lambda^{gmx}(t_1) = 67.7419$.
3. $t = t_2$, assume the spot price rises 10% to $P_{t_2} = 121$.
   - Alice's LPT amount, $\mathscr{B}_1(t_2) = \mathscr{B}_1(t_1)(1 + b_1 \times 10\%) = 240 \times (1 + 2 \times 10\%) = 288$.
   - Bob's LPT amount, $\mathscr{B}_2(t_2) = \mathscr{B}_2(t_1)(1 + b_2 \times 10\%) = 70 \times (1 - 3 \times 10\%) = 49$.
   - Total collateral is $m_{pool} = m_1 + m_2 = 300$.
   - Total LPT amount is $\mathscr{B}(t_2) = \mathscr{B}_1(t_2) + \mathscr{B}_2(t_2) = 337$.
   - New LPT price $\lambda^{gmx}(t_2) = m_{pool}/\mathscr{B}(t_2) = 0.8902$.
   - Alice's position value $w_1^{gmx}(t_2) = \mathscr{B}_1(t_2) \cdot \lambda^{gmx}(t_2) = 256.3798$.
   - Bob's position value $w_2^{gmx}(t_2) = \mathscr{B}_2(t_2) \cdot \lambda^{gmx}(t_2) = 43.6202$.

B.2 pvpAMM's PLT Solution

Use USD as the default numéraire. At time t_0, Alice places a long order with 200 USD, 2× leverage. Bob places a short order with 100 USD, 3× leverage.

1. $t = t_0$, the initial spot price P_{t_0}=100, the initial PLT price $\lambda^{pvp}(t_0) = 1$.
   - Alice's PLT amount, $\mathscr{B}_1(t_0) = m_1/\lambda^{pvp}(t_0) = 200$.
   - Bob's PLT amount, $\mathscr{B}_2(t_0) = m_2/\lambda^{pvp}(t_0) = 100$.
   - Total collateral is $m_{pool} = m_1 + m_2 = 300$.
   - Total PLT amount is $\mathscr{B}(t_0) = \mathscr{B}_1(t_0) + \mathscr{B}_2(t_0) = 300$.
   - New PLT price $\lambda^{pvp}(t_0) = m_{pool}/\mathscr{B}(t_0) = 1$.
   - Alice's position value $\tilde{w}_1(t_0) = \mathscr{B}_1(t_0) \cdot \lambda^{pvp}(t_0) = 200$.
   - Bob's position value $\tilde{w}_2(t_0) = \mathscr{B}_2(t_0) \cdot \lambda^{pvp}(t_0) = 100$.
2. $t = t_1$, assume the spot price rises 10% to $P_{t_1} = 110$.
   - Alice's PLT amount, $\mathscr{B}_1(t_1) = w_1(t_1)/\lambda^{pvp}(t_0) = \left(200 - 200 \times 2 + \frac{200 \times 2}{100} \times 110\right)/1 = 240$.
   - Bob's PLT amount, $\mathscr{B}_2(t_1) = w_2(t_1)/\lambda^{pvp}(t_0) = \left(100 + 100 \times 3 - \frac{100 \times 3}{100} \times 110\right)/1 = 70$.
   - Total collateral is $m_{pool} = m_1 + m_2 = 300$.
   - Total PLT amount is $\mathscr{B}(t_1) = \mathscr{B}_1(t_1) + \mathscr{B}_2(t_1) = 310$.
   - New PLT price $\lambda^{pvp}(t_1) = m_{pool}/\mathscr{B}(t_1) = 0.9677$.
   - Alice's position value $\tilde{w}_1(t_1) = \mathscr{B}_1(t_1) \cdot \lambda^{pvp}(t_1) = 232.2581$.
   - Bob's position value $\tilde{w}_2(t_1) = \mathscr{B}_2(t_1) \cdot \lambda^{pvp}(t_1) = 67.7419$.
3. $t = t_2$, assume the spot price rises 10% to $P_{t_2} = 121$.
   - Alice's PLT amount, $\mathscr{B}_1(t_2) = w_1(t_2)/\lambda^{pvp}(t_0) = \left(200 - 200 \times 2 + \frac{200 \times 2}{100} \times 121\right)/1 = 284$.
   - Bob's PLT amount, $\mathscr{B}_2(t_2) = w_2(t_2)/\lambda^{pvp}(t_0) = \left(100 + 100 \times 3 - \frac{100 \times 3}{100} \times 121\right)/1 = 37$.
   - Total collateral is $m_{pool} = m_1 + m_2 = 300$.
   - Total PLT amount is $\mathscr{B}(t_2) = \mathscr{B}_1(t_2) + \mathscr{B}_2(t_2) = 321$.
   - New PLT price $\lambda^{pvp}(t_2) = m_{pool}/\mathscr{B}(t_2) = 0.9346$.
   - Alice's position value $\tilde{w}_1(t_2) = \mathscr{B}_1(t_2) \cdot \lambda^{pvp}(t_2) = 265.4206$.
   - Bob's position value $\tilde{w}_2(t_2) = \mathscr{B}_2(t_2) \cdot \lambda^{pvp}(t_2) = 34.5794$.

B.3 LONG positions only in pvpAMM

Use USD as the default numéraire. At time t_0, Alice places a long order with 200 USD, 2× leverage. Bob places a long order with 100 USD, 3× leverage.

1. $t = t_0$, the initial spot price P_{t_0}=100, the initial PLT price $\lambda^{pvp}(t_0) = 1$.
   - Alice's PLT amount, $\mathscr{B}_1(t_0) = m_1/\lambda^{pvp}(t_0) = 200$.
   - Bob's PLT amount, $\mathscr{B}_2(t_0) = m_2/\lambda^{pvp}(t_0) = 100$.
   - Total collateral is $m_{pool} = m_1 + m_2 = 300$.
   - Total PLT amount is $\mathscr{B}(t_0) = \mathscr{B}_1(t_0) + \mathscr{B}_2(t_0) = 300$.
   - New PLT price $\lambda^{pvp}(t_0) = m_{pool}/\mathscr{B}(t_0) = 1$.
   - Alice's position value $\tilde{w}_1(t_0) = \mathscr{B}_1(t_0) \cdot \lambda^{pvp}(t_0) = 200$.
   - Bob's position value $\tilde{w}_2(t_0) = \mathscr{B}_2(t_0) \cdot \lambda^{pvp}(t_0) = 100$.
2. $t = t_1$, assume the spot price rises 10% to $P_{t_1} = 110$.
   - Alice's PLT amount, $\mathscr{B}_1(t_1) = w_1(t_1)/\lambda^{pvp}(t_0) = \left(200 - 200 \times 2 + \frac{200\times 2}{100} \times 110\right)/1 = 240$.
   - Bob's PLT amount, $\mathscr{B}_2(t_1) = w_2(t_1)/\lambda^{pvp}(t_0) = \left(100 - 100 \times 3 + \frac{100\times 3}{100} \times 110\right)/1 = 130$.
   - Total collateral is $m_{pool} = m_1 + m_2 = 300$.
   - Total PLT amount is $\mathscr{B}(t_1) = \mathscr{B}_1(t_1) + \mathscr{B}_2(t_1) = 370$.
   - New PLT price $\lambda^{pvp}(t_1) = m_{pool}/\mathscr{B}(t_1) = 0.8108$.
   - Alice's position value $\tilde{w}_1(t_1) = \mathscr{B}_1(t_1) \cdot \lambda^{pvp}(t_1) = 194.5946$.
   - Bob's position value $\tilde{w}_2(t_1) = \mathscr{B}_2(t_1) \cdot \lambda^{pvp}(t_1) = 105.4054$.
3. $t = t_2$, assume the spot price rises 10% to $P_{t_2} = 121$.
   - Alice's PLT amount, $\mathscr{B}_1(t_2) = w_1(t_2)/\lambda^{pvp}(t_0) = \left(200 - 200 \times 2 + \frac{200\times 2}{100} \times 121\right)/1 = 242$.
   - Bob's PLT amount, $\mathscr{B}_2(t_2) = w_2(t_2)/\lambda^{pvp}(t_0) = \left(100 - 100 \times 3 + \frac{100\times 3}{100} \times 121\right)/1 = 163$.
   - Total collateral is $m_{pool} = m_1 + m_2 = 300$.
   - Total PLT amount is $\mathscr{B}(t_2) = \mathscr{B}_1(t_2) + \mathscr{B}_2(t_2) = 403$.
   - New PLT price $\lambda^{pvp}(t_2) = m_{pool}/\mathscr{B}(t_2) = 0.7444$.
   - Alice's position value $\tilde{w}_1(t_2) = \mathscr{B}_1(t_2) \cdot \lambda^{pvp}(t_2) = 180.1489$.
   - Bob's position value $\tilde{w}_2(t_2) = \mathscr{B}_2(t_2) \cdot \lambda^{pvp}(t_2) = 119.8511$.

On-Chain Decentralized Learning and Cost-Effective Inference for DeFi Attack Mitigation

Abdulrahman Alhaidari
School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA

Balaji Palanisamy
School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA

Prashant Krishnamurthy
School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA

Abstract

Billions of dollars are lost every year in DeFi platforms by transactions exploiting business logic or accounting vulnerabilities. Existing defenses focus on static code analysis, public mempool screening, attacker contract detection, or trusted off-chain monitors, none of which prevents exploits submitted through private relays or malicious contracts that execute within the same block. We present the first decentralized, fully on-chain learning framework that: (i) performs gas-prohibitive computation on Layer-2 to reduce cost, (ii) propagates verified model updates to Layer-1, and (iii) enables gas-bounded, low-latency inference inside smart contracts. A novel Proof-of-Improvement (PoIm) protocol governs the training process and verifies each decentralized micro update as a self-verifying training transaction. Updates are accepted by *PoIm* only if they demonstrably improve at least one core metric (e.g., accuracy, F1-score, precision, or recall) on a public benchmark without degrading any of the other core metrics, while adversarial proposals get financially penalized through an adaptable test set for evolving threats. We develop quantization and loop-unrolling techniques that enable inference for logistic regression, SVM, MLPs, CNNs, and gated RNNs (with support for formally verified decision tree inference) within the Ethereum block gas limit, while remaining bit-exact to their off-chain counterparts, formally proven in Z3. We curate 298 unique real-world exploits (2020 - 2025) with 402 exploit transactions across eight EVM chains, collectively responsible for $3.74 B in losses. We demonstrate that on-chain ML governed by *PoIm* detects previously unseen attacks with over 97% attack detection accuracy and 82.0% F1. A single inference, such as one made via an external call, typically incurs zero cost. Fully on-chain inference consumes 57,603 gas (≈ $0.18) for linear models, 143,647 gas (≈ $0.49) for CNN(F2, K1), and 506,397 gas (≈ $1.77) for CNN(F8, K4) on L1 (e.g., Ethereum). Our results show that practical and continually evolving DeFi defenses can be embedded directly in protocol logic without trusted guardians, and our solution achieves highly cost-effective protection while filling a critical gap between vulnerability scanners and real-time transaction screening.

2012 ACM Subject Classification Security and privacy → Network security; Security and privacy → Distributed systems security; Security and privacy → Security protocols

Keywords and phrases DeFi attacks, on-chain machine learning, decentralized learning, real-time defense

Digital Object Identifier 10.4230/LIPIcs.AFT.2025.35

Funding This material is based upon work supported by the National Science Foundation under Grant #2020071. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

© Abdulrahman Alhaidari, Balaji Palanisamy, and Prashant Krishnamurthy;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Vulnerabilities in decentralized finance (DeFi) protocols are triggered through transactions [8]. Attackers often do not bypass contract safeguards directly [55], instead they craft transactions that invoke legitimate functions to trigger state changes that the protocol did not intend. Exploits occur when contracts proceed with malicious inputs while assuming invariant state conditions, such as executing a withdrawal without checking balance or allowance or privileged operations without enforcing access control [50]. These attacks can be single transactions or atomic sequences that satisfy syntax checks but produce unauthorized asset control. The execution logic can be formally valid, yet produce outcomes that violate the protocol's security assumptions [9]. Vulnerabilities come from code bugs and manipulating contracts. Attackers often chain operations (for example, in atomic transactions) that appear normal in unexpected ways to generate exploits [12]. However, most previous work focuses on static code-level bugs [52] and overlooks protocol-level flaws [68].

Static analysis detects patterns that violate predefined coding conventions, but fails to capture logic flaws that depend on contract state, cross-function flows, or interactions across multiple protocols [10]. For instance, Cream Finance lost over $130M through an interaction with another protocol that allowed borrowing without triggering the appropriate collateral checks [47]. Formal verification tools prove that certain invariants hold under all code paths but do not encode financial semantics or simulate attacker incentives. Most verification frameworks cannot model adversarially composed transaction sequences, in which attackers combine individually valid operations to produce exploits. Thus, zero-day vulnerabilities continue to appear in DeFi [68]. Tools for detecting malicious smart contracts assume a time window between contract deployment and first exploitation [54], during which vulnerabilities can be analyzed. In practice, attackers often deploy and execute the exploit within a single block or bypass deployment entirely by sending malicious transactions from externally owned accounts (EOA) [41].

Other methods, such as post-attack analysis, provide insight into what has happened but offer no protection. They begin only after an exploit has occurred and rely on retrospective debugging of protocol states [19]. However, recent attacks in DeFi have compromised over $79.8 billion in DeFi assets and only $6.7 billion of them have been recovered [18]. This emphasizes a core challenge in shared threat intelligence, particularly for aftermath attack analysis, where dissemination is often delayed. This delay increases the likelihood of repeat attacks on existing deployed protocols that may harbor the same unaddressed vulnerabilities. Methods such as front-running protection [67] monitor public mempools (public queue of transactions) but miss transactions submitted through private relays such as Flashbots [25]. While off-chain monitoring systems (e.g., [67, 6, 54]) can detect some attacks, they typically focus on specific classes (e.g., flash loans, reentrancy). Moreover, their effectiveness is diminished by latency, which can lead to costly responses (due to gas fees) and reduced efficacy, particularly against sophisticated attacks [61]. Therefore, there is a need for protocol-integrated security mechanisms that add an evolving layer of protection (i.e., Intrusion Prevention System (IPS)) to smart contracts and react in real time without relying on external entities, such as pre-attack (e.g., auditing) or post-attack (i.e., attack tracing) countermeasures. On the other hand, DeFi protocols currently lack mechanisms to evaluate transaction intent during execution other than hard-coded logic, which makes them vulnerable to attacks that were not anticipated during smart contract design.

Solutions driven by machine learning (ML) show strong capabilities in attack detection [33, 37]. However, deploying ML models directly to Layer-1 blockchains such as Ethereum faces major obstacles in computation and storage. The computational demands of ML

algorithms result in prohibitive gas costs. Ethereum's block gas limits (≈ 30 million units per block [26]) impose constraints on the computational complexity of transactions, making complex ML models impossible to run in a single transaction. A basic model inference might consume 30-50% of an entire block. Storage costs create another barrier. Saving model parameters on-chain is extremely expensive [37]. These combined limitations make direct Layer-1 deployment of ML solutions economically and technically impractical for real-world deployment.

To address the above-mentioned gaps, we propose a decentralized training architecture where all model training and governance occur on Layer-2 (L2), while inference is optimized and happens on Layer-1 (L1) under strict gas constraints (Table 1). In our framework, an ML model is trained to provide a layer of security for smart contracts running on the Ethereum (L1) network. Our framework leverages L2 (e.g., Optimism rollup) for intensive computation and L1 for optimized inference to overcome L1 resource constraints (such as computation limits and gas fees). The rollup provides cost-effective computation while inheriting the base blockchain's security guarantees [60]. Training is performed on L2, and L1 updates are governed by L2 decentralized nodes, where computation is cheaper, and the learned model is cryptographically verified and propagated to L1 for inference. Inference runs at zero cost through read-only classifiers (for example, pure or view functions) or is fully verified on L1 embedded in on-chain contracts for real-time transaction classification. To support a range of use cases, we propose two tiers: *fully zero-cost inference* (for users and smart contracts) on L1, and *fully on-chain ML on L1*. The system acts as a transaction gatekeeper (i.e., firewall) even for high-throughput or low-value use cases. Our approach is model-agnostic and supports various ML algorithms, including linear algorithms (e.g., logistic regression, SVMs) and non-linear models (e.g., neural networks up to 10 layers, including 10-layer convolutional neural networks (CNN)). These models are optimized and serialized into constant-time evaluation logic fully bounded by L1 constraints. We found that even low-overhead mechanisms (linear models) are sufficient to detect a wide range of attacks. The models are trained using micro-steps by decentralized peers based on the collective knowledge of the DeFi platforms, and their formally verifiable performance matches traditional on-chain counterparts without approximation, enabling detection of both known and novel transaction behaviors.

As DeFi exploits evolve over time and to maintain continuous learning of new exploits as they appear, we introduce the notion of *Proof-of-Improvement (PoIm)*. *PoIm* is a decentralized protocol on L2 that governs and verifies micro-step model training and its L1-propagated updates. A deployed model is designed to be shared globally across DeFi protocols, allowing any platform to contribute by training it on one candidate (malicious or normal) transaction at a time. These DeFi platforms collectively shape a model that becomes robust over time and enables unified sharing of attack intelligence. All updates are evaluated against an on-chain (L2) committed, agreed-upon benchmark of past exploit and benign transaction data. Updates are accepted only if they demonstrably improve at least one of the key performance metrics (namely precision, recall, accuracy, or F1-score) without degrading others, as evaluated by *PoIm* against the on-chain benchmark and it is verifiable by any node. Submitters are rewarded in proportion to the verified performance gain, while failed proposals lose their stake. If a malicious update seemingly improves metrics but enables a detection bypass, peers can vote to roll back the model to the last stable version. *PoIm* enables the L1 classifier to evolve with newly emerging exploits without relying on centralized oversight.

Table 1 Comparison of our work with prior on-chain ML studies. **DT** = Decentralized Training, **MC** = Model Consistency (On-/Off-chain), **G** = Governance, **DF** = DeFi Focus, **IC** = Inference Cost, **TM** = Trust Model, **VM** = Validation Method.

Study	DT	MC	G	DF	IC	TM	VM
Our Work	✓	✓	✓	✓	Zero / Low	*PoIm*	On-chain + commit-reveal
ML2SC [42]	✗	*Partial*[1]	✗	✗	High	✗	✗
LMST [57]	✗	*Partial*[2]	✗	✗	Mod-High	✗	✗
opML [15]	✗[3]	✓	✓	✗	Low*	AnyTrust	Fraud-proof

[1] Minor mismatches due to PRBMath; [2] Accuracy drop from fixed-point; [3] Training off-chain, validated via fraud proof; * Optimistic assumption, only $O(1)$ on-chain arbitration.

For our evaluation, since there is no publicly available transaction data for DeFi attacks, we manually collected 298 confirmed exploit transactions from real DeFi attacks across eight major blockchains: *Ethereum, Binance Smart Chain (BSC), Polygon, Avalanche, Arbitrum, Fantom, Moonriver, and Base*. Our exploit collection generalizes to cover attacks that exploited smart contract vulnerabilities in the past five years (2020 - April 2025), reported in news, social media platforms (e.g., X), blogs, and DeFi attack documentation. We utilize blockchain data from public explorers (for example, Etherscan [22]) and Web3 Remote Procedure Calls (RPCs) (e.g., using libraries/services like Web3.js [14] and Alchemy [4]), along with DeFiHackLabs [58], the DeFi Rekt Database [18], and DeFiLlama [19] as ground truth for guiding historical exploit collection. Each transaction contains the exact exploit transaction data used by the attacker, as observed by the smart contract at execution time, including call parameters, sender addresses (EOAs or smart contracts), and relevant blockchain state at the time of the exploit. In addition, we collected comprehensive metadata for each attack, including protocol names, exploited functions, attack methods, root causes, and financial losses. We use these real-world exploits, which collectively caused over $3.74 billion in losses. Our evaluation shows that our approach is highly gas-efficient. CNN and RNN models governed by our decentralized *PoIm* protocol achieve over 97% recall and up to an 82% F1-score on unseen DeFi exploits.

Contributions. This paper makes the following contributions:

- We design a decentralized learning framework with training and governance on L2 and enable two tiers: zero-cost inference and fee-optimized inference on L1 for real-time classification of DeFi transactions.
- We introduce *PoIm*, a decentralized L2 protocol that governs model training and propagates verified updates to L1 for inference.
- We formally verify inference correctness, model update integrity, and L1/L2 consistency under gas and computation constraints.
- We evaluated our framework on a curated set of 298 manually collected real-world exploits. It achieves high detection performance: SVM reaches an F1-score of 80%, and CNN(F4, K4) achieves 82% F1, 90.04% accuracy, and over 97% recall on unseen exploits. L1 inference is efficient, requiring only 57,603 gas for linear models and 143,647 gas for CNN(F2, K1). Zero-cost inference is supported via external EVM nodes.

2 Background

Layer 2 Blockchains. Layer 2 (L2) blockchains (e.g., Optimism, Arbitrum, Base) are built on top of Layer 1 (L1) platforms (e.g., Ethereum, Binance Smart Chain (BSC)) to improve scalability and reduce costs [56]. In contrast to L1s that execute and store every transaction

directly on-chain [26], L2s batch many transactions and periodically submit compressed proofs to the L1 [63]. This design allows L2s to offer significantly lower gas fees and faster execution and inherits the security guarantees from L1 [56]. Notably, L2s support the same smart contract logic as L1s but with relaxed resource constraints, which makes them ideal for computationally heavy tasks like model training or repeated inference.

Transactions. Transactions are the main component used to facilitate both asset movement and protocol interactions. A nonce in a transaction acts as a sequence counter to prevent replay attacks, where the attacker could replicate the transaction. The gas fee, influenced by user-specified price and the complexity of transactions, determines inclusion priority, while the gas limit defines the upper computational allowance per transaction [62].

Exploit transactions. Most of the DeFi attacks succeed through a maliciously crafted sequence of instructions (e.g., internal transactions) [20]. These attacks exploit weaknesses in smart contracts rather than modifying contract code at the infrastructure level [71]. Attackers manipulate transaction parameters, call sequences, and attempt to manipulate permission states through public interfaces to gain unauthorized assets [55].

Mempool and private relays. Blockchain transactions are submitted either through the public transaction buffer, known as the mempool, or via private relays [25], where the transaction is sent directly to miners. Transactions submitted via private relays remain hidden from the public, whereas those using the mempool are broadcast and await confirmation [13]. Nodes share this queue across the network. The selection of which transactions to include in a block is typically based on miners/validators' incentives and the gas fees offered. Transactions with higher fees are generally prioritized [43]. The mempool is publicly visible, enabling a brief window during which real-time monitoring can be used to detect malicious behavior, such as front-running [36]. Since each node maintains a synchronized copy of unconfirmed transactions, this window supports early threat detection [31]. However, detection is not always possible due to the complexity of some attacks, and in many cases, the malicious transaction is submitted directly to miners/validators, bypassing public visibility [46].

2.1 Limitations of Existing Defenses

This section discusses fundamental limitations of existing preventive measures and real-time defense mechanisms. First, off-chain detection systems such as LookAhead [54] rely on a temporal gap between transaction submission and execution. When an attacker deploys a malicious contract and immediately initiates an exploit within a single block, these systems fail to respond on time. Second, private relay services like Flashbots [25] allow attackers to bypass mempool-based detection by submitting transactions directly to miners/validators. In such settings, traditional monitoring tools lose all visibility, allowing stealthy attacks without external traceability.

Third, signature-based detection methods depend on identifying known function selectors or matching call patterns against static signatures [6]. This approach fails when attackers use proxy contracts, delegate calls, or obfuscated logic flows, where surface-level transaction signatures are intentionally masked. Fourth, current on-chain defenses lack the ability to validate transaction behavior dynamically during execution. Existing systems either rely on static pre-deployment audits or basic access control checks, leaving dynamic runtime behavior, such as unauthorized fund movements or privilege escalations, undetected.

Fifth, most prior defenses prioritize contract vulnerabilities rather than transaction behavior. However, even with this focus, a recent study [10] found that automated tools can detect only 8% of vulnerabilities. Another study found that existing tools can detect only 20% of vulnerabilities [68], which leaves 80% undetected. Consequently, they can be exploited and cause huge financial losses. One reason is that not all bugs are exploitable, and not all correct logic code is vulnerability-free [50]. Thus, the focus on code logic overlooks the fact that many exploits occur through crafted transaction sequences exploiting valid contract interfaces without exploiting traditional code vulnerabilities.

Also, existing solutions typically adopt a centralized approach. Systems that depend on trusted off-chain detectors or external alert mechanisms inherently introduce single points of failure and trust dependencies incompatible with DeFi principles. Pre-interaction analysis tools such as DeFiAligner [28] can detect inconsistencies between a project's documentation and on-chain bytecode. However, they do not provide real-time defense against malicious transaction behavior.

2.2 Limitations of On-Chain ML

Embedding machine learning models into L1 smart contracts is resource-intensive, and replicating off-chain models directly on-chain can be impossible. L1 blockchains are not designed for heavy computing [3]. However, inference, if designed without inheriting the complexity of off-chain models (e.g., using TensorFlow [2], Keras [38], Scikit-learn [39]) yet performing equivalently, can be feasible if heavy computations are separated from inference.

Previous designs (e.g., [42, 57]) face many practical barriers, including prohibitive gas costs for inference, inconsistent behavior between off-chain and on-chain models due to numerical limitations, and incompatibility with smart contract languages lacking native support for floating-point operations, which causes model output deviation (off-chain vs on-chain counterpart). For example, the authors in [57] reported inconsistent accuracy between the PyTorch and on-chain models: 86.00% off-chain versus 81.00% on-chain. The deployment cost also exceeds the current Ethereum block gas limit, with a reported cost of 73,721,648 gas. This is more than twice Ethereum's current block gas limit, which is approximately 30 million gas units.

Our work achieves exact consistency (formally and empirically verified) between off-chain and on-chain model outputs. Unlike prior methods, our on-chain design uses an optimized smart contract that replicates the off-chain model's behavior without approximation to achieve the same outcome. This will allow DeFi attacks mitigation dynamically using practical, low-cost, and fully decentralized on-chain machine learning.

3 DeFi Attacks and Threat Model

The current state of defenses is not sufficient to mitigate the growing threat of unauthorized fund exploits in Decentralized Finance (DeFi). Recent studies [10, 40, 13, 71, 68] show that attack techniques are rapidly advancing but defenses remain inadequate. To address these threats, we need a new defense model based on a real-time, in-protocol (smart contract-level) transaction classification system that must operate independently of external monitors or pre-defined signature lists [71], and it must work even when transactions are hidden from the public mempool. It should also generalize to both known and novel (zero-day) attacks that exploit DeFi contract flaws. However, statistical methods have shown significant promise in detecting not only known attacks but also previously unseen zero-day exploits [33]. This naturally raises several key questions: Can data from past DeFi attacks help us predict and

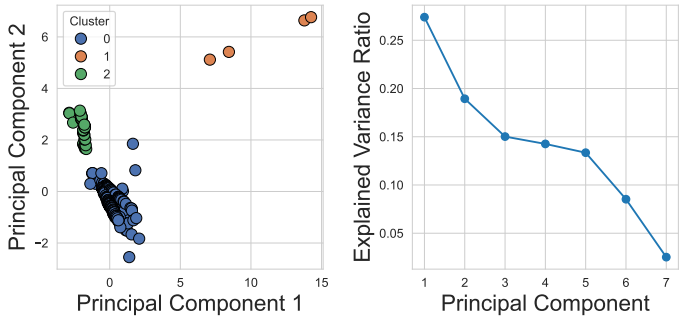


Figure 1 Left: PCA projection of DeFi attack transactions using two principal components with KMeans clustering ($k = 3$). Right: The scree plot shows the explained variance ratio per principal component. Even though PC1 and PC2 capture less than 50% of the total variance, the projection shows distinct DeFi attack transaction patterns.

prevent future ones? What are the possible designs for trustless, cost-effective, on-chain inference systems that can evolve with the attack landscape and be governed by decentralized peers? To answer these questions, we investigate whether real historical exploits exhibit detectable patterns that statistical techniques can reliably uncover and learn from.

3.1 Empirical Patterns of DeFi Attacks

To explore the feasibility of detecting DeFi attacks using only transaction data observable by smart contracts, we analyzed 402 confirmed DeFi exploit transactions (for 298 attacks). Transaction data includes complete EVM-level execution context such as gas usage, transferred value, calldata, and block metadata. Since previous studies offer limited analysis of DeFi attacks from a transactional perspective, we investigate whether exploit transactions exhibit measurable similarities that support transaction-level detection.

Table 2 Transaction metadata visible to smart contracts during execution.

Attribute	Description
`msg.sender`	Sender address (initiator of the call)
`msg.value`	ETH amount sent with the call
`msg.data`	Calldata containing function selector and arguments
`tx.origin`	Original externally-owned sender of the transaction
`gas`	Remaining gas (via `gasleft()`)
`gas_price`	Effective gas price paid (via `tx.gasprice`)
`msg.to`	Receiving contract's address (via `address(this)`)

We selected the numerical features observable during smart contract execution (Table 2) and excluded transaction hashes or addresses. Features were standardized, and dimensionality was reduced using Principal Component Analysis (PCA). We applied KMeans clustering with $k = 3$ to detect grouping behavior across attack transactions.

The PCA projection preserved 46.3% of the variance in two dimensions (Figure 1). Out of 402 exploits, 355 (88.3%) formed a distinct cluster. This suggests high behavioral consistency across transactions despite protocol and chain differences. Clustering quality was supported by a Calinski-Harabasz index of 981.78 and a Silhouette score of 0.773, both indicating well-separated, compact clusters.

These findings indicate that DeFi attacks share consistent runtime characteristics. The ability of PCA + KMeans to cluster these attacks supports the feasibility of transaction-level classification during execution. We classify the root causes of these attacks into five empirically supported categories, such as access control failures and business logic flaws.

Exploit Execution Patterns. Despite different root causes, several runtime-level patterns are consistent across attacks:

- **Atomicity:** 71.2% of exploits execute within a single transaction with no prior on-chain activity.
- **Benign interface misuse:** 67.5% invoke functions like `withdraw()`, `approve()`, or `sweep()` with malicious arguments.
- **Cross-chain replication:** Identical bytecode deployments (9.7%) are exploited across chains [35] (e.g., Hedgey [1]).

These patterns further support the view that exploitability is determined by transaction behavior rather than only static code.

3.2 Challenges in Real-Time Defense

We identified several key challenges that prevent current systems from defending against these attacks in real time:

- **C1. Private relay invisibility:** Transactions sent through Flashbots [25] and similar relays bypass the public mempool, evading pre-inclusion detection.
- **C2. Multiple attack paths per contract:** A single protocol may contain unrelated flaws, making static patching or signature detection ineffective.
- **C3. Benign-looking interfaces:** Safe-looking functions are abused with crafted inputs, undermining static signature-based detection.
- **C4. Cross-chain propagation:** An undetected exploit on one chain quickly propagates to other protocols with shared logic.
- **C5. No rollback:** Once executed, DeFi transactions are final. Most protocols do not have pause switches or delayed execution.

3.3 Threat Model and Assumptions

Our system defends against two primary threat scenarios. ① An attacker crafts a malicious transaction targeting a vulnerable contract. The transaction is evaluated by our on-chain ML classifier before execution. The attacker may vary calldata, timing, or submission channel (e.g., private relay), but cannot alter the deployed model or governance system. ② A malicious peer attempts to poison the training process by submitting manipulated updates that degrade model performance. The system detects and rejects such updates via on-chain benchmarking. Insider threats, compromised signing keys, and off-chain infrastructure attacks are out of scope for this threat model.

4 Decentralized Training, Inference, and Governance Framework

Our objective is to integrate decentralized learning with verifiable and efficient on-chain detection into the execution paths of DeFi contracts to mitigate exploit transactions in real time. We address limitations in existing works (e.g., [42, 67, 5, 54, 57]) from two perspectives: DeFi attack mitigation and on-chain ML design. First, we detect private relay attack transactions and generalize the detection mechanism to unseen attacks in real time. Second,

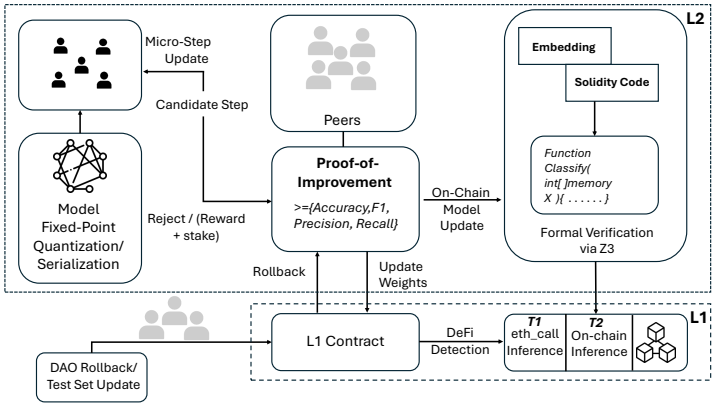


Figure 2 Framework overview.

we ensure bitwise consistency between off-chain and on-chain models, and cost-effective decentralized training and inference. We also guarantee bounded resource consumption, such as gas fees for training and inference. Furthermore, we introduce a decentralized mechanism that enables peers (i.e., DeFi platforms) to collectively train ML/DL models (e.g., for attack defense) by proposing training samples. These proposed updates are transparently governed, validated, and challengeable by others. An overview of our end-to-end framework is presented in Figure 2.

Our framework enables decentralized learning by translating traditional off-chain machine learning and deep learning model architectures into gas-efficient, formally verified Solidity contracts for blockchain execution via L2 rollups (e.g., Optimism [48]). It supports a wide range of architectures, from simple to complex, and produces tamper-proof ML/DL contracts with quantized parameters serialized for on-chain weight propagation (Figure 4). This capability not only enhances DeFi threat detection but also provides a foundation for developing adaptive and secure financial systems in other domains. Our approach addresses the limitations of prior work discussed in Section 2.2. Previous efforts (e.g., [42, 57]) are impractical and incur high gas costs when deploying complex models on L1, which can be very complex.

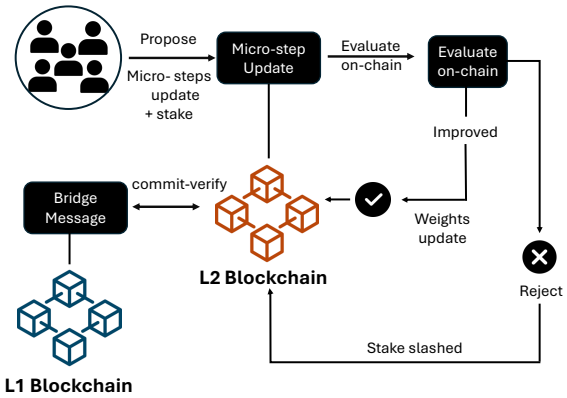


Figure 3 *PoIm* overview.

In contrast, our design supports full-scale, real-world models without compromising accuracy due to Solidity's floating-point limitations, and it enables fully decentralized training and model updates. To achieve this, we introduce Proof-of-Improvement (PoIm), a decentralized governance protocol that tracks each training step and governs model updates, as shown in Figure 3. It allows anyone (e.g., any DeFi platform) to incrementally train the model, effectively proposing updates that improve detection metrics. Updates are predictably propagated from L2, where computation and verification are performed, to L1, where inference takes place. Peers can collectively override a propagated model update on L1 in cases of suspected malicious training (poisoning) by another peer. Overall, our decentralized design is motivated by DeFi security requirements, and we show that a decentralized, dynamic, and cost-effective ML/DL-based defense is feasible under current blockchain architectures enabled by L2, capable of mitigating attacks that have caused billions in financial losses.

4.1 Decentralized Micro-Step Training and Model Evolution

Our system enables verifiable, decentralized training by decomposing the learning process into micro-step updates. Each micro-step corresponds to a single incremental improvement, such as retraining on one example or applying a localized adjustment, and is proposed directly on-chain. All proposals are immediately evaluated using a canonical, public test set stored on-chain.

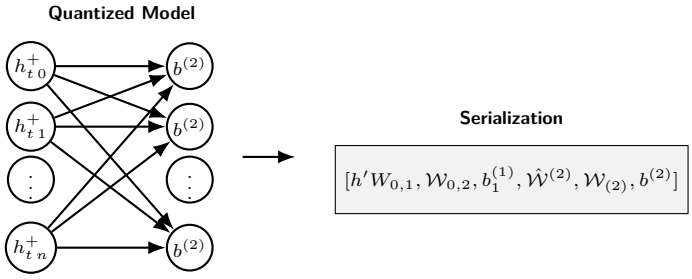


Figure 4 Serialization example of quantized model parameters.

Only updates that improve at least one core metric (accuracy, precision, recall, or F1-score) without degrading any others are accepted. This logic is enforced deterministically by the *PoIm* contract, which governs model evolution and ensures that every change is auditable, trustless, and resistant to adversarial manipulation.

Every accepted or rejected update, along with its evaluation results, is permanently logged on-chain. This guarantees transparent, tamper-proof model provenance and allows peers to track, audit, or challenge any step of the training process. In contrast to centralized retraining, this decentralized micro-step protocol allows the model to evolve continuously and securely, driven entirely by peer contributions and verified in real time on-chain.

4.2 Inference Architecture

In our architecture, we support two cost-effective tiers for executing ML/DL inference over blockchain networks. Each tier provides a different trade-off between execution cost, verifiability, and decentralization.

On-chain Logic, Off-chain Execution (zero-cost). In this tier, the model parameters and execution logic are stored fully on-chain. However, the actual inference computation is performed off-chain by calling `view` functions, which are executed by any EVM client. This tier incurs zero gas cost for DeFi users or protocols while ensuring that the decision logic is derived directly from verifiable on-chain bytecode and model state. Since every node executes the same bytecode deterministically, outcomes are consistent and tamper-resistant. This design is ideal for platforms seeking lightweight classification with full code transparency and no execution fees.

Fully On-chain Inference (inference verifiable on-chain). Here, the inference is executed entirely on-chain as part of a state-modifying transaction, typically one that interacts with a DeFi protocol. The input is passed to the smart contract, which executes the classifier internally and enforces the classification result. This enables end-to-end verifiability and enforces decisions during transaction execution (e.g., rejecting or allowing access to protocol funds). While this incurs gas costs, our optimized design allows even moderately complex models, such as 10-layer CNNs with quantized integer arithmetic, to execute efficiently within Ethereum's gas limits. This tier is suited for scenarios like high TVL or security-critical DeFi functions, where classification must be verified on-chain without relying on off-chain entities to interpret results, as the outcome is enforced by on-chain consensus.

4.3 Layer-2 to Layer-1 Computation Separation

Model parameters are updated through decentralized training and governed by the Proof-of-Improvement (PoIm) protocol on Layer-2 (L2), where gas costs are significantly lower. These L2-validated model parameters must then be securely and accurately propagated to Layer 1 (L1) for use by inference contracts in either Tier 1 or Tier 2. To ensure that L1 inference contract parameters do not get tampered with, we employ a commit-verify propagation mechanism.

Commit to Model Hash (L2). Upon acceptance of a training update (yielding new parameters θ') by the *PoIm* protocol on L2, the contract computes a cryptographic hash `modelHash = keccak256(abi.encodePacked(`θ'`))`. This hash serves as a tamper-proof commitment to the new model.

Transmit Commitment (L2 → L1). The L2 contract sends this `modelHash` to L1 via the native L2-to-L1 bridge (e.g., Optimism's `L2ToL1MessagePasser`). The hash is recorded by the L1 contract that manages model updates.

Transmit Parameters and Verify (L2 → L1). In a subsequent transaction, the full model parameters θ' are transmitted. The L1 contract recomputes the hash and verifies it against the prior commitment. A mismatch results in the rejection of the update [59].

4.4 Formal Bit-Exact Verification

Weights are scaled by $S \in [10^6, 10^{18}]$ then packed into `int32[]` and up-cast to `int128` where safe. A fully-connected layer l executes $z^{(l)} = \text{idiv}\big(\hat{W}^{(l)} z^{(l-1)} + \hat{b}^{(l)},\, S\big)$ If $\gamma > \Delta$, sign consistency holds for all validation inputs. For every compiled model, we prove $\forall x \in \mathbb{Z}^d$: $\mathcal{F}_{\text{on}}(x) = \mathcal{F}_{\text{off}}(x)$ under fixed-point scale S. We encode both paths as bit-vector (e.g., 256-bit) formulas and ask Z3 for `expr_on` $\neq$ `expr_off`. All models (linear, CNN, RNN) return `unsat`,

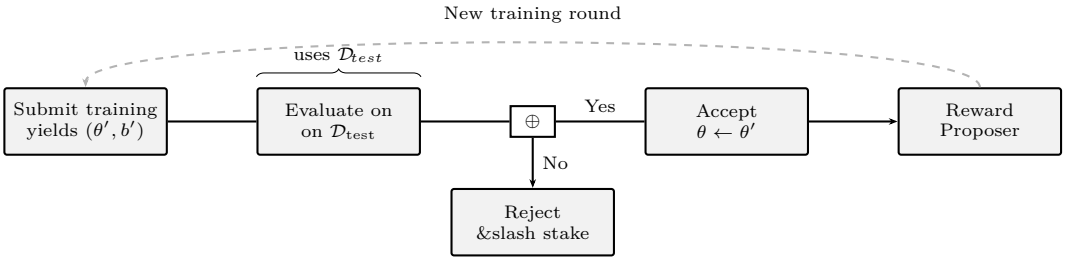


Figure 5 *PoIm* protocol update flow. Each proposed model is evaluated on $\mathcal{D}_{\text{test}}$ and accepted only if it improves performance.

giving a machine-checked guarantee of equality. This design-time formal proof offers a strong guarantee that the compiled on-chain model faithfully implements the intended off-chain model logic under the specified fixed-point representation for all possible inputs. It ensures the intrinsic correctness of the model's translation to Solidity. This is distinct from, yet complementary to, the operational consistency checks performed during the lifecycle of the model on-chain. Listing 1 provides an example of a forward pass implemented in Solidity using fixed-point arithmetic.

Listing 1 Forward pass example for layer l using fixed-point arithmetic

```
for (uint i = 0; i < d_l; i++) {
    z[i] = bias_l[i];
    for (uint j = 0; j < d_{l-1}; j++) {
        z[i] += idiv(weights_l[i * d_{l-1} + j] * input[j], SCALE);
    }
}
```

Beyond the formal verification of the model logic itself, our protocol incorporates runtime consistency checks at critical junctures, such as model updates on L2, propagation to L1, and sample inferences, to ensure operational integrity in runtime.

Table 3 Analytic MAC bounds for CNN and RNN models with total input dimension $d = 3$.

Model	F	K	Bound
$\text{CNN}_{2\times2}$	2	2	1 714
$\text{CNN}_{4\times2}$	4	2	3 428
$\text{CNN}_{8\times3}$	8	3	4 832

(1)

Model	U	T	Bound
$\text{RNN}_{4\times2}$	4	2	7 064
$\text{RNN}_{8\times4}$	8	4	40 160

(2)

4.5 Gas Cost and Runtime Bound Analysis

Opcode Budget. For every multiply-accumulate (MAC) $a \leftarrow a + \frac{w \cdot x}{S}$ inside `classify`, the EVM executes (1) $\texttt{SLOAD}(w) = G_S = 100$ gas (warm), (2) $\texttt{CALLDATALOAD}(x) = G_C = 3$ gas, (3) $\texttt{MUL} = G_M = 5$ gas, (4) $\texttt{DIV} = G_D = 5$ gas, (5) $\texttt{ADD} = G_A = 3$ gas, (6) loop/stack bookkeeping $\approx G_L = 8$ gas.[1]

[1] Berlin fee schedule [21].

Hence, the cost per MAC is $g_{\text{MAC}} = G_S + G_C + G_M + G_D + G_A + G_L = 124$ gas. ReLU adds $G_R = 5$ gas per activation, and bias initialization costs $G_S + G_A = 103$ gas.

Linear Classifiers (LR / SVM). A linear model with d inputs executes one MAC per feature and adds a bias. Hence, $\boxed{G_{\text{LIN}}(d) = 124\,d + 103}$

For logistic regression, we approximate $\sigma(z)$ by a threshold on the logit, therefore no extra exponentiation is incurred; the cost matches SVM. With d=3 (example feature set) the bound gives $G_{\text{LIN}}(3) = 124 \times 3 + 103 = 475$ gas, which is three orders of magnitude below the deep models and negligible at call-sites.

CNN Bound. For an input of length d, kernel size K, and F filters, the convolution yields $o = d - K + 1$ positions and evaluates $F\,o\,K$ MACs; the fully connected read-out contributes a further $F\,o$ MACs. Thus, $\boxed{G_{\text{CNN}}(d, K, F) = g_{\text{MAC}}\,F\,o\,(K+1) + G_R\,F\,o + 103\,F}$ (1).

RNN Bound. Let U be the hidden-state size, T the number of time steps, and $d_{\text{in}} = d/T$ the per-step input width. A gated update performs $U(d_{\text{in}} + U)$ MAC operations per time step. The total cost is $\boxed{G_{\text{RNN}}(d, U, T) = g_{\text{MAC}}\,T\,U(d_{\text{in}} + U + 1) + 15\,T\,U}$ (2). We fix the total input dimensionality $d = 3$ for both architectures. Table 3 shows the resulting gas bounds for several example CNN and RNN configurations.

4.6 Decentralized Model Update

Only models that show improvement in multiple evaluation metrics are accepted. No external oracle trust is required and all updates are verifiable on-chain within strict gas constraints.

Table 4 Notations used.

Symbol	Description
$\theta,\ \theta'$	current and proposed weight vectors
$b,\ b'$	current and proposed bias scalars
S	scaling constant, $S = 10^x$
$\mathcal{D}_{\text{test}} = \{(x_i, y_i)\}_{i=1}^{n}$	on-chain evaluation dataset
$x_i \in \mathbb{R}^d,\ y_i \in \{0, 1\}$	feature vector and label
$f_\theta(\cdot)$	classifier parameterized by (θ, b)
s	stake (in wei) deposited by proposer
$M, M', M_{\text{old}}, \widetilde{M}$	metric vectors before/after update
Δt	challenge window (e.g., 1-day)

Proof of Improvement (PoIm). Let f_θ denote the deployed classifier with an immutable architecture and on-chain weights θ (see Table 4 for notations). The core of our *PoIm* mechanism relies on $\mathcal{D}_{\text{test}} = \{(x_i, y_i)\}_{i=1}^{n}$, a canonical evaluation dataset stored directly on-chain. This dataset is intentionally curated to be compact, yet it is representative of critical attack vectors and desired model behaviors rather than being an exhaustive list of all historical transactions. Its manageable size is crucial for enabling efficient on-chain evaluation of proposed model updates within gas limits. Furthermore, to maintain its relevance and resist ossification, $\mathcal{D}_{\text{test}}$ itself is governed by a decentralized peer-based mechanism (e.g., DAO voting), allowing for agreed-upon additions, modifications, or removals of test samples over time (see Figure 5). Let θ' represent a submitted update. We define the classifier (i.e.,

linear) as $f_\theta(x) = \text{sign}\Big(\frac{1}{S}\sum_{i=1}^{d}\theta_i\, x_i + b\Big) \in \{0,1\}$. We define the evaluation function as $\text{Eval}(f_\theta, \mathcal{D}_{\text{test}}) \to (\text{Acc}, \text{F1}, \text{Prec}, \text{Rec})$, where all metrics are computed deterministically using Solidity logic and can be challenged.

New Training Submission. Users can propose a new model update by submitting a new training sample directly to the *PoIm* contract, which yields a change (if accepted) in new model weights θ' and biases b', and staking a minimum amount of tokens (or, e.g., any ERC20 token). Each submission must submit a stake, e.g., $s > 0$. The staked ERC20 serves as collateral for the proposal. If accepted, the contributor receives: $R = s + \sum_{k\in\{\text{Acc,F1,Prec,Rec}\}} \alpha_k\,(M'_k - M_k)$. Each coefficient $\alpha_i \geq 0$ reflects the vault's value weighting for each metric improvement. For instance, if the vault has accumulated 1 ETH from failed update attempts, the payout R is proportionally distributed based on the magnitude of improvement across the four metrics.

Model Update Acceptance. An update θ' is accepted if it improves at least one core metric (accuracy, precision, recall, or F1-score) without degrading any of the other core metrics compared to the current model, based on on-chain evaluation over $\mathcal{D}_{\text{test}}$. The contract enforces this by ensuring that any accepted model update demonstrates improvement in at least one core metric without degrading others, thereby preventing overfitting a single target (e.g., maximizing recall while degrading precision). This improvement is verified on-chain on $\mathcal{D}_{\text{test}}$, which ensures that the proposed model performs better than the current one.

Test Set and Adversarial Update Rollback. We integrate an on-chain DAO mechanism that allows stakers (e.g, DeFi platforms) to collectively manage both model rollbacks and $\mathcal{D}_{\text{test}}$ test set updates. In our design, participants stake tokens to gain voting rights to propose and vote on critical actions such as adding or modifying test samples and reverting model weights (to previously committed L1 update) if suspicious behavior is observed (e.g., malicious training sample). In the case of a challenged update (i.e., one that introduces a loophole but still satisfies the acceptance criteria), a revealed update can be rolled back within a fixed period (e.g., <7 days). If the new model weights worsen in any performance metric compared to the previous metrics, the proposer loses their collateral and the proposal is discarded.

5 Evaluation

This section details our experimental setup, dataset construction, model configurations, and metrics used to evaluate our proposed framework for on-chain DeFi exploit detection and mitigation across various machine learning architectures. We also present a quantitative comparison against a baseline for on-chain ML.

5.1 Experimental Setup

All experiments were run on a machine equipped with an Intel Core i7 CPU and 16GB of RAM, running Windows 10. Key Python libraries we utilized include `scikit-learn` [39], `pandas` [45] (e.g., for data manipulation), `NumPy` [34] (e.g., for numerical operations), `web3.py` (e.g., for blockchain interaction), `py-solc-x` (for Solidity compilation), and the `Z3-solver` [17] (e.g., for formal verification logic).

For blockchain interactions, local L1 and L2 test environments were used. The L2 environment was configured using a Hardhat Network instance forking a live Optimism L2 rollup state (e.g., Optimism Mainnet), facilitating realistic gas calculations and execution

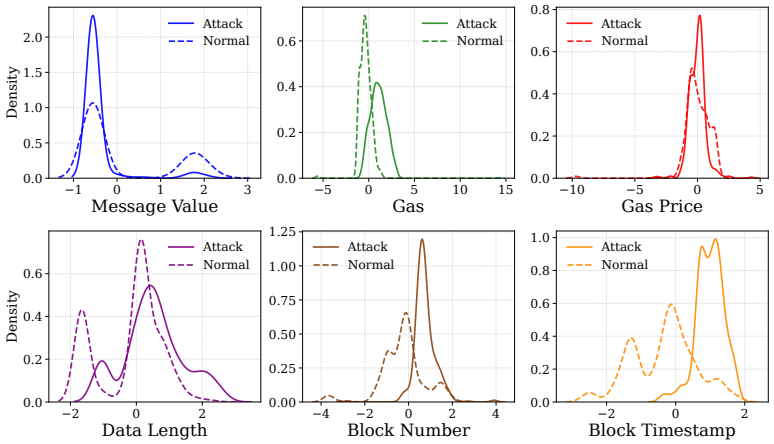


Figure 6 Distributions of normal and attack transactions.

behavior mirroring Optimism's characteristics. This L2 environment is primarily used for the decentralized training and *PoIm* mechanism. The L1 environment was simulated using Anvil[27], configured for persistence across experimental runs to reflect a stable mainnet-like chain.

Attack Dataset Construction. A significant challenge in evaluating DeFi exploit detection systems is the general absence of comprehensive, publicly available, raw transaction datasets suitable for behavioral modeling. While some prior work has focused on smart contract code analysis [69, 64, 66], readily usable transactional datasets for exploit detection are scarce. To address this, we undertook a meticulous manual collection and multi-stage verification process to construct a robust dataset for this study, covering attacks from 2020 to 2025. Our initial identification of potential exploits involved a broad survey of diverse sources, including industry security news (e.g., Rekt News [18]), detailed analyses on technical blogs, discussions on social media platforms (e.g., X), and curated public incident databases such as the DeFi Rekt Database [18] and DeFiLlama Exploits Dashboard [19]. Incidents and leads gathered from these channels, often further indexed or summarized by resources like DeFiHackLabs [58], guided our targeted retrieval of the specific transactions that executed each confirmed attack.

These transactions are primarily identified by their hashes on public blockchain explorers (e.g., Etherscan [22], Polygonscan [53]). All transactional records were independently retrieved and subsequently verified through direct blockchain queries via Web3 RPCs [14] to ensure authenticity and completeness of call data, receipts, and traces. The primary features extracted for model input include: `gas` (gas limit provided by sender), `block.timestamp`, `func_selector_encoded` (a label-encoded representation of the first four bytes of `msg.data`), `chain` ID (label-encoded), `msg.sender` (label-encoded), `tx.origin` (label-encoded), and `msg.to` (label-encoded), as shown in Figure 6. Numerical features were standardized, and categorical features were label-encoded. Features such as `block.timestamp` may overfit if the same timestamp is found in both training and testing. To mitigate this risk, we implemented a strict temporal train-test split. Training is exclusively on historical data, and testing is on future (unseen attacks). Therefore, the model generalizes better based on transaction behaviors rather than memorizing temporal artifacts. Transaction traces and

contract interactions were analyzed to produce contextual annotations, including `attack_name`, links to exploited contract source files, incident report URLs, dates, identified root causes, and financial loss in USD (normalized to the time of exploit). This process was applied to transactions across multiple EVM-compatible blockchains, including *Ethereum, Binance Smart Chain (BSC), Polygon, Avalanche, Arbitrum, Fantom, Moonriver, and Base.* Table 5 presents an illustrative subset of these attributes.

Table 5 Example of a subset of fields from our exploit transaction dataset. Full transaction records include additional attributes such as detailed EVM context, blockchain metadata, and semantic annotations.

Field	Example	Description
`tx_hash`	0x78d7...2df4	Unique identifier of the exploit transaction
`msg.sender`	0x5aa5...30b7	Address that initiated the exploit
`msg.value`	0	ETH directly transferred in the transaction
`gas_used`	298210	Total gas consumed during execution
`block.timestamp`	1688462834	Unix timestamp of the exploit
`root_cause`	Unchecked external call	Vulnerability exploited in the contract
`loss_USD`	$1.94M	Financial loss via unauthorized token transfers
`chain`	Ethereum	Blockchain on which the exploit occurred

* *Only a subset of fields is shown here.*

We observed that an attack can manifest in a single atomic transaction (approximately 71% of the attacks in our dataset) or across multiple transactions. For multi-transaction attacks, all constituent transactions were grouped and assigned entirely to either the training or testing set to prevent data leakage and maintain the integrity of the attack sequence. The combined dataset of normal and attack transactions was then sorted chronologically by `block.timestamp`. Table 6 summarizes the distribution of the unique exploits that informed the construction of our dataset.

Table 6 Distribution of *unique* exploits (2020- 2025) and for DeFi financial losses.

Category	Count	Loss (USD)
Total distinct exploits	298	–
Exploits in training set	202	$1,877,229,549.86
Exploits in testing set	96	$1,858,400,900.66
Total exploits loss	–	**$3,735,630,450.52**

The resulting transaction data captures fine-grained, executable-level behaviors and intricate attacker-victim dynamics. This detailed resolution enables precise detection and classification of exploits and provides a foundation for exploring potential attack discovery. The final curated dataset of 298 unique attack vectors, corresponding to confirmed financial losses exceeding $3.74 billion (as per Table 6), combined with temporally aligned normal transactions, offers a rich and unique foundation for evaluating real-time defense mechanisms and rigorously training on-chain classifiers.

Model Architectures. Our framework was evaluated with diverse model architectures, including Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and standard classifiers like Logistic Regression (LogReg), Support Vector Machines (SVM), and Decision Trees (DTs). All models utilized the 7 processed

input features detailed previously. These features are observable by smart contracts during execution, enabling real-time attack detection. All models were evaluated with fixed-point parameters (scale 10^1 to 10^{18}).

Proof-of-Improvement (PoIm) Protocol. Each instance starts from baseline parameters and their performance metrics (Accuracy, F1-score, Precision, Recall) on a fixed test set ($\mathcal{D}_{\text{test}}$). An update is accepted only if it improves at least one metric without degrading others. Approved parameters are then eligible for propagation to L1 for inference.

Metrics. The performance of our framework was evaluated across three key dimensions: on-chain inference efficiency (gas costs, bytecode size, throughput, and prediction consistency), attack detection performance (accuracy, precision, recall, F1-score, and false positive rate), and model update mechanism costs (*PoIm* L2 update gas and L2-L1 propagation cost).

5.2 Performance of On-Chain DeFi Exploit Detection

We evaluate the effectiveness of on-chain machine learning classifiers in detecting previously unseen DeFi exploits. Each model was tested on a hold-out test set $\mathcal{D}_{\text{test}}$ comprising real-world attacks and benign transactions, temporally separated from the training set to simulate generalization to zero-day attacks. The unseen exploits in the test set include unique, protocol-specific attacks from various root causes that are absent from the training set.

We tested a diverse set of models: Logistic Regression (LogReg), Support Vector Machine (SVM), Decision Tree (DT), Multi-Layer Perceptron (MLP), 15 CNN variants (filters $F \in \{2, 4, 8, 10, 16\}$, kernel sizes $K \in \{1, 4, 5\}$), and RNNs with 8 units and with 1 and 7 timesteps (T=1 and T=7). All models utilized a shared 7-feature input vector, preprocessed via standardization and label-encoding as described in Section 5.1.

The fully trained CNN variants demonstrated strong detection capabilities, particularly in terms of recall. Many configurations achieved recall ≥ 0.96, with F1-scores generally ranging from $\sim$0.78 to 0.82, and precision reaching up to 0.8077 (CNN (F4, K1)). For instance, CNN(F4, K4) achieved a high accuracy of 0.9004 and an F1-score of 0.8200, preventing an estimated \$1,857.6M in losses. This performance marks a significant improvement over any preliminary simulations where simpler CNN setups might have exhibited degenerate behavior.

RNN models also showed robust and balanced performance. Specifically, RNN(U=8, T=1) achieved an accuracy of 0.8517 with a high recall of 0.9792, contributing to \$1,858.2M in prevented losses. The RNN(U=8, T=7) configuration maintained competitive performance, notably achieving higher precision (0.6607) and a lower false positive count (58 FPs) compared to RNN(U=8, T=1), with a slight trade-off in recall (0.9479).

Among other classifiers, LogReg and SVM performed well, with SVM achieving the highest AUC in this set (0.9739) and LogReg attaining perfect recall (1.0000). The Multi-Layer Perceptron (MLP) also demonstrated strong results, with an accuracy of 0.8665, F1-score of 0.7823, and a high recall of 0.9688. The DecisionTree, while achieving perfect recall, did so at the cost of a higher number of false positives (224 FPs), indicating overfitting to the attack class.

Table 7 On-Chain costs, sizes, and *PoIm* dynamics for LogReg, SVM, DT, MLP, CNN, and RNN models. **Ext. cl.:** inference by an EVM node.

Model	Deploy L2 (Gas)	Deploy L1 (Gas)	L1 Inf. Gas (Gas)	L2 Size (Bytes)	L1 Size (Bytes)	L2-L1 (PoIm) (Gas)	Ext. cl.* ($)
LogReg	1,185,010	722,020	57,603	4,597	3,044	231,168	0
SVM	1,184,998	722,020	57,603	4,597	3,044	231,168	0
DT	878,284	435,490	33,414	3,352	1,707	96,489	0
MLP	2,137,045	1,116,415	138,173	7,266	4,866	5,754,732	0
CNN(F2, K1)	1,721,688	1,536,046	143,647	6,692	5,834	2,872,940	0
CNN(F4, K1)	2,134,766	1,949,124	244,284	6,692	5,834	4,885,680	0
CNN(F8, K1)	2,961,086	2,775,539	445,562	6,692	5,834	8,911,240	0
CNN(F10, K1)	3,374,357	3,188,810	546,202	6,692	5,834	10,924,040	0
CNN(F16, K1)	4,614,143	4,428,596	848,126	6,692	5,834	16,962,520	0
CNN(F2, K4)	1,721,688	1,536,046	158,856	6,692	5,834	3,177,120	0
CNN(F4, K4)	2,134,778	1,949,136	274,703	6,692	5,834	5,494,060	0
CNN(F8, K4)	2,961,098	2,775,551	506,397	6,692	5,834	10,127,940	0
CNN(F10, K4)	3,374,369	3,188,822	622,244	6,692	5,834	12,444,880	0
CNN(F16, K4)	4,614,167	4,428,620	969,788	6,692	5,834	19,395,760	0
CNN(F2, K5)	1,721,688	1,536,046	150,304	6,692	5,834	3,006,080	0
CNN(F4, K5)	2,134,778	1,949,136	257,598	6,692	5,834	5,151,960	0
CNN(F8, K5)	2,961,086	2,775,539	472,188	6,692	5,834	9,443,760	0
CNN(F10, K5)	3,374,345	3,188,798	579,482	6,692	5,834	11,589,640	0
CNN(F16, K5)	4,614,143	4,428,596	901,368	6,692	5,834	18,027,360	0
RNN(U8, T1)	3,520,668	3,338,663	561,791	7,413	6,571	11,235,820	0
RNN(U8, T7)	2,437,361	2,255,356	1,131,350	7,413	6,571	22,627,000	0

**Read-only RPC calls consume 0 gas on-chain.*

5.3 Efficiency, Cost, and Consistency

We analyze the on-chain operational costs, resource utilization, and behavioral consistency of the evaluated machine learning models. All quantitative data discussed refer to Table 7, which details L2 and L1 deployment gas, L1 inference gas, contract bytecode sizes, and key Proof-of-Improvement (PoIm) interaction costs where applicable.

Deployment and Inference Gas Costs. The on-chain footprint of simpler models like Logistic Regression (LogReg), Support Vector Machines (SVM), Decision Trees (DT), and a Multi-Layer Perceptron (MLP) showcases varying efficiencies. L2 *PoIm* contract deployment gas ranged from approximately 0.88M for the DT to 2.14M for the MLP, while their corresponding L1 inference contracts were lighter. Notably, L1 inference gas for LogReg, SVM, and DT was very low (33k–58k gas). The evaluated MLP required approximately 138k gas for L1 inference, still well within practical limits for on-chain execution.

Convolutional Neural Network (CNN) variants demonstrated a clear trend where both L2/L1 deployment gas and L1 inference gas scaled with architectural complexity, primarily driven by the number of filters (F) and kernel size (K). For instance, L2 deployment gas ranged from approximately 1.72M for CNN(F2, K1) up to 4.61M for larger configurations like CNN(F16, K1). Similarly, L1 inference gas for these CNNs varied from around 144k (CNN(F2, K1)) to over 969k (CNN(F16, K4)). Bytecode sizes for CNN L1 inference contracts were observed to be consistent for models sharing the same kernel size, as changes in filter count primarily affect the parameter size of the subsequent fully connected layer rather than the convolutional logic structure itself.

For the Recurrent Neural Network (RNN) models evaluated (both with U=8 units), the T=1 configuration (processing all 7 features in a single timestep) incurred higher L2 deployment gas (3.52M) compared to the T=7 configuration (processing 1 feature per 7 timesteps, 2.44M L2 gas). This difference is likely attributable to the larger input-to-hidden

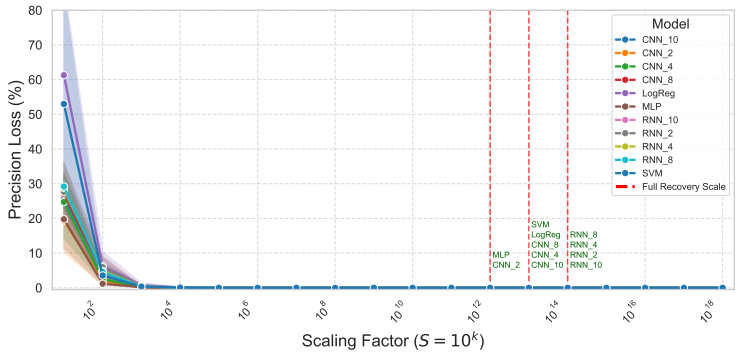


Figure 7 Bitwise consistency for different scaling factors of various models.

weight matrix (W_{xh}) in the RNN(U=8, T=1) model. Conversely, L1 inference gas was substantially higher for RNN(U=8, T=7) (1.13M gas) than for RNN(U=8, T=1) (0.56M gas), reflecting the increased number of recurrent steps executed on-chain for the sequence-based input.

The gas cost for transferring updated model parameters from an L2 *PoIm* contract to its L1 inference counterpart varied across model types. For SVM and DT, this L2-L1 update was relatively efficient (96k–231k gas). However, the LogReg and the dynamically trained MLP exhibited higher transfer costs (5.2M–5.7M gas, respectively), likely due to the encoding or size of their complete parameter sets being transferred. For the more complex CNN and RNN models, this L2-L1 parameter transfer for their (typically output layer) updates also showed significant gas consumption, ranging from 2.8M to over 22M gas, underscoring the cost implications of updating larger or more intricate models across layers.

Bitwise Consistency Verification. We evaluate empirically how fixed-point quantization affects on-chain inference consistency under varying scaling factors from 10^1 to 10^{18}, using MLP, logistic regression (LogReg), SVM, CNN, and RNN models (Figure 7). Precision loss diminishes rapidly as the scale increases, and full recovery (zero bitwise loss across all weights) is achieved at or above 10^{12} for all models. This confirms that fixed-point quantization, when aligned with sufficiently large scaling factors, can reliably preserve inference semantics on-chain across diverse architectures, which was also evaluated in terms of the models' performance.

***PoIm* Resilience Under Adversarial Training.** We present *PoIm* resistance against malicious updates via stress testing. We consider bootstrapping the decentralized model with 50 real training samples from both classes, and injecting 50% of the training data size as malicious or fabricated updates. These malicious updates either flip the class label or inject feature-noise. The 50 real samples are randomly selected from the original attack and normal transaction data to quantify how *PoIm* reacts when training samples degrade one or more metrics, and whether such poisoning affects performance. At each new training step, the update is re-evaluated on the full test set. If accepted by *PoIm*, the model is updated; otherwise, the current model is retained. We note that the bootstrapping phase influences how *PoIm* responds to subsequent updates. Overall, *PoIm* maintains more stable performance than the original (unfiltered) training process across all metrics on the same test data, as shown in Figure 8. Green vertical bands indicate accepted training samples, while reddish bands denote rejected ones. Notably, original linear models (e.g., logistic regression, SVM) suffer severe degradation without *PoIm* filtering.

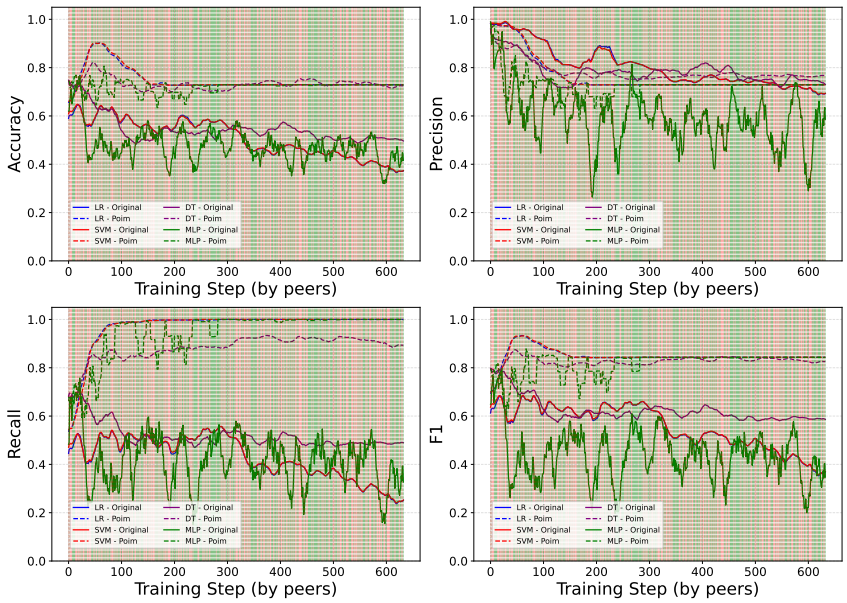


Figure 8 *PoIm* performance under 50% adversarial training samples. Green vertical lines indicate accepted updates. Red lines indicate rejected ones. *PoIm* stabilizes performance compared to the original training.

Inference Throughput. To evaluate real-time classification capability for the zero-cost `eth_call` tier, we measured inference throughput using our 10-layer CNN. Table 8 details these findings, which were obtained by sequentially querying a local Ethereum node. A single transaction is classified in approximately 68ms. While the average per-sample processing time for batches of 5 to 50 transactions stabilizes around 88-91ms, this performance enables a throughput of approximately 11 classifications per second by an off-chain monitoring entity. Such capacity is comparable to Ethereum's current average TPS (≈ 15), suitable for timely analysis.

Table 8 Inference throughput per second for (e.g., 10-layer) CNN for different transactions as batches. Ethereum's current TPS ≈ 15.

Batch Size	Total Time (s)	Avg Time/Sample (s)
1	0.0680	0.0680
5	0.3890	0.0778
10	0.9150	0.0915
20	1.7650	0.0883
50	4.5580	0.0912

5.4 Baseline Comparison

We compare our framework against ML2SC [42], which compiles MLP models into smart contracts for on-chain inference. While training in ML2SC is centralized, on-chain ML studies remain sparse. Our evaluation focuses on gas costs and contract behavior. We re-deployed all ML2SC MLP contracts, but observed that their design targets batch processing over a fixed internal dataset of 50 samples. Specifically, their `classify()` function returns an aggregate result (e.g., count of correct classifications), whereas our framework supports single-instance inference.

Table 9 Our framework vs. Ml2SC as a baselines [42].

Metric	Our Framework	Baseline Method (ML2SC [42])
Deployment Gas (Contract + Params)	1.07M - 2.17M gas	1.96M - 2.36M gas (contract code only)
Separate Parameter Setting Gas	0 gas	0.30M - 1.04M gas
Data Setting Gas	N/A	~**10.63M gas** (for 50 internal samples)
Total Setup Gas	**1.07M - 2.17M gas**	**~12.9M - ~14.0M gas**
Bytecode Size (Bytes)	~4,224 bytes	~9,425 - ~11,151 bytes
Inference Gas (classification on chain L1)	**81k - 261k gas (per call)**	**~2.0M - ~2.4M gas (for 50 internal samples)**
Output Type	Prediction (0 or 1)	Batch Result (e.g., count like 0 or 14)

Baseline `classify()` output (e.g., 14 (for MLP 1 layer and 1 neuron), 0 for others) indicates the number of correct predictions from its internal batch.

Deployment and setup costs differ significantly between our approach and the baseline. Our models embed parameters directly via the constructor and become operational with 1.07M–2.17M gas. In contrast, ML2SC baselines incur higher cumulative setup costs (~12.9M–14.0M gas) due to separate contract deployment (1.96M–2.36M gas), parameter setting (0.30M–1.04M gas), and dataset population (~10.63M gas for 50 samples), as detailed in Table 9. For inference, our framework supports efficient single-input predictions (81k–261k gas) with measurable throughput (~7.6–8.0 calls/sec), whereas baseline contracts consume ~2.0M–2.4M gas per call, reflecting batch evaluation of 50 internal samples. Their `view/pure classify()` functions are not suited for single-instance transactional inference. Finally, our contracts are smaller in size (~4,224 bytes) compared to baseline contracts (~9,425–11,151 bytes). The results demonstrate the efficiency and suitability of our framework for single-instance, real-time ML inference on-chain, especially in terms of gas cost and inference flexibility.

6 Discussion

Transaction Class Imbalance Mitigation. In our decentralized training, the model is updated incrementally by peers through submitted training samples. If normal transactions exceed attack samples by a large margin threshold (e.g., 5x), *PoIm* blocks further normal samples. Only attack samples are accepted until the balance is restored. This prevents skewed updates, which would decrease attack transaction detection accuracy.

Trust and Inference Verification. Our system is designed under the assumption that decentralized participants (i.e., DeFi platforms) perform training on an L2 network. However, we do not assume these participants are honest. To defend against adversarial training data injection, we incorporate per-sample verification and adversarial robustness checks at each training step. Specifically, after each submitted training sample, the *PoIm* evaluates the model against a fixed, immutable (agreed upon by peers) test set using four metrics: accuracy, precision, recall, and F1 score. The update is accepted only if it maintains or improves at least one of these metrics without degrading any others. Otherwise, the training step is discarded and the data point is excluded from the model. This method is effective in both cases: when malicious training samples are used or when honest user data fails to improve the model's performance. It ensures that even if malicious actors attempt to inject poisoned or manipulative data, their contributions cannot degrade the classifier's performance. Furthermore, all model training steps are conducted on L2 via a verifiable smart contract, and every accepted update is auditable through its associated state transition and event log. This guarantees transparency and accountability for each model state change. Once a model has reached a finalized state on L2 (e.g., by consensus or performance threshold), its quantized weights are serialized and committed via a cryptographic hash. This hash is

propagated to L1 along with the raw weights. The L1 inference contract is a static model (non-trainable) that accepts the weights only if the hash matches, ensuring the integrity of the L2-originated model and protecting the L1 blockchain from tampering or weight substitution attacks. Thus, our architecture enforces correctness and robustness both during training (via metric-based rejection of adversarial updates) and during propagation (via hash-based integrity verification), with no reliance on external oracles, centralized validators.

Inference Tiers. Our two-tier inference architecture imposes a trade-off for protocol designers. The zero-cost tier uses `eth_call` execution. It can be used for protocols that do not need on-chain inference verifiability, yet get the on-chain verified data, such as wallet interfaces, warning users of potentially malicious transactions without incurring gas fees. For fully verified on-chain defense, protocols integrate the fully on-chain tier, which acts as a gatekeeper (e.g., IPS) by embedding the classification logic interface directly within a state-modifying transaction.

Future Work. The *PoIm* protocol only accepts updates that yield metric improvements. This approach might lead to convergence to a local optimum, where no single micro-update can further improve the model, even if a better global solution exists. Future work could explore other acceptance criteria, such as stochastic policies like simulated annealing [32], which would permit occasional slightly degrading steps to encourage broader exploration of the model space. Also, *PoIm*, similar to other stake-based governance systems, could be susceptible to centralization if voting rights, represented by tokens (e.g., linear voting), are openly tradable. A malicious actor could accumulate enough stake to influence the update of the test set or roll back stable updates. Mitigating this, for example, through quadratic voting or identity-based mechanisms such as know-your-customer (KYC) [23, 24], is a potential avenue for future work. Our framework assumes a decentralized and engaged set of participants (e.g., DeFi protocols themselves) whose long-term incentive is to ensure model integrity to protect individual DeFi protocols' funds. Finally, future work may study sophisticated adversarial strategies. This includes addressing game-theoretic risks such as front-running and detecting if latent backdoors in model updates exist.

7 Related Work

On-chain AI research seeks transparent, tamper-proof inference but faces the EVM's fixed-point arithmetic and gas limits [30, 44, 57, 42, 26]. Translators from ML to smart contract code such as ML2SC compile MLPs from PyTorch to Solidity, proving feasibility for small models yet incurring high gas per call on complex networks [57, 42]. For DeFi exploit detection, LookAhead [54], STING [67], and FlashGuard [5] inspect mempools or historical transactions to flag and mitigate attacks. Off-chain placement of these systems introduces latency [67], misses private-relay transactions, and introduces centralized control. Off-chain ML with on-chain verification is another direction of research. zkML [11], for example, attaches zk-SNARK [51] proofs to each inference, preserving privacy but multiplying compute and memory requirements by orders of magnitude [49, 65, 29].

Our cryptographic verification is distinct from zkML systems. ZKML frameworks use zero-knowledge proofs (e.g., zk-SNARKs) to verify that a specific computation, such as an ML inference, was executed honestly with a private model [11]. This provides computational integrity but has substantial overhead. For instance, ZKML needs powerful hardware (up to 1TB of RAM for a distilled GPT-2 model) and can have proving times of nearly an hour [11].

In contrast, our framework employs a much simpler and more gas-efficient commit-verify scheme. Our approach prioritizes provenance and data integrity over computational privacy since the DeFi attacks are public. This makes our approach practical for low-cost, real-time use and suitable for mitigating DeFi attacks.

opML [15] (fraud-proof) treats results as valid unless a verifier proves otherwise, reducing prover cost at the price of economic guarantees. However, it does not provide cryptographic security [49, 16]. Agatha applies similar fraud proofs to DNNs on Ethereum [70]. Proposals for decentralized model marketplaces, federated learning with ZK privacy, and DAO-based model governance [7] either offload heavy compute or evaluate on limited node sets [15]. In contrast, our approach is end-to-end, fully audited inference while remaining within L2 and L1 mainnet gas limits.

8 Conclusion

We presented a fully decentralized and verifiable, on-chain ML/DL framework for real-time DeFi exploit detection. Our approach enables classification of transactions at execution time using a deterministic, gas-free inference mechanism embedded in smart contracts. We proposed Proof-of-Improvement (PoIm), a decentralized, stake-based model update protocol that accepts only provably superior updates. The system guarantees inference consistency, bounded gas usage, and resistance to adversarial submissions. Empirical evaluation on 298 real-world DeFi exploits indicates high detection performance and practical feasibility. This work establishes a new model for integrating ML-driven defenses into DeFi protocols with minimal latency, overhead, and maximal decentralization.

References

1 Hedgey finance. `https://hedgey.finance/`, 2024. Accessed: 2024-05-08.

2 Martín Abadi. Tensorflow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN international conference on functional programming*, pages 1–1, 2016.

3 Sa'ed Abed, Reem Jaffal, Bassam J Mohd, and Mohammad Al-Shayeji. An analysis and evaluation of lightweight hash functions for blockchain-based iot devices. *Cluster computing*, 24:3065–3084, 2021. `doi:10.1007/S10586-021-03324-1`.

4 Alchemy. Web3 Development Platform. `https://www.alchemy.com`, 2024.

5 Abdulrahman Alhaidari, Balaji Palanisamy, and Prashant Krishnamurthy. Poster: Flashguard: Real-time disruption of non-price flash loan attacks in defi. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*, page 3, Salt Lake City, UT, USA, october 14–18 2024. ACM. `doi:10.1145/3658644.3691385`.

6 Abdulrahman Alhaidari, Balaji Palanisamy, and Prashant Krishnamurthy. Protecting defi platforms against non-price flash loan attacks. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy (CODASPY '25)*, pages 1–12, Pittsburgh, PA, USA, 2025. ACM. `doi:10.1145/3714393.3726503`.

7 Dana Alsagheer, Lei Xu, and Weidong Shi. Decentralized machine learning governance: Overview, opportunities, and challenges. *IEEE Access*, 11:96718–96732, 2023. `doi:10.1109/ACCESS.2023.3311713`.

8 Raphael Auer, Bernhard Haslhofer, Stefan Kitzler, Pietro Saggese, and Friedhelm Victor. The technology of decentralized finance (defi). *Digital Finance*, 6(1):55–95, 2024.

9 Nic Carter and Linda Jeng. Defi protocol risks: The paradox of defi. *Regtech, suptech and beyond: innovation and technology in financial services" riskbooks–forthcoming Q*, 3, 2021.

10 Stefanos Chaliasos, Marcos Antonios Charalambous, Liyi Zhou, Rafaila Galanopoulou, Arthur Gervais, Dimitris Mitropoulos, and Benjamin Livshits. Smart contract and defi security tools: Do they meet the needs of practitioners? In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–13, 2024. `doi:10.1145/3597503.3623302`.

11 Bing-Jyue Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 560–574, 2024. doi:10.1145/3627703.3650088.

12 Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Computing Surveys (CSUR)*, 53(3):1–43, 2020. doi:10.1145/3391195.

13 Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. *Cryptology ePrint Archive*, 2024.

14 Web3.js Contributors. Web3.js, 2024. URL: https://github.com/web3/web3.js.

15 KD Conway, Cathie So, Xiaohang Yu, and Kartin Wong. opml: Optimistic machine learning on blockchain. *arXiv preprint arXiv:2401.17555*, 2024. doi:10.48550/arXiv.2401.17555.

16 Sourav Das, Vinay Joseph Ribeiro, and Abhijeet Anand. Yoda: Enabling computationally intensive contracts on blockchains with byzantine and selfish nodes. *arXiv preprint arXiv:1811.03265*, 2018. arXiv:1811.03265.

17 Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

18 De.Fi. Rekt db, 2024. Accessed: 2024-06-07. URL: https://de.fi/rekt-database.

19 DeFiLlama. DeFi Dashboard. https://defillama.com, 2024. Accessed: 2024-05-14.

20 Kaustubh Dwivedi, Ankit Agrawal, Ashutosh Bhatia, and Kamlesh Tiwari. A novel classification of attacks on blockchain layers: Vulnerabilities, attacks, mitigations, and research directions. *arXiv preprint arXiv:2404.18090*, 2024. doi:10.48550/arXiv.2404.18090.

21 Ethereum Foundation. Ethereum improvement proposals. https://eips.ethereum.org/, 2025.

22 Etherscan. Ethereum blockchain explorer. http://etherscan.io/, 2024.

23 Andres Fabrega, Amy Zhao, Jay Yu, James Austgen, Sarah Allen, Kushal Babel, Mahimna Kelkar, and Ari Juels. Voting-Bloc Entropy: A New Metric for DAO Decentralization. In *2025 USENIX Security Symposium (USENIX Security 25)*, 2025.

24 Rainer Feichtinger, Robin Fritsch, Lioba Heimbach, Yann Vonlanthen, and Roger Wattenhofer. Sok: Attacks on daos. In *6th Conference on Advances in Financial Technologies (AFT 2024)*, pages 28–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.AFT.2024.28.

25 Flashbots. Mitigating MEV in Blockchain. https://www.flashbots.net, 2024. Accessed: 2024-07-23.

26 Ethereum Foundation. Ethereum, 2024. URL: https://ethereum.org/.

27 Foundry. Blazing fast, portable and modular toolkit for Ethereum application development written in Rust. https://getfoundry.sh, 2024.

28 Rundong Gan, Liyi Zhou, Le Wang, Kaihua Qin, and Xiaodong Lin. Defialigner: Leveraging symbolic analysis and large language models for inconsistency detection in decentralized finance. In *6th Conference on Advances in Financial Technologies (AFT 2024)*, pages 7–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.AFT.2024.7.

29 Bianca-Mihaela Ganescu and Jonathan Passerat-Palmbach. Trust the process: Zero-knowledge machine learning to enhance trust in generative ai interactions. *arXiv preprint*, 2024. doi:10.48550/arXiv.2402.06414.

30 Caleb Geren, Amanda Board, Gaby G Dagher, Tim Andersen, and Jun Zhuang. Blockchain for large language model security and safety: A holistic survey. *ACM SIGKDD Explorations Newsletter*, 26(2):1–20, 2025. doi:10.1145/3715073.3715075.

31 Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16, 2016. doi:10.1145/2976749.2978341.

32 Thomas Guilmeau, Emilie Chouzenoux, and Víctor Elvira. Simulated annealing: A review and a new scheme. In *2021 IEEE statistical signal processing workshop (SSP)*, pages 101–105. IEEE, 2021. doi:10.1109/SSP49050.2021.9513782.

33 Yang Guo. A review of machine learning-based zero-day attack detection: Challenges and future directions. *Computer communications*, 198:175–185, 2023. doi:10.1016/J.COMCOM.2022.11.001.

34 Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020. doi:10.1038/S41586-020-2649-2.

35 Ningyu He, Lei Wu, Haoyu Wang, Yao Guo, and Xuxian Jiang. Characterizing code clones in the ethereum smart contract ecosystem. In *International Conference on Financial Cryptography and Data Security*, pages 654–675. Springer, 2020. doi:10.1007/978-3-030-51280-4_35.

36 Lioba Heimbach and Roger Wattenhofer. Eliminating sandwich attacks with the help of game theory. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 153–167, 2022. doi:10.1145/3488932.3517390.

37 Safak Kayikci and Taghi M Khoshgoftaar. Blockchain meets machine learning: a survey. *Journal of Big Data*, 11(1):9, 2024. doi:10.1186/S40537-023-00852-Y.

38 Nikhil Ketkar. Introduction to keras. In *Deep learning with python: a hands-on introduction*, pages 97–111. Springer, 2017.

39 Oliver Kramer and Oliver Kramer. Scikit-learn. *Machine learning for evolution strategies*, pages 45–53, 2016.

40 Wenkai Li, Xiaoqi Li, Yuqing Zhang, and Zongwei Li. Defitail: Defi protocol inspection through cross-contract execution analysis. In *Companion Proceedings of the ACM on Web Conference 2024*, pages 786–789, 2024. doi:10.1145/3589335.3651488.

41 Xiaofan Li, Jin Yang, Jiaqi Chen, Yuzhe Tang, and Xing Gao. Characterizing ethereum upgradable smart contracts and their security implications. In *Proceedings of the ACM Web Conference 2024*, pages 1847–1858, 2024. doi:10.1145/3589334.3645640.

42 Zhikai Li, Steve Vott, and Bhaskar Krishnamachari. Ml2sc: Deploying machine learning models as smart contracts on the blockchain. In *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 645–649. IEEE, 2024. doi:10.1109/ICBC59979.2024.10634431.

43 Yulin Liu, Yuxuan Lu, Kartik Nayak, Fan Zhang, Luyao Zhang, and Yinhong Zhao. Empirical analysis of eip-1559: Transaction fees, waiting times, and consensus security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2099–2113, 2022. doi:10.1145/3548606.3559341.

44 Rischan Mafrur. Ai-based crypto tokens: The illusion of decentralized ai? *IET Blockchain*, 5(1):e70015, 2025. doi:10.1049/BLC2.70015.

45 Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

46 Johnnatan Messias, Vabuk Pahari, Balakrishnan Chandrasekaran, Krishna P Gummadi, and Patrick Loiseau. Dissecting bitcoin and ethereum transactions: On the lack of transaction contention and prioritization transparency in blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 221–240. Springer, 2023. doi:10.1007/978-3-031-47751-5_13.

47 Nexus Mutual. CREAM Finance Hack. https://nexusmutual.io/claims-stories/cream-finance-hack, 2025. Accessed: 2025-02-05.

48 Optimism Foundation. Optimism. https://www.optimism.io/, 2025. Accessed: 2025-02-21.

49 Zhizhi Peng, Taotao Wang, Chonghe Zhao, Guofu Liao, Zibin Lin, Yifeng Liu, Bin Cao, Long Shi, Qing Yang, and Shengli Zhang. A survey of zero-knowledge proof based verifiable machine learning. *arXiv preprint arXiv:2502.18535*, 2025.

50 Daniel Perez and Benjamin Livshits. Smart contract vulnerabilities: Vulnerable does not imply exploited. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1325–1341, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/perez.

51 Maksym Petkus. Why and how zk-snark works. *arXiv preprint arXiv:1906.07221*, 2019. arXiv:1906.07221.

52 Valentina Piantadosi, Giovanni Rosa, Davide Placella, Simone Scalabrino, and Rocco Oliveto. Detecting functional and security-related issues in smart contracts: A systematic literature review. *Software: Practice and experience*, 53(2):465–495, 2023. doi:10.1002/SPE.3156.

53 PolygonScan. Polygon Gas Tracker. https://polygonscan.com/gastracker, 2024. Accessed: 2024-08-30.

54 Shoupeng Ren, Lipeng He, Tianyu Tu, Di Wu, Jian Liu, Kui Ren, and Chun Chen. Lookahead: Preventing defi attacks via unveiling adversarial contracts. *arXiv preprint arXiv:2401.07261*, 2024.

55 Sarwar Sayeed, Hector Marco-Gisbert, and Tom Caira. Smart contract: Attacks and protections. *Ieee Access*, 8:24416–24427, 2020. doi:10.1109/ACCESS.2020.2970495.

56 Cosimo Sguanci, Roberto Spatafora, and Andrea Mario Vergani. Layer 2 blockchain scaling: A survey. *arXiv preprint arXiv:2107.10881*, 2021. arXiv:2107.10881.

57 Nikumbh Sarthak Sham, Sandip Chakraborty, and Shamik Sural. Generation of optimized solidity code for machine learning models using llms. *arXiv preprint arXiv:2503.06203*, 2025. doi:10.48550/arXiv.2503.06203.

58 SunWeb3Sec. DeFiHackLabs. https://github.com/SunWeb3Sec/DeFiHackLabs, 2024. Accessed: 2024-05-08.

59 Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. In *Aspects of Computation and Automata Theory with Applications*, pages 377–424. World Scientific, 2024.

60 Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10:93039–93054, 2022. doi:10.1109/ACCESS.2022.3200051.

61 Xiaojie Wang, Hanxue Li, Ling Yi, Zhaolong Ning, Song Guo, and Yan Zhang. A survey on off-chain networks: Frameworks, technologies, solutions and challenges. *arXiv preprint arXiv:2311.10298*, 2023. doi:10.48550/arXiv.2311.10298.

62 Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

63 Zihuan Xu and Lei Chen. L2chain: Towards high-performance, confidential and secure layer-2 blockchain solution for decentralized applications. *Proceedings of the VLDB Endowment*, 16(4):986–999, 2022. doi:10.14778/3574245.3574278.

64 Chavhan Sujeet Yashavant, Saurabh Kumar, and Amey Karkare. Scrawld: A dataset of real world ethereum smart contracts labelled with vulnerabilities. *arXiv preprint arXiv:2202.11409*, 2022. arXiv:2202.11409.

65 Zheming Ye, Xiaodong Qi, Zhao Zhang, and Cheqing Jin. Yoimiya: A scalable framework for optimal resource utilization in zk-snark systems. *arXiv preprint arXiv:2502.18288*, 2025. doi:10.48550/arXiv.2502.18288.

66 Pengcheng Zhang, Feng Xiao, and Xiapu Luo. A framework and dataset for bugs in ethereum smart contracts. In *2020 IEEE international conference on software maintenance and evolution (ICSME)*, pages 139–150. IEEE, 2020. doi:10.1109/ICSME46990.2020.00023.

67 Zhuo Zhang, Zhiqiang Lin, Marcelo Morales, Xiangyu Zhang, and Kaiyuan Zhang. Your exploit is mine: Instantly synthesizing counterattack smart contract. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1757–1774, 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-zhuo-exploit.

68 Zhuo Zhang, Brian Zhang, Wen Xu, and Zhiqiang Lin. Demystifying exploitable bugs in smart contracts. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 615–627. IEEE, 2023. doi:10.1109/ICSE48619.2023.00061.

69 Zibin Zheng, Jianzhong Su, Jiachi Chen, David Lo, Zhijie Zhong, and Mingxi Ye. Dappscan: building large-scale datasets for smart contract weaknesses in dapp projects. *IEEE Transactions on Software Engineering*, 2024.

70 Zihan Zheng, Peichen Xie, Xian Zhang, Shuo Chen, Yang Chen, Xiaobing Guo, Guangzhong Sun, Guangyu Sun, and Lidong Zhou. Agatha: Smart contract for dnn computation. *arXiv preprint arXiv:2105.04919*, 2021. arXiv:2105.04919.

71 Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. Sok: Decentralized finance (defi) attacks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2444–2461. IEEE, 2023. doi:10.1109/SP46215.2023.10179435.