

Optimal Reward Allocation via Proportional Splitting

Lukas Aumayr  

Common Prefix, Athens, Greece
University of Edinburgh, UK

Zeta Avarikioti  

Common Prefix, Athens, Greece
TU Wien, Austria

Dimitris Karakostas  

Common Prefix, Athens, Greece

Karl Kreder  

Dominant Strategies, Austin, TX, USA

Shreekara Shastry  

Dominant Strategies, Austin, TX, USA

Abstract

Following the publication of Bitcoin’s arguably most famous attack, selfish mining, various works have introduced mechanisms to enhance blockchain systems’ game-theoretic resilience. The only proof-of-work reward rule with a Nash-equilibrium guarantee, FruitChains, demands reward finality on the order of days. The rules that settle in minutes have no such guarantee, and one of them, Reward Splitting, still outperforms FruitChains on most of the metrics that matter in deployment. This paper closes that gap between theory and practice.

We introduce **FairChain**, a two-level transformation for any proof-of-work Nakamoto-style protocol. At the protocol layer, **FairChain** records low-difficulty samples called *workshares* alongside blocks. At the reward layer, it applies *Proportional Reward Splitting (PRS)*: each height’s reward is divided among the competing work objects in proportion to the intrinsic work behind them, with workshares supplying a fresh power estimate at every height. The fork-choice rule and block-production loop are left untouched, so the host chain’s security carries over unchanged. Workshares can be discarded once the corresponding rewards mature, leaving zero on-chain footprint.

We prove **FairChain** is a ρ -coalition-safe ϵ -Nash equilibrium for sufficiently large parameters, matching FruitChains in theory. To evaluate practical performance, we leverage Markov decision processes and compute the optimal adversarial policy under each utility function, rather than the gain of any one attack. At a six-block confirmation window, **FairChain** raises the deviation threshold to 38% of mining power and beats every mechanism in that framework on incentive compatibility, subversion gain (except FruitChains above 42%), and censorship susceptibility (except FruitChains below 25%). Our construction has been adopted by a live PoW blockchain, with sub-kilobyte per-block storage overhead in deployment.

2012 ACM Subject Classification Security and privacy → Security protocols; Security and privacy → Distributed systems security; Security and privacy → Social aspects of security and privacy

Keywords and phrases blockchain, proof-of-work, fairness, game-theory, rewards

Digital Object Identifier 10.4230/LIPIcs.AFT.2026.30

Related Version *Full Version*: <https://arxiv.org/abs/2503.10185> [1]

Supplementary Material *Software (Source Code)*: <https://github.com/commonprefix/proportional-reward-splitting-MDP> [10]

archived at `swb:1:dir:4a244dd0f68a10c4abb1fa0bf218527ba36dd085`



© Lukas Aumayr, Zeta Avarikioti, Dimitris Karakostas, Karl Kreder, and Shreekara Shastry; licensed under Creative Commons License CC-BY 4.0
8th Conference on Advances in Financial Technologies (AFT 2026).

Editors: Aggelos Kiayias and Maria Kyropoulou; Article No. 30; pp. 30:1–30:23



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This work was partially supported by Quai, Common Prefix, the Austrian Science Fund (FWF) through the SFB SpyCode project F8512-N, and the Vienna Science and Technology Fund (WWTF) through the projects 10.47379/ICT25056 and 10.47379/ICT22045, and by Input Output (<https://iohk.io>) through their funding of the Edinburgh Blockchain Technology Lab.

1 Introduction

Proof-of-work (PoW) blockchains pay miners to secure the ledger, and the rule that allocates these payments decides whether honest participation is rational. Selfish mining [7] was the first attack to make this precise: by withholding blocks and releasing them strategically, a miner with a minority of the hashing power can capture a share of the rewards exceeding its share of the power. The lasting contribution of that work was the lens rather than the strategy. It analyzed Bitcoin under proportional, rather than absolute, rewards, and showed that under this utility Bitcoin is not an equilibrium. Subsequent work optimized both the attack and its analysis [20, 9, 15]. Recent events have underscored that these incentives are not merely theoretical. In August 2025, the Qubic mining pool conducted a documented attack against the PoW-based Monero network, causing multiple chain reorganizations and triggering emergency mitigation proposals from the Monero research community [17].

FruitChains [16] answered selfish mining at the level of mechanism design. It guarantees *fairness*: every miner receives rewards approximately proportional to its power, so deviations are unprofitable. The mechanism rewards low-difficulty objects, called fruits, alongside blocks. Fruits are mined far more frequently than blocks and therefore sample the mining-power distribution more finely, so that rewards track a miner’s power rather than the variance in block discovery. FruitChains is a ρ -coalition-safe ϵ -Nash equilibrium, the strongest game-theoretic guarantee in this line of work. The guarantee, however, holds only asymptotically. Fairness requires every fruit to be recorded within a window of $R \cdot \kappa$ blocks, with recency parameter $R = 17$ and security parameter κ , which places reward finality on the order of days. For any deployment where reward finality must be fast, this is not a usable regime.

The distance between what is provable and what is deployable was made explicit by Zhang and Preneel [25], who evaluated Nakamoto consensus [14], FruitChains [16], Reward Splitting [25, 13, 4], and Subchains [19] under a single framework. They measured three quantities: (a) incentive compatibility, the fairness of the reward distribution; (b) subversion gain, the profitability of double spending; and (c) censorship susceptibility, the income an attacker can deny honest miners. At realistic parameters, FruitChains was not the best mechanism on most of these metrics. *No prior mechanism in this literature simultaneously achieves asymptotic equilibrium guarantees and state-of-the-art practical performance.*

This paper closes that gap. We present *Proportional Reward Splitting (PRS)*, a reward mechanism that is a ρ -coalition-safe ϵ -Nash equilibrium for large parameters, matching FruitChains in theory, and that achieves state-of-the-art practical performance at a six-block confirmation window, outperforming every mechanism of [25] on incentive compatibility and, with limited exceptions, on subversion gain and censorship susceptibility.

The design begins with separating two roles that existing mechanisms conflate. On the one hand, Reward Splitting [25, 13, 4] makes attacks less profitable by compensating non-canonical work, but it uses a coarse 50–50 split whenever competing objects exist at the same height. On the other hand, FruitChains obtains a finer estimate of mining power from low-work samples, but its theoretical parameters are too large for practical finality. PRS combines these ideas by using low-work objects, which we call *workshares*, only as a local estimator for proportional splitting: it keeps the anti-forking effect of rewarding

competing work, while replacing the coarse split by the measured mining-power distribution. For example, an adversary contributing 30% of the power at a height receives approximately 30% of its reward. This advantage is largest when the adversary's power is meaningfully below one half and when the workshare sample is accurate; as the adversary approaches one half, the proportional split naturally converges to ordinary Reward Splitting.

PRS could be applied to any PoW Nakamoto-style blockchain (e.g., Bitcoin [14], PoEM [12], Monero [22]) by making two changes: blocks record workshares on-chain, and the reward rule is replaced with proportional splitting. Neither change affects the fork choice rule or block production, so security follows directly from the host protocol's guarantees. We call the resulting transformation **FairChain**. We further provide an optimization in which workshares can be kept off-chain, thereby eliminating any on-chain costs of our transformation. Thus, the storage overhead is minimal: workshare data can be discarded once the corresponding rewards finalize.

To evaluate practical performance, we follow the framework of Zhang and Preneel [25]. For each reward mechanism, we construct a Markov decision process (MDP) whose states encode the adversary's private chain, the public chain, the current fork status, and the recent reward history. Solving this MDP yields the optimal adversarial policy for each utility objective, rather than measuring the performance of a fixed attack strategy.

Under this evaluation, **FairChain** both increases the mining-power threshold beyond which an adversary profits from deviating and lowers its expected gain from doing so. For incentive compatibility, **FairChain** requires an adversary to exceed 38% of the power to earn more than its fair share, against 35% for the next-best mechanism, Reward Splitting. At 38% power, an adversary earns 38% of all rewards under **FairChain**, against 40% under Reward Splitting and more than 50% under Bitcoin or FruitChains. On subversion gain, **FairChain** matches or beats every alternative across the practical adversarial range, with a strict gap over every other mechanism between 30% and 38% power. On censorship susceptibility, it is the best mechanism above 25% power and within two points of the best below. These advantages hold at every workshare and fork-eligibility window we test.

Finally, we note that the reward mechanism introduced in this paper has been adopted in a deployed system: a live PoW blockchain based on PoEM [12],¹ which records workshares on-chain and allocates rewards proportionally to workshare-estimated mining power. Our discussion reports its workshare cap, freshness window, staleness discount, and resulting sub-kilobyte per-block storage overhead. These parameters ground **FairChain** in a deployed setting and show that the workshares required by **FairChain** incur only small overhead.

Summary of Contributions and Roadmap. We make the following contributions:

- We introduce **FairChain**, a protocol transformation that efficiently incorporates workshares in any PoW Nakamoto-style blockchain without changing the fork choice rule or block production. **FairChain** uses these workshares to realize PRS, the first reward allocation mechanism for PoW blockchains that is both incentive-compatible and practical for real-world parameters (Section 3).
- We prove that **FairChain** is a ρ -coalition-safe ϵ -Nash equilibrium for sufficiently large parameters, matching the asymptotic guarantee of FruitChains (Section 4).
- We evaluate **FairChain** via Markov decision processes under practical parameters, comparing against prior mechanisms on incentive compatibility, subversion gain, and censorship susceptibility (Section 5).

¹ <https://www.qu.ai>

■ **Table 1** Summary of notation.

Symbol	Description
<i>Operators and sequences</i>	
$a.b$	element b of a (complex) object a
$A \parallel A'$	concatenation of A and A'
$ \cdot $	length of an element (list items or bits)
$L \prec L'$	L is a prefix of L'
<i>Chains and blocks</i>	
$C[i]$	i -th block of a chain C ($C[-1]$ is the tip)
$C[:i], C[-i:]$	first (resp. last) i blocks of C
B_{tx}	block in which transaction tx is published
view	execution-trace view
<i>Parties and power</i>	
n	total number of parties
$t = \rho n$	number of adversarial parties
ρ	adversarial mining power, as a fraction of the total
<i>Mining and timing</i>	
Q	hashing-oracle queries per party per round
p	probability that a player mines a block in a round
Δ	network delay, in rounds
w	wait-time (liveness) parameter
k	common-prefix (safety) parameter
<i>Security parameters</i>	
κ	security parameter
λ	output length of $H(\cdot)$, in bits

2 Model and Notation

This section first summarizes the notation (Table 1) and cryptographic primitives and then outlines the execution, cryptographic, and game-theoretic models of our analysis.

2.1 Notation and Cryptographic Primitives

Table 1 summarizes the notation used throughout the paper. Workshare-specific parameters are introduced where they first appear, in Sections 3 and 4.

We rely on two cryptographic primitives. The function $H(\cdot)$ is a hash function that maps an arbitrary bitstring to a value of λ bits and models a random oracle.² The function $d(\cdot)$ maps a list of arbitrary bitstrings to a value of λ bits and is collision resistant; it can be instantiated e.g., via a Merkle tree, in which case $d(\cdot)$ returns the tree's root.

² H should satisfy the standard hash function properties, namely collision, preimage, and second-preimage resistance.

2.2 Execution Model

Our analysis assumes a multi-party setting, following the model of Bitcoin Backbone [8]. An “environment” program $\mathcal{Z}$ drives the execution of a protocol Π , by spawning for each party $\mathcal{P}$ an instance of an “interactive Turing machine” (ITM) which executes the protocol. Interaction between parties is controlled by a program C , such that $(\mathcal{Z}, C)$ create a system of ITMs. Our analysis applies to “locally polynomially bounded” systems, ensuring polynomial time execution.

Network. The execution proceeds in (a polynomial number of) rounds of size Δ . In each round r , each party is activated and performs various operations based on its algorithm. An honest message produced at round r is delivered to all other parties at the beginning of round $r + 1$, i.e., our analysis assumes a *synchronous network*. Finally, we assume a diffuse functionality, i.e., a gossip protocol which allows the parties to share messages without the need for a fully connected graph.

Parties. The set of n parties $\mathbb{P}$ remains fixed for the duration of the execution. Each party $\mathcal{P}$ can make a fixed number of queries to a hashing oracle, which enables participation in the distributed ledger protocol. The adversary controls at most $t = \rho \cdot n$ parties, out of the total n at any given time, and its power is the sum of the corrupted parties’ power.

Proof-of-Work. All parties have access to a random oracle $\mathcal{O}_{\text{rnd}}$. Each party can perform Q queries to the oracle in each round, while the adversary can perform $t \cdot Q$ queries per round. Additionally, the protocol defines a *difficulty parameter* p , which denotes the probability that a player successfully creates a block in a round. In practice, the random oracle is instantiated as a hash function $H(\cdot)$. We assume that the codomain of H is large enough such that the outcomes of queries within a batch are practically independent.

2.3 Cryptographic Model

In the cryptographic model, some parties are assumed honest (i.e., they follow the protocol) whereas the rest are Byzantine (i.e., they act arbitrarily). Note that incentives do not play a role in the security analysis under this model, and instead will be considered in the game theoretic analysis.

Adversary. The adversary A is an ITM which, upon activation, may “corrupt” a number of parties by sending a relevant message to the controller after $\mathcal{Z}$ instructs it to do so. Subsequently, when a corrupted party is supposed to be activated, A is activated instead. Furthermore, A is “adaptive”, i.e., corrupts parties on the fly, and “rushing”, i.e., decides its strategy and acts after observing the other parties’ messages.

Randomized Execution. The execution of a protocol is probabilistic, with randomness coming from both honest parties and the adversary, denoted as A , which controls all corrupted nodes, as well as the environment Z , which provides inputs to honest nodes throughout the protocol’s execution.

We denote a randomly sampled execution trace as $\text{view} \stackrel{\$}{\leftarrow} EXEC^{\Pi}(A, Z, \kappa)$, where $|\text{view}|$ represents the number of rounds in the execution trace. Specifically, view is a random variable that captures the joint perspective of all parties, encompassing their inputs, random coins, and received messages (including those from the random oracle). This joint view uniquely determines the entire execution.

Restrictions on the Environment. Constraints on (A, Z) [8]. The environment Z and the adversary A must respect certain constraints. We say that a p.p.t. pair (A, Z) is (n, ρ, Δ) -respecting w.r.t. Π , iff for every $\kappa \in \mathbb{N}$, every view in the support of $EXEC^\Pi(A, Z, \kappa)$, the following holds:

- Z activates n parties in view;
- For any message broadcast by an honest player at any time t in view, any player that is honest at time $t + \Delta$ or later must have received the message. This means that in the case of newly spawned players, it instantly delivers messages that were sent more than Δ rounds ago. As long as this Δ constraint is respected, A is allowed to delay or reorder honest players' messages arbitrarily.
- At every round r in view, A controls at most $\rho \cdot n$ parties.

Let $\Gamma(\cdot, \cdot, \cdot)$ be a boolean predicate. We say that a p.p.t. pair (A, Z) is Γ -compliant w.r.t. protocol π iff: (i) (A, Z) is (n, ρ, Δ) -respecting w.r.t. π ; (ii) $\Gamma(n, \rho, \Delta) = 1$.

2.4 Game Theoretic Model

In the game theoretic model, we assume that all parties are rational. Specifically, each party $\mathcal{P}$ employs a strategy S , which is a set of rules and actions that the party makes depending on its input. In other words, S defines the part of the distributed protocol that $\mathcal{P}$ performs.

Each party has a well-defined utility U and chooses a strategy S with the goal of maximizing their utility. The literature of game theoretic analyses of blockchain protocols considers various types of utilities [2]: (i) absolute rewards; (ii) absolute profit, i.e., absolute rewards minus cost; (iii) relative rewards, i.e., the percentage of the party's reward among all allocated rewards; (iv) relative profit, i.e., relative rewards minus (absolute) cost; (v) deposits and penalties, e.g., using a slashing mechanism; (vi) external rewards, e.g., bribes.

A strategy profile σ is a vector of all parties' strategies and it is an ϵ -Nash equilibrium if no party can increase its utility more than ϵ by unilaterally changing its strategy (Definition 1).

► **Definition 1** (ϵ -Nash equilibrium). Let $\mathbb{P}$ be the set of parties, $\mathbb{S}$ the set of available strategies, and $\sigma = \langle S_i, S_{-i} \rangle$ a strategy profile, where S_i is the strategy of $\mathcal{P}_i$ and S_{-i} are the strategies of all other parties. Let $U_i(\sigma)$ denote the utility of $\mathcal{P}_i$ under σ . The profile σ is an ϵ -Nash equilibrium with respect to $\bar{U} = \langle U_1, \dots, U_{|\mathbb{P}|} \rangle$ if

$$\forall \mathcal{P}_i \in \mathbb{P} \forall S'_i \in \mathbb{S} : U_i(\langle S_i, S_{-i} \rangle) \geq U_i(\langle S'_i, S_{-i} \rangle) - \epsilon.$$

3 The FairChain Protocol

FairChain is a two-level transformation on any proof-of-work longest-chain protocol. At the *protocol level*, it adds the recording of low-work objects, called *workshares*, on-chain alongside blocks. At the *reward level*, it replaces the existing allocation rule with proportional splitting: each height's reward is distributed among all competing objects at that height in proportion to their intrinsic work. Neither modification touches the host protocol's fork-choice rule or block-production mechanism; as such, FairChain inherits the host protocol's security guarantees.

FairChain Overview. The intrinsic work of any hash-based object obj is

$$\text{WORK}(\text{obj}) = \lambda - \lfloor \log H(\text{obj}) \rfloor, \quad (1)$$

the number of leading zero bits in its hash: the more zero bits, the higher the work. FairChain uses this measure to estimate each miner's relative power via workshares. A workshare has the same structure as a block and is produced by the same mining loop, but satisfies a lower work threshold. Parties propagate workshares through the network and publish them in blocks, as they do with transactions. Since workshares and blocks are outputs of the same hash query, a miner neither chooses between the two nor knows in advance whether a query yields a workshare, a block, or neither; the hash function models a random oracle. Recording workshares is the only modification FairChain makes to the host protocol. Because workshares affect neither fork choice nor block production, FairChain inherits the host protocol's security properties unchanged.

At a high level, in each round a miner runs the PoW loop; a successful query yields a workshare, a block, or both, depending on the leading zero bits of its hash. The miner broadcasts every new object, and whoever mines the next block records the workshares and uncles it has seen inside that block. Once the relevant blocks are stable, the work recorded for a given height, across its block and the workshares pointing to it, determines how that height's reward is split among miners (Section 3.4). Workshares thus serve one purpose: they sample mining power between blocks, so that rewards track power closely.

The accuracy of this power estimate is not a fixed per-block quantity: it depends jointly on the workshare rate, the block-production rate, and the reward-maturity window. A deployment can improve accuracy by increasing the workshare rate, by extending the maturity window, or both, without requiring all samples to appear in a single block.

In the remainder of this section, we describe the protocol's data objects, its core algorithm, the chain-selection and ledger-extraction mechanisms, the reward allocation rule, and an optimization that removes the on-chain storage overhead.

3.1 Blocks and Workshares

A work object in FairChain has the following format:

$$\text{WorkObject} = \langle (d(\text{TX}); d(\text{WS}); h_{B'}; h_{\hat{B}}), \text{ctr} \rangle$$

where TX is the transaction list, WS is the list of work objects (workshares or uncles) published when WorkObject acts as a block,³ $h_{B'}$ and $h_{\hat{B}}$ reference two prior blocks via their hashes, and ctr is the PoW nonce. A full block is the work object together with the explicit lists TX and WS. When an object is interpreted as a workshare, $d(\text{TX})$ and $d(\text{WS})$ are ignored; those fields are used only by the chain-selection and ledger-extraction mechanisms. Let R denote the recency window: a workshare or uncle is eligible for inclusion only if the block it references is at most R blocks behind the including block.

In more detail, the validity rules for blocks and workshares are as follows. In addition, we assume the existence of a transaction validity predicate, which specifies the application logic of the ledger, e.g., preventing double spending.

Block validity. $B = ((d(\text{TX}); d(\text{WS}); h_{B'}; h_{\hat{B}}), \text{ctr}), \text{TX})$ is valid w.r.t. a chain C if the following conditions hold:

- $d(\text{TX})$ is the digest of TX.
- $h_{B'}, h_{\hat{B}}$ are references to $C[-1], C[-k]$ respectively.

³ An uncle satisfies the block work threshold but is not part of the canonical chain, i.e., it has been forked out.

30:8 Optimal Reward Allocation via Proportional Splitting

- Every Workobject (workshare or uncle) in WS is valid w.r.t. C (see below);
- No Workobject is present in C multiple times.
- Every transaction in TX satisfies the protocol's validity predicate.
- $WORK(B) > T_{\text{block}}$, where T_{block} is a protocol parameter that defines the PoW lower bound for a block to be valid.

Uncle validity. $B = (\langle d(TX); d(WS); h_{B'}; h_{\bar{B}} \rangle, \text{ctr}, TX)$ is a valid uncle w.r.t. chain C , if all of the following hold:

- B is included in WS of exactly one block $\bar{B}$ in C and $B \notin C$;
- $h_{B'}$ is a reference to $C'[-1]$, which in turn is either a valid block in C or a valid uncle w.r.t. C ;
- $h_{\bar{B}}$ is a reference to $C'[-k]$, which is a valid block in C ;
- Let $\bar{C}$ be the prefix of C that goes up to this block $\bar{B}$. $|C'|$ is between $|\bar{C}| - R$ and $|\bar{C}| - 1$;
- Following the reference $h_{B'}$ to a block B' , and then this block's reference and so on at most k times, gives a block that is part of C ;
- $WORK(B) > T_{\text{block}}$.

Workshare validity. $ws = \langle d(TX); d(WS); h_{B'}; h_{\bar{B}} \rangle$ is valid w.r.t. chain C if:

- ws is included in WS of exactly one block $\bar{B}$ in C ;
- Let $\bar{C}$ be the prefix of C that goes up to this block $\bar{B}$. $h_{B'}$ is a reference to any valid block or uncle with a height between $|\bar{C}| - R$ and $|\bar{C}| - 1$;
- $WORK(ws) > T_{\text{ws}}$, where T_{ws} is a protocol parameter that defines the PoW lower bound for a workshare to be valid.

Intuitively, T_{ws} and T_{block} define the minimum amount of intrinsic work that a workshare or block should have, respectively, and it holds that $T_{\text{ws}} \leq T_{\text{block}}$.

Chain validity. Finally, a chain C is valid w.r.t. a genesis block G (denoted $\text{VALIDATE}_G(C)$) if every block in C is valid and they form a chain that starts from G .

3.2 Main Protocol

The main function of FairChain is defined in Algorithm 1. First, the party is constructed using the CONSTRUCTOR function (Line 3). In every round, each party is executed by the environment using the function EXECUTE (this function is due to the round-based nature of our time model). The algorithm maintains a local chain C , which is initialized to the genesis block G , and proceeds as follows.

Chains. The party first receives from the network the set of available chains via the function RECEIVE (Line 8). Out of all the available chains, including the local chain C , the party adopts the chain selected by the host protocol's chain-selection rule, denoted MAXVALID.

Mempool. The party receives the set of transactions in the mempool TX via the function INPUT (Line 10) and attempts to publish them in the newly-mined block. Specifically, the party orders the transactions in TX in an arbitrary manner in order to form the list TX ; for ease of notation, we abstract the ordering process outside of the protocol of Algorithm 1. The list's digest $d(TX)$ is included in the new block's header.

Algorithm 1 The FairChain protocol.

```

1:  $G \leftarrow \emptyset$ 
2:  $C \leftarrow []$ 
3: function CONSTRUCTOR( $G'$ )
4:    $G \leftarrow G'$  ▷ Select Genesis Block
5:    $C \leftarrow [G]$  ▷ Add Genesis Block to start of chain
6:    $\text{round} \leftarrow 1$ 
7: function EXECUTE $_{d,H}(1^\kappa)$ 
8:    $\mathbb{C} \leftarrow \text{RECEIVE}()$  ▷ Receive chains from the network
9:    $C \leftarrow \text{MAXVALID}(\{C\} \cup \mathbb{C})$  ▷ Apply host protocol's chain-selection rule
10:   $\langle \text{TX}, \text{WS} \rangle \leftarrow \text{INPUT}()$ 
11:   $\mathbb{B} \leftarrow \emptyset$ 
12:   $\text{ctr} \xleftarrow{\$} \{0, 1\}^\lambda$ 
13:  for  $i \leftarrow 1$  to  $Q$  do
14:     $\text{WorkObject} \leftarrow \langle (d(\text{TX}); d(\text{WS}); H(C[-1]); H(C[-k])), \text{ctr} \rangle$ 
15:    if  $\text{WORK}(\text{WorkObject}) > T_{\text{ws}}$  then
16:       $\text{WS} \leftarrow \text{WS} \parallel \text{WorkObject}$ 
17:    if  $\text{WORK}(\text{WorkObject}) > T_{\text{block}}$  then
18:       $B \leftarrow \langle \text{WorkObject}, \text{TX} \rangle$ 
19:       $\mathbb{B} \leftarrow \mathbb{B} \parallel B$ 
20:       $C \leftarrow C \parallel B$ 
21:     $\text{ctr} \leftarrow \text{ctr} + 1$ 
22:     $\text{DIFFUSE}(C, \text{WS} \parallel \mathbb{B})$ 
23:     $\text{round} \leftarrow \text{round} + 1$ 

```

Workshares. The party receives from the network workshares WS , including uncle blocks (Line 10), and attempts to publish them in the new block. As opposed to transactions, the order of workshares does not matter, since they are only used for reward allocation (cf. Section 3.4).

We note that TX and WS are both sanitized to contain valid objects w.r.t. the party's local chain C , so that the transactions and workshares can be published in a new block. For ease of notation we omit this process from Algorithm 1 and assume that TX and WS (obtained from INPUT) are sanitized w.r.t. the local chain C at all rounds. In particular, all workshares fulfill the PoW inequality, there are no duplicate workshares in the chain, and uncle blocks are not part of the chain.

Proof-of-Work loop. The main part of the protocol is the PoW loop, during which the party *potentially* creates workshares and blocks. Specifically, on each round the party can execute Q queries to the hash function. For each query, the work object includes a PoW nonce ctr and, depending on the amount of work that the object has, it may be a valid workshare and/or block. If a workshare is created, then it is immediately inserted into the list WS . This has the implication that a workshare which is created on some round r may be published on a block that is created on the same round r . Similarly, if a block is created, it is immediately appended to the local chain C . Regardless of whether a workshare or block is created, the party exhausts all available hashing queries at their disposal without interrupting the PoW loop. Therefore, a party that follows the protocol may create multiple workshares and/or blocks on a single round.

Finally, the party diffuses to the network their local chain and list of workshares and proceeds to the next round.

3.3 Chain Selection and Ledger Extraction

Chain selection follows the host protocol's MAXVALID rule. Ledger extraction is parameterized by the safety parameter k . Given a chain C , the last k blocks are excluded from the ledger extraction process. The remaining blocks in C are parsed sequentially, starting from the genesis block G , and the ledger L is formed by concatenating each transaction in the list TX of each block.

3.4 Proportional Reward Splitting Mechanism

We now specify the reward rule of FairChain. At each block height, PRS distributes a fixed reward among all eligible work objects at that height in proportion to their intrinsic work. This refines Reward Splitting [25, 13, 4], which also distributes a height's reward across competing objects but uses a fixed 50-50 split regardless of the work behind them. PRS replaces this coarse split with the measured power distribution: a miner contributing 30% of the work at a height receives 30% of that height's reward.⁴ The estimate is recomputed at every height to track fluctuations in mining power as they occur.

Eligibility windows. Two parameters control which objects are eligible for rewards at a given height (Figure 1). The *recency window* R sets the finalization delay: the reward for height h is settled at height $h + R$, and only work objects published within that window are counted. The *fork eligibility window* k limits which forks contribute: an object is eligible only if the block it references lies at most k blocks into a fork, excluding objects that build on deeply invalid chains. In our analysis, k is set to the common-prefix (consistency) parameter T_0 of the host protocol (cf. [1]).

Reward rule. At height $h + R$, each eligible work object at height h receives a fraction of the height's reward equal to its intrinsic work divided by the total intrinsic work of all eligible objects at that height. The height of a workshare or uncle is the height of the block it references via $h_{B'}$, not the block in which it is published. Maturity delays of this kind are standard: Bitcoin requires 100 blocks before a coinbase transaction can be spent.⁵

3.5 Removing on-chain storage overhead

FairChain does not modify the MAXVALID fork-choice rule of the underlying proof-of-work chain, so work objects play no role in chain selection, only in reward allocation. Instead of recording them on-chain within blocks, it suffices that miners share them over the P2P network. Miners will verify reward (coinbase) transactions based on the work objects sent along with the block. Once the rewards for height h_i are finalized at height $h_i + R$ (cf. Figure 1), all work objects older than $R + k$ blocks behind the current tip can be safely discarded. Bootstrapping nodes need only download the longest chain; they can skip work object verification entirely. Thus, the storage overhead of FairChain is bounded by the work objects of the last $R + k$ blocks.

⁴ The power distribution is not directly observable; PRS estimates it from workshares, as analyzed in Section 5.

⁵ <https://learnmeabitcoin.com/technical/mining/coinbase-transaction/#coinbase-maturity>

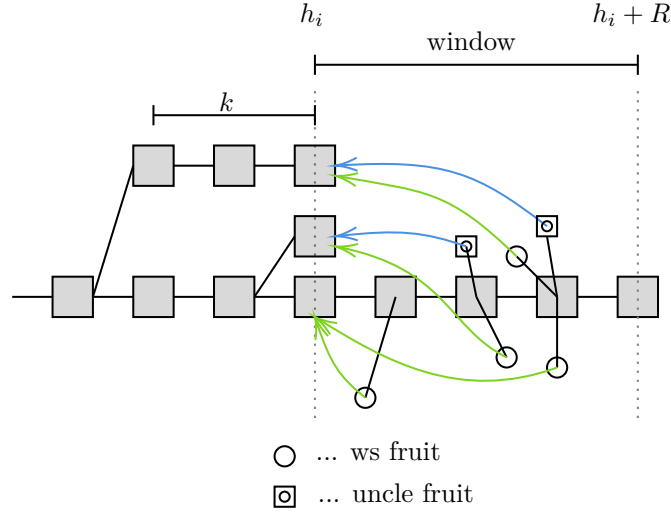


Figure 1 Uncles can be referenced up to R blocks away. Objects pointing to blocks more than k blocks into a fork are ineligible. At height $h_i + R$, the reward at height h_i is distributed proportionally to the intrinsic work of the eligible objects at h_i .

This storage improvement comes with a small tradeoff: a bootstrapping node cannot independently recompute past reward allocations. However, under the honest-majority assumption, honest miners validate reward allocations before extending a block, so a bootstrapping node can still infer their historical validity by following the heaviest PoW chain.

4 Theoretical Analysis

Our analysis closely follows that of FruitChains [16]. The protocol difference mainly affects the analysis of workshare freshness. In the rest of this section, we prove workobject freshness, consistency, growth, and fairness, ultimately leading to incentive compatibility. We denote the probability that any player mines a block in a round as p , and similarly the probability that a player mines a workshare in a round as p_f , such that $q = \frac{p_f}{p}$.

4.1 Main Theorem

Theorem 2 is the main theoretical result of our work, which shows that the workshare-based protocol guarantees the useful properties that we aim to achieve, namely consistency, chain growth, and fairness.

► **Theorem 2** (Security of FairChain). *For any constant $0 < \delta < 1$, and any p, p_f , let $R = 16$, $k_f = 2qRk$, and $T_0 = 5\frac{k_f}{\delta}$. Then, in $\Gamma_{\text{FairChain}}^{p, p_f, R}$ -environments, the Workshare protocol denoted $\Pi_{\text{FairChain}}(p, p_f, R)$ satisfies (i) k_f -consistency, (ii) chain growth rate (T_0, g_0, g_1) with $g_0 = (1 - \delta)(1 - \rho)np_f$ and $g_1 = (1 + \delta)np_f$, and (iii) fairness (T_0, δ) .*

We use the standard notions of consistency, chain growth, and fairness (e.g., [16]); their formal definitions, together with the definition of $\Gamma_{\text{FairChain}}^{p, p_f, R}$ and the proofs of Workobject freshness, consistency, and growth, are given in the paper's full version [1].

4.2 Workobject Fairness

Disregard the chain growth, chain quality, Workobject freshness, and Workobject growth failure events, which happen with negligible probability (cf. [1]). Consider some ϕ -fraction player subset selection S , some view view in the support of $EXEC^{\Pi_{\text{FairChain}}(p, p_f, R)}(A, Z, \kappa)$, some round r and some player i that is honest in round r of view . Let $C = b_0, \dots, b_{|C|}$ be the blocks in the view of i at r , let $f_0, \dots, f_\ell$ be the Workobjects contained in them, and let $m_0, \dots, m_\ell$ be the records contained in the Workobjects; let $f_j, \dots, f_{j+T}$ be the T consecutive Workobjects for some j , where $T \geq \frac{5k_f}{\delta}$.

Let r_0 be the round when the *block* in the view of i at r containing f_{j+k_f} was first added to *some* honest player j_0 's chain; let r_1 be the round when the block (again in the view of i at r) containing f_{j+T} was first added to some honest player j_1 's chain, and let $t = r_1 - r_0 - 2$ be the number of rounds from $r_0 + 1$ to $r_1 - 1$. We lower bound the number of S -compatible (honest) Workobjects in the sequence, similar to the proof of Workobject growth *lower bound*:

- By the *Workobject freshness* condition, every Workobject mined by some honest player between $(r_0 + 1)$ and $(r_1 - 1) - \text{wait}$ will be in the chain of j_1 at some position pos that is at least k positions from the end of the chain, before the beginning of round r_1 and will remain so.
- By the Chernoff bound, in the $t - \text{wait}$ rounds from $r_0 + 1$ to $(r_1 - 1) - \text{wait}$, except with probability $e^{-\Omega((t - \text{wait})\phi np_f)}$, the honest parties in S mine at least $(1 - \delta')(t - \text{wait})\phi np_f$ Workobjects (where δ' is some arbitrarily small constant), which thus are all included in the chain of j_1 by $r_1 - 1$.
- Since Workobjects are ordered by the block containing them, and since in round r_1 a *new* block is added which contains f_{j+T} , it follows from *blockchain consistency* that all these Workobjects contained in the sequence $f_1, \dots, f_{j+T}$ (recall that all these Workobjects are found in blocks that are at least k positions from the end of the chain, so by consistency, those blocks cannot change and thus were not added in round r_1 and consequently must come before the block containing f_{j+T}).
- Except with negligible probability (cf. [1]), these Workobjects are also all “new” (i.e., not included in the chains of j_0 at r_0) and different. Since in round r_0 , the block containing f_{j+k_f} was added to the chain of j_0 , and excluding blockchain collisions (cf. [1]), the chain of j_0 at r_0 up until (and including) the block which contains f_{j+k_f} is a prefix of C , all these Workobjects must in fact be contained in the sequence $f_{j+k_f}, \dots, f_{j+T}$.
- Finally, by Workobject consistency, at r_0 all honest players' Workobject chains contain $f_1, \dots, f_j$ (since recall that some player added f_{j+k_f} at r_0). Thus, all these Workobjects are S -compatible w.r.t. the prefix $f_1, \dots, f_{j-1}$ before the T segment we are considering.

We proceed to show that t is sufficiently large. Recall that j_0 is honest at r_0 and j_1 is honest at r_1 . We know that at r_1 , the Workobject chain contains at least f_{j+T} Workobjects. Additionally, at r_0 the Workobject f_{j+k_f} is added for the first time, so by Workobject consistency, at most $j + 2k_f$ Workobjects could have been in the chain of i at this point (since a Workobject at position $j + k_f$ is modified). Thus, the Workobject chain must have grown by at least $T - 2k_f$ from r_0 to r_1 . By the *upper bound on Workobject growth* we have that: $T - 2k_f \leq k_f + (1 - \delta')^2 np_f(t + 2)$. Thus: $t \geq \frac{1}{(1 + \delta')^2 np_f}(T - 3k_f) - 2$.

We conclude that (except with negligible probability) the number of Workobjects in the sequence is at least:

$$\begin{aligned}
& (1 - \delta')\phi np_f \left(\frac{1}{(1 + \delta')^2 np_f} (T - 3k_f) - 2 - \text{wait} \right) = \\
& (1 - \delta')\phi \left(\frac{1}{(1 + \delta')^2} (T - 3k_f) - np_f(\text{wait} + 2) \right) \geq (1 - \delta')\phi \left(\frac{1}{(1 + \delta')^2} (T - 3k_f) - k_f \right) \geq \\
& (1 - \delta')\phi \left(\frac{1}{(1 + \delta')^2} (T - 4.5k_f) \right) \geq \phi(T - 5k_f)
\end{aligned}$$

where we use $(\text{wait} + 2)np_f \leq k_f$, established in the full version of this paper [1], and take δ' sufficiently small. Since $T \geq \frac{5k_f}{\delta}$, we have that $T - 5k_f \geq (1 - \delta)T$, thus the number of Workobjects in the sequence is at least: $(1 - \delta)\phi T$.

4.3 Incentive Compatibility

Our final theoretical result is game-theoretic and concerns utility-maximizing participants.

Any secure blockchain protocol that satisfies δ -approximate fairness (where $\delta < 0.3$) w.r.t $T(\kappa)$ length windows can be used as the ledger underlying a cryptocurrency system while ensuring 3δ -incentive compatibility if players (i.e., miners) only care about how much money they receive.

We say that honest mining is a ρ -coalition-safe ϵ -Nash equilibrium if, with overwhelming probability, no $\rho' < \rho$ fraction coalition can gain more than a multiplicative factor $(1 + \epsilon)$ in utility, no matter what transactions are being processed; formally, consider some environment providing transactions into the system. We restrict to a setting where the total rewards and transaction fees during the run of the system are some fixed constant V .

Fairness implies that no matter what deviation the coalition performs, with overwhelming probability, the fraction of adversarial blocks in any $T(\kappa)$ -length window of the chain is upper bounded by $(1 + \delta)\rho$ and thus the total amount of compensation received by the attacker is bounded by $(1 + \delta)\rho \cdot V$; in contrast, by fairness, if the coalition had been following the honest protocol, they are guaranteed to receive at least $(1 - \delta)\rho \cdot V$; thus, the multiplicative increase in utility is: $\frac{1+\delta}{1-\delta} \leq 1 + 3\delta$ when $\delta < 0.3$.

5 Evaluation

We now compare the Proportional Reward Splitting (PRS) mechanism to other reward mechanisms under realistic parameters. In order to perform the comparison, we utilized the Markov Decision Process (MDP) implementation of [25]. MDPs are fitting tools in our case since we want to evaluate the optimal strategies w.r.t. a specific utility. Therefore, when parameterizing our MDPs with realistic values, e.g., in terms of block finality, we can identify the optimal strategy that utility-maximizing rational parties would opt for.

The model does not require knowledge of the identities of parties, or which parties are adversarial, but only assumes that the adversary A holds a certain amount of power. In practice, a protocol's designer could use our analysis to get a lower bound on their protocol's performance, by pinpointing the advantage of the strongest adversary that their protocol can tolerate.

Next, we first outline the construction of the MDP and then compare PRS with other reward mechanisms under three metrics, namely incentive compatibility, subversion gain, and censorship susceptibility. We then evaluate how different values of the PRS's parameters, namely the workshare and fork eligibility windows, affect its performance. Finally, we analyze the accuracy of estimating the adversarial and honest parties' power via workshares and outline the tradeoff between sampling accuracy and storage overhead.

5.1 Description of Markov Decision Process

The MDP implementation of PRS is a direct adaptation of the vanilla Reward Splitting (RS) MDP. Here we give a brief description of the MDP and refer to [25] for more details. Our implementation is available at [10].

We note that our evaluation is not a Monte Carlo simulation with independent runs. For each parameter point, we construct the corresponding finite MDP and solve for the optimal adversarial policy under the selected reward function. Therefore, the reported values correspond to expected long-run rewards induced by the optimal policy, rather than empirical means over repeated executions.

The state of the MDP is a tuple of four elements $\langle l_a, l_c, \text{fork}, \text{history} \rangle$:

- l_a denotes the length of the (private) adversarial chain;
- l_c denotes the length of the honest (public) chain;
- fork takes three values: *active* if there is a tie, i.e., honest miners are split between two chains; *cLast* if the latest block is honest; *aLast* if the latest block is adversarial.
- history is a bitstring as follows: (i) its length represents the number of consecutive attacker blocks in the common (main) chain; (ii) each position corresponds to a height of the main chain, starting from the last block (the tip of the chain); (iii) a 0 bit denotes that no competing honest block exists for the corresponding height, whereas 1 means that an honest block for that height exists.

State transitions occur via the following actions:

- *Adopt*: A adopts the public chain.
- *Wait*: A keeps mining privately and publishes no blocks.
- *Match*: A publishes a chain of private blocks equal to the length of the public chain, causing a tie; this is feasible when $l_a \geq l_c$ and $\text{fork} = \text{cLast}$.
- *Override_k*: A publishes a chain of private blocks which is k blocks longer than the public chain.

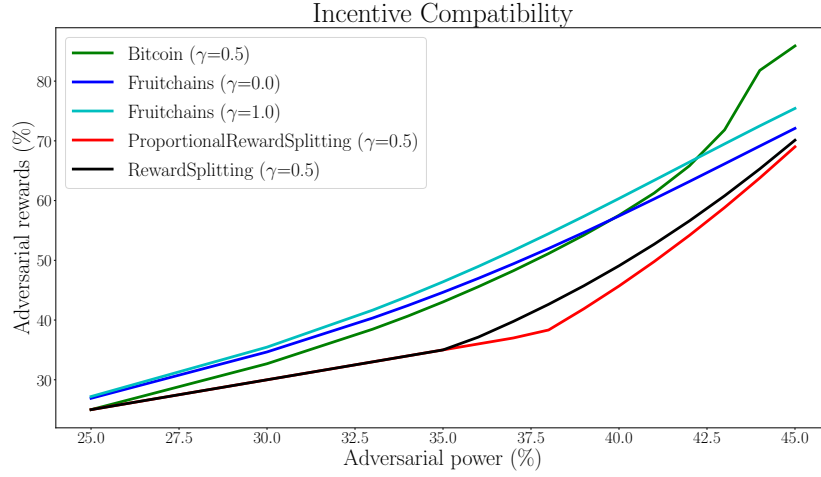
Finally, the reward allocation is the point of difference between PRS and RS. In RS, when choosing *Adopt*, the honest miners and A each receive half of the rewards for heights where both honest and adversarial blocks exist. Instead, in PRS the rewards are split proportionally to each side's mining power; for instance, if the honest miners (are set to) have 70% of mining power, they receive 70% of the rewards for these heights, instead of 50% as in RS.⁶ For the heights where only honest blocks exist, the honest miners get the full reward. Additionally, the adversary receives the full reward for heights where honest objects (blocks and/or workshares) are “pushed out” of the history, i.e., the adversary creates enough consecutive blocks to censor their inclusion.

5.2 Comparison of Reward Mechanisms

We first explore the metrics established in [25], namely: (i) incentive compatibility; (ii) subversion gain; (iii) censorship susceptibility. For each metric, we compare PRS with Bitcoin, FruitChains, and vanilla RS.

In all evaluations, we used the same parameters for the eligibility windows. The first window concerns object eligibility (i.e., the recency window), that is the number of block heights within which an object should be published in order to be eligible for rewards. This

⁶ In practice it is impossible to know the exact power distribution between honest parties and the adversary. Instead, an approximation is made using workshares. The accuracy of this approximation is explored in Section 5.4.



■ **Figure 2** Incentive compatibility for various reward mechanisms. The eligibility window for fruits (in FruitChains) and objects (in Proportional Reward Splitting) is set to $R = 6$. The fork eligibility window for FruitChains, Reward Splitting, and Proportional Reward Splitting is set to $k = 6$. Larger values indicate worse performance.

window was set to $R = 6$. The second window concerns fork eligibility, that is the depth of a fork up to which objects (that point to blocks of that depth) are eligible for rewards. This window was set to $k = 6$.

The parameter γ , which indicates the percentage of honest miners that adopt the adversarial forks, is set to 0.5 for Bitcoin, RS, and PRS, and 0 and 1 for FruitChains.⁷ In essence, when two chains of the same height are available, $\gamma = 0.5$ implies that honest miners choose between competing chains of equal height uniformly at random.

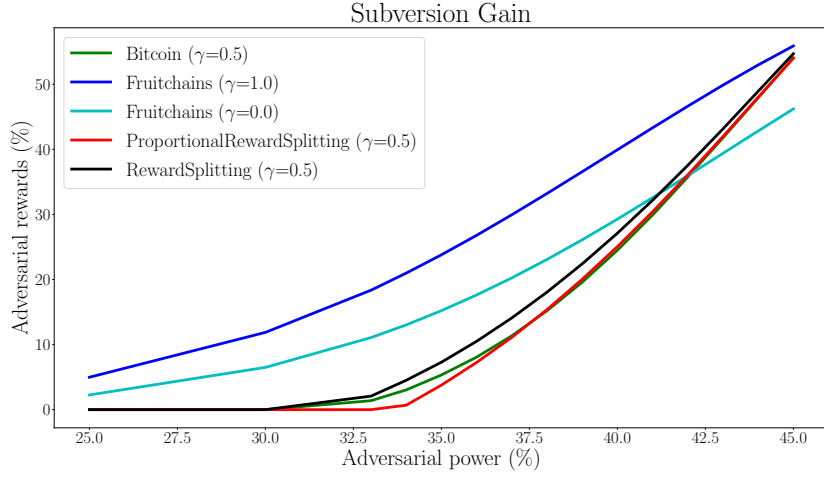
Following the baseline configuration of [25], we set the fruit-to-block ratio to 1. While higher fruit rates improve the incentive compatibility of FruitChains, at the cost of increased bandwidth, [25] observes only small improvements for higher ratios and finds RS to outperform FruitChains in its comparison.

Incentive Compatibility. The first metric of comparison is incentive compatibility (IC). Briefly, IC expresses the percentage of rewards that the adversary gains compared to its fair share, that is, its relative mining power. Figure 2 depicts the IC for the reward mechanisms under question. In essence, values above the line $x = y$ indicate that the adversary gains disproportionately more rewards, compared to its mining power, so larger values indicate worse IC performance.

PRS outperforms all other mechanisms on IC. For adversarial power up to 25%, all mechanisms except FruitChains are optimal. RS and PRS both remain optimal through 35%; beyond that, PRS strictly outperforms RS.

Subversion Gain. Subversion gain indicates the profitability of double-spending attacks. In particular, the profit from a double-spending attack is time-averaged and set to 3 times the block reward. In other words, if the adversary manages to orphan k blocks during the

⁷ For FruitChains we use both 0 and 1 because the FruitChains MDP implementation of [25] did not enable using 0.5 due to complexity issues.



■ **Figure 3** Comparison of subversion gain for various reward mechanisms. The eligibility window for fruits (in FruitChains) and objects (in Proportional Reward Splitting) is set to $R = 6$. The fork eligibility window for FruitChains, Reward Splitting, and Proportional Reward Splitting is set to $k = 6$. Larger values indicate worse performance.

attack, it is granted $3 \cdot k$ rewards. The confirmation time is set to 6 blocks, so the attack is successful if the adversary reverts 6 blocks in order to successfully double-spend the assets. If the attack is unsuccessful, the adversary gets no rewards, but also incurs no penalty.

Figure 3 depicts the comparison of the reward mechanisms w.r.t. subversion gain. For the most part, PRS is again the better option. In particular, up to 30% all mechanisms except FruitChains perform optimally, i.e., the adversary has zero gain. For the range of adversarial power between 30% and 38% PRS is the best option, whereas between 38% and 42% it is in effect equally good to Bitcoin. Above 42%, FruitChains with $\gamma = 0$ performs better, which is a result of the particularly favorable setting (compared to $\gamma = 0.5$ for the other mechanisms); instead, under $\gamma = 1$, FruitChains is strictly worse than all others for all adversarial powers.

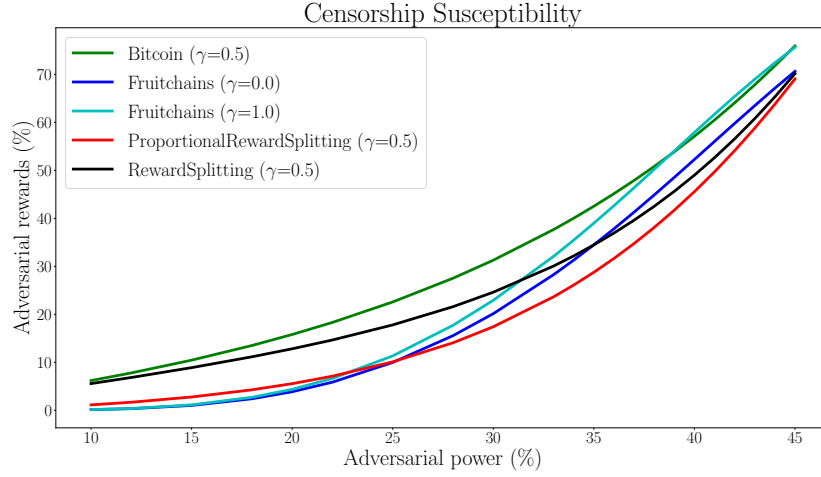
In summary, PRS is optimal w.r.t. subversion gain and, in some cases, it is strictly better than all other options.

Censorship Susceptibility. Censorship susceptibility indicates the maximum fraction of income loss that the adversary can incur on honest miners under the threat of a censorship attack. Specifically, the adversary’s reward corresponds to the honest miners’ loss in the form of orphaned blocks.

Figure 4 depicts the results for censorship susceptibility. These are consistent with [25], wherein FruitChains outperforms the other mechanisms for smaller adversaries but becomes relatively worse for larger values. Interestingly, PRS strictly outperforms vanilla RS for all adversarial powers. As a result, FruitChains is the best option only for adversarial power up to 25%, after which point PRS is the best option. Notably, even for smaller values of adversarial power, the difference between PRS and FruitChains is less than 2%.

5.3 Comparison of Eligibility Parameters

We now evaluate how the different eligibility parameters affect the performance of the reward mechanisms. In particular, we will compare the two types of reward splitting w.r.t. the fork and the workshare eligibility windows. The metric of comparison is incentive compatibility, since all metrics are affected in the same manner when changing the parameters. As before, we set γ to be 0.5.



■ **Figure 4** Comparison of censorship susceptibility for various reward mechanisms. The eligibility window for fruits (in FruitChains) and objects (in Proportional Reward Splitting) is set to $R = 6$. The fork eligibility window for FruitChains, Reward Splitting, and Proportional Reward Splitting is set to $k = 6$. Larger values indicate worse performance.

Workshare Eligibility Window. The first parameter is the workshare eligibility window R . This is the distance between the block that a workshare references and the block within which it gets published. Intuitively, larger window values require the adversary to censor the chain for longer periods of time, in order to force an honest workshare to become ineligible, thus making the attack harder. However, they also incur a delay on the finalization of objects and, consequently, the allocation of rewards.

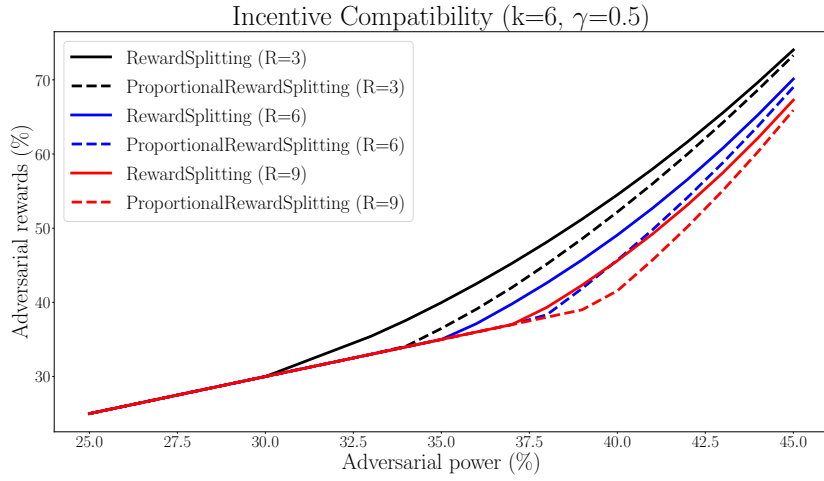
Figure 5 depicts how the two reward splitting mechanisms perform for three workshare eligibility window values, $R \in \{3, 6, 9\}$. Evidently, Proportional Reward Splitting outperforms vanilla Reward Splitting in all cases. As expected, larger values of R result in better performance, that is, lower adversarial rewards. Consequently, PRS is optimal for larger values of adversarial power; when $R = 3$, PRS is optimal for adversarial power up to 33%, whereas for $R = 9$ this goes up to 38%.

Fork Eligibility Window. The second parameter is the fork eligibility window k . This window sets the depth of a fork up to which objects are eligible for rewards. Specifically, if $k = 3$, then objects that point to a block in a fork are eligible for rewards only if this block is among the first 3 of the fork. Note that the height of the workshare, which is considered for the workshare eligibility window, is computed as the height of the referenced block. For example, if a workshare points to the second block of a fork which occurred at height 100, then the workshare's height is 102 and, in order to be eligible for rewards, it should be published in a (main chain) block up to height $102 + R$.

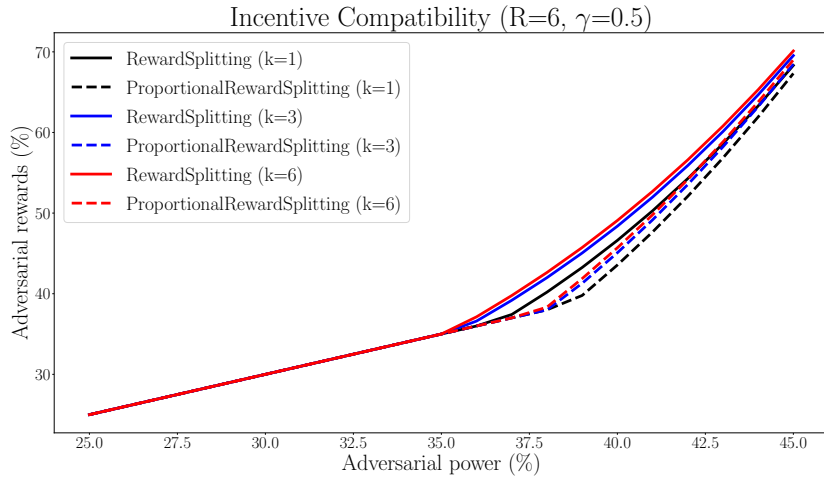
Figure 6 depicts the effect of the fork eligibility window on incentive compatibility. Evidently, lower values of k result in better performance, i.e., lower adversarial rewards. One explanation for this could be that lower values of k penalize forking more, so the adversary is incentivized to abandon forks more often, hence its rewards get closer to its fair share.

Interestingly, we observe that PRS outperforms RS even for different values of k . Specifically, PRS under $k = 6$ performs better (for the most part) than RS under $k = 1$.

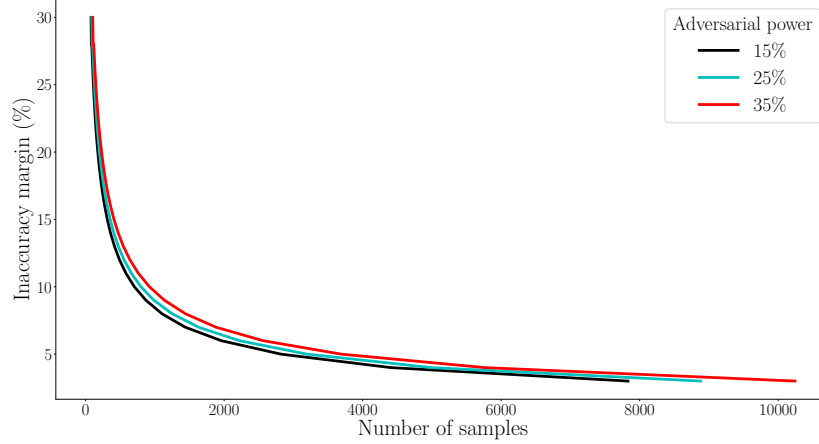
30:18 Optimal Reward Allocation via Proportional Splitting



■ **Figure 5** Incentive compatibility w.r.t. the workshare eligibility window between Reward Splitting and Proportional Reward Splitting. The fork window for objects is set to $k = 6$. Larger values indicate worse performance.



■ **Figure 6** Incentive compatibility for various values of the fork window between Reward Splitting and Proportional Reward Splitting. The eligibility window for objects is set to $R = 6$. Larger values indicate worse performance.



■ **Figure 7** Number of workshares needed to estimate the honest/adversarial power distribution, at error bound $\epsilon = 0.05$. Inaccuracy is the acceptable deviation from the ideal value.

5.4 Sampling Accuracy

So far, our evaluation of the proportional reward splitting mechanism assumes a perfect knowledge of the distribution of power between the honest parties and the adversary. In practice, this is not possible to achieve for proof-of-work (PoW) ledgers, as each party's proportional power is not visible to anyone, not even to the party itself. Although a party knows its *absolute* power, they do not know the exact amount of power of other parties.

Instead, the distribution of power can be inferred from observed events, specifically block creations. Since the probability that each party creates a block on each round is proportional to their power and each such event is independent, a large enough sample of blocks enables inferring the power distribution.

A pertinent question is how many samples are needed to make an accurate enough estimation, with a low margin of error. In our setting, the samples correspond to workshares and blocks. Therefore, the question of interest is how many workshares should be published per height, such that each block's rewards are distributed proportionally to each side's power, accurately and in most cases (low margin of error).

To estimate the number of workshares needed, we model the generation of workshares as a series of independent Bernoulli trials. Let $X_i \in \{0, 1\}$ be the i -th sample of random variable X , where 0 corresponds to an adversarial workshare creation and 1 corresponds to honest workshare. The probability of 1 for each X_i is p_h , so $\{X_i\}_{i \in [n]}$ is independent and identically distributed to a Bernoulli trial $\text{Bern}(p_h)$.

Let $X = \sum_{i=1}^n X_i$ be the random variable that represents the sum of the observed values, i.e., the number of honest workshares. It holds that $E[X] = n \cdot p_h$, where n is the number of samples (i.e., Bernoulli trials) and p_h is the probability of creating an honest workshare.

By the Chernoff bound: $\Pr[X \leq (1 - \delta) \cdot E[X]] < \epsilon = e^{-\frac{\delta^2 \cdot n \cdot p_h}{2}}$, where: (i) δ denotes the acceptable deviation from the completely accurate (expected) value; (ii) ϵ denotes the error bound of the observation, i.e., the probability that the observed value is less accurate than δ . By solving for n we find, for a given honest-power fraction p_h , the minimum number of workshares needed to get an accurate enough observation with low error probability:

$$n = -\frac{2 \cdot \log \epsilon}{\delta^2 \cdot p_h}.$$

Figure 7 depicts how n is affected given various values of the other parameters. Here, we set the error bound to 0.05; in other words, the probability that the observed distribution between the honest and adversarial parties is less accurate than the acceptable level of inaccuracy is 5%.

The x-axis corresponds to the number of workshares, while the y-axis corresponds to the acceptable level of inaccuracy. We observe that inaccuracy drops exponentially w.r.t. the number of samples used. Additionally, inaccuracy drops at a faster rate when the adversarial power is lower. We assume that each workshare is equivalent to a Bitcoin block’s header, which amounts to 80 bytes. Therefore, the overhead *per block* increases linearly with the number of workshares. As we explain in Section 3.5, however, this storage is off-chain and can be discarded after finalizing the rewards.

For example, assume adversarial power is 15%. 7,832 workshares achieve 3% inaccuracy (with error probability 5%), requiring 612 KB per block. Similarly, 1,000 workshares achieve approx. 8.5% inaccuracy at 79 KB. On the other hand, assuming 35% adversarial power, 3% inaccuracy is achieved with 10,242 workshares, which corresponds to 800 KB.

6 Discussion and Deployment

PRS in theory. PRS matches FruitChains in theoretical guarantees but outperforms it in practical settings. This is because the theoretical analysis evaluates the protocols *asymptotically*, for (very) large parameters. In practice, parameters like freshness or the fork eligibility window cannot be too large without delaying transaction and reward finalization.

At realistic parameter values, the theoretical guarantees no longer apply, so we turn to the MDP evaluation. One explanation for why PRS outperforms FruitChains lies in the tradeoff between “Rewarding the Bad” and “Punishing the Good” identified in [25]. FruitChains eliminates the incentive to fork by rewarding the adversary for failed attempts. PRS does the same through workshares, but to a lesser degree: it splits rewards proportionally rather than yielding the full contested reward to the adversary. This finer split strikes a better balance in practice.

PRS in practice. PRS outperforms Reward Splitting (RS) in practical terms. Both mechanisms follow the same punishment strategy, that is, splitting the rewards, but PRS does so in a more fine-grained way, which better approximates the distribution of power between the adversary and the honest parties. In other words, the gain of PRS comes from the difference between 50% and the actual value of adversarial power; hence, when the latter approaches the former, the two lines converge.

Dynamic difficulty. Our theoretical analysis follows the methodology of FruitChains [16], which operates in the static-difficulty setting; we therefore do not establish formal guarantees for variable-difficulty executions. We expect the same intuition to apply in this setting, similarly to variable-difficulty analyses of accumulated-work protocols such as NIPoPoWs: when the target changes, work should be interpreted relative to the target that was active when the object was created. In a deployment, the workshare target would be recalculated using the same difficulty-adjustment rule as the main block target, so that workshares continue to provide a fresh sample of the mining-power distribution. A formal analysis of PRS under variable difficulty is left for future work.

Deployment of FairChain. Our FairChain transformation has been deployed in practice as part of a Proof-of-Work blockchain launched in January 2025 [5] that uses PoEM as its fork-choice rule. The live blockchain records workshares on-chain with a hard cap of 9 workshares per block and a soft target of 3, where each workshare header is approximately 120 bytes, which is comparable to a standard transaction header. Workshares older than 2 blocks

(freshness parameter) are rejected by peers. Rewards are paid proportionally to each miner’s workshare-estimated power, with a linear staleness discount applied to shares approaching the freshness timeout, incentivizing timely propagation. At the current parameters, the storage overhead amounts to under 1 KB per block above baseline.

7 Related Work

The work most directly related to FairChain is FruitChains [16]. FruitChains was shown to be a ρ -coalition-safe ϵ -Nash equilibrium, i.e., any coalition of parties with a fraction of power less than ρ cannot gain more than ϵ of their fair share of rewards. As shown in Section 4, our mechanism achieves the same guarantee, as the analysis of FruitChains carries over in our case almost directly. Nonetheless, FairChain outperforms FruitChains in practice, that is, for parameter values smaller than the theoretical, unrealistic, bounds (cf. Section 5).

Recording low-work objects on-chain was explored in StrongChain [21]. StrongChain publishes workshares on-chain, as opposed to keeping them within a mining pool and invisible to the protocol. The paper offers a protocol implementation and a brief discussion of the protocol’s performance in the optimistic setting, where all parties are honest. However, StrongChain’s analysis is lacking in many aspects. Specifically, StrongChain offers only a high-level description of reward allocation and an intuition of why this allocation is beneficial. In comparison, our mechanism is analyzed both in theory, in a game-theoretic setting where parties are assumed rational instead of simply honest, and in practice via Markov decision processes. Moreover, StrongChain’s analysis considers only a specific instance of a selfish mining strategy, and analyzes the protocol’s performance under various parameter values, whereas our MDP analysis evaluates the performance of the optimal adversarial strategy.

Another line of work on recording low-work objects on-chain builds on the idea of “weak blocks”. These works include Subchains [19] and Flux [23]. Here, the main idea is the formation of “sub-chains”, that is, chains of objects with less PoW than blocks. Therefore, between two consecutive (heavy) blocks, a sub-chain of weak blocks may be created, enabling a more fine-grained estimation of the power distribution among participants. Flux offered a simulation-based analysis, showing that it performs better w.r.t. selfish mining compared to Bitcoin. However, as was shown in [25], Subchains performs worse in practice compared to Reward Splitting (RS). In comparison, our mechanism outperforms RS, and thus also Subchains, under the framework of [25].

The uncle reward system used in Ethereum [3] prior to its transition to Proof-of-Stake rewarded non-canonical work via *uncle blocks*. Ritz and Zugenmaier [18] show empirically that such a fixed partial reward is insufficient: it actually *lowers* the mining power threshold at which selfish mining becomes profitable, compared to Bitcoin, rather than eliminating the incentive. The core issue is that a fixed reward subsidizes the adversary regardless of their actual mining power. FairChain avoids this by splitting rewards proportionally to the power distribution, ensuring the adversary cannot benefit beyond their fair share.

Finally, FairChain can be instantiated on any longest-chain protocol that guarantees safety and liveness, including Nakamoto Consensus [14], uniform tie-breaking [6, 7], smallest-hash tie-breaking [13], unpredictable deterministic tie-breaking [11], Publish-or-Perish [24], and PoEM [12]. The security analysis requires only that the host protocol satisfies these two properties; no other assumptions on the fork-choice rule are needed.

References

- 1 Lukas Aumayr, Zeta Avarikioti, Dimitris Karakostas, Karl Kreder, and Shreevara Shastri. Optimal reward allocation via proportional splitting, 2025. doi:10.48550/arXiv.2503.10185.
- 2 Sarah Azouvi and Alexander Hicks. Sok: Tools for game theoretic models of security for cryptocurrencies. *CoRR*, abs/1905.08595, 2019. arXiv:1905.08595.
- 3 Vitalik Buterin. Ethereum: A next-generation smart contract and decentralised application platform. https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf, 2014.
- 4 Philippe Camacho and Sergio Demian Lerner. Decor+ lami: A scalable blockchain protocol. In *Scaling Bitcoin Workshops*, 2016.
- 5 Dominant Strategies. go-quai: Official Go implementation of the Quai protocol. <https://github.com/dominant-strategies/go-quai>, 2025. Accessed: 08-2026.
- 6 Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. Bitcoin-ng: A scalable blockchain protocol. In Katerina J. Argyraki and Rebecca Isaacs, editors, *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016*, pages 45–59. USENIX Association, 2016. URL: <https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/eyal>.
- 7 Ittay Eyal and Emin Gün Sirer. Majority is not enough: bitcoin mining is vulnerable. *Commun. ACM*, 61(7):95–102, 2018. doi:10.1145/3212998.
- 8 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015. doi:10.1007/978-3-662-46803-6_10.
- 9 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 Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 3–16. ACM, 2016. doi:10.1145/2976749.2978341.
- 10 Dimitris Karakostas. commonprefix/proportional-reward-splitting-MDP. Software, swhId: swh:1:dir:4a244dd0f68a10c4abb1fa0bf218527ba36dd085 (visited on 2026-08-19). URL: <https://github.com/commonprefix/proportional-reward-splitting-MDP>.
- 11 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 Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 279–296. USENIX Association, 2016. URL: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kogias>.
- 12 Karl Kreder, Shreevara Shastri, Apostolos Tzinas, Sriram Vishwanath, and Dionysis Zindros. A better proof-of-work fork choice rule. *Cryptology ePrint Archive*, Paper 2024/200, 2024. URL: <https://eprint.iacr.org/2024/200>.
- 13 Sergio Demian Lerner. Decor+ hop: A scalable blockchain protocol. *Semantic Scholar*, 2015.
- 14 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
- 15 Kartik Nayak, Srikanth Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 305–320. IEEE, 2016. doi:10.1109/EUROSP.2016.32.
- 16 Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 315–324. ACM, 2017. doi:10.1145/3087801.3087809.

- 17 retrodrive. Qubic performs 51% monero network takeover demonstration. <https://qubic.org/blog-detail/historic-takeover-complete-qubic-miners-now-secure-monero-network>, 2025. Accessed: 08-2026.
- 18 Fabian Ritz and Alf Zugenmaier. The impact of uncle rewards on selfish mining in ethereum. In *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 50–57, 2018. doi:10.1109/EuroSPW.2018.00013.
- 19 Peter R. Rizun. Subchains: A technique to scale bitcoin and improve the user experience. *Ledger*, 1:38–52, 2016. URL: <https://ledgerjournal.org/ojs/index.php/ledger/article/view/40>.
- 20 Ayelet Sapirshstein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. 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 515–532. Springer, 2016. doi:10.1007/978-3-662-54970-4_30.
- 21 Pawel Szalachowski, Daniël Reijsbergen, Ivan Homoliak, and Siwei Sun. Strongchain: Transparent and collaborative proof-of-work consensus. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 819–836. USENIX Association, 2019. URL: <https://www.usenix.org/conference/usenixsecurity19/presentation/szalachowski>.
- 22 Nicolas van Saberhagen. Cryptonote v 2.0. <https://github.com/monero-project/research-lab/blob/master/whitepaper/whitepaper.pdf>, 2013. Accessed: 08-2026.
- 23 Alexei Zamyatin, Nicholas Stifter, Philipp Schindler, Edgar R. Weippl, and William J. Knottenbelt. Flux: Revisiting near blocks for proof-of-work blockchains. *IACR Cryptol. ePrint Arch.*, page 415, 2018. URL: <https://eprint.iacr.org/2018/415>.
- 24 Ren Zhang and Bart Preneel. Publish or perish: A backward-compatible defense against selfish mining in bitcoin. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 277–292. Springer, 2017. doi:10.1007/978-3-319-52153-4_16.
- 25 Ren Zhang and Bart Preneel. Lay down the common metrics: Evaluating proof-of-work consensus protocols' security. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 175–192. IEEE, 2019. doi:10.1109/SP.2019.00086.