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.

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 $a \overset{\$}{\leftarrow} A$, 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 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 GOracle with the command 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 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 `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 `buffer`.
3. Run 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 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.

	$0, \dots, t_1$	$t_1, \dots, t_2$	$t_2 + \Delta, \dots, t_3$	**Performance/security in the interval** $t_2 + \Delta, \dots, t_3$
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 $t \in [0, \dots, m]$ 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 $\tau \in [0, \dots, m - \Delta]$, then the transaction $(\textsc{oracle}, \mathsf{query})$ 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 $t \in [0, \dots, m]$ and a transaction $(\textsc{oracle}, \mathsf{query})$ 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 $t \in [0, \dots, m]$.

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 $t \in [0, \dots, m]$, then for every j with $0 \le j \le m$, 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.

- $\mathsf{tx} = (\textsc{standard}, \mathsf{v}, \mathsf{addr}_i, \mathsf{addrs}, \mathsf{id}, \sigma)$. 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.
- $\mathsf{tx} = (\textsc{pledge}, \mathsf{addr}_i, \mathsf{addr}_j, \mathsf{id}_i, T, \sigma_i)$. 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.
- $(\textsc{reward}, C, m, \mathsf{cert})$, 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 $(\textsc{check-point}, H(\mathcal{G}^{\mathsf{PoR}}), 0, \perp)$, 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* $t \in [0, \dots, m]$ *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* $[0, \dots, m]$ *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.

- $\mathsf{tx} = (\textsc{check-point}, h, j, \mathsf{cert})$ 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).
- $\mathsf{tx} = (\textsc{complaint}, h^\star, \mathsf{cert}^\star, j)$ 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.
- $\mathsf{tx} = (\textsc{payback}, \mathsf{addr}_j, \mathsf{tx}, \sigma)$ 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 $\mathsf{tx}' = (\textsc{standard}, \mathsf{addr}_i, \cdot)$ 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 $\mathsf{tx} = (\textsc{pledge}, \mathsf{addr}_i, \mathsf{addr}_j, \mathsf{id}_i, T, \sigma_i)$ 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.*
- $\mathsf{tx} = (\textsc{liveness-violation})$. 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
- $\mathsf{tx} = (\textsc{block-request}, i)$. This transaction is accepted only if the ledger has already accepted a transaction $\mathsf{tx}' = (\textsc{liveness-violation})$, 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 $\textsc{block-request-respose}, i, B)$.

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 $(\textsc{check-point}, H(B), j, \mathsf{cert})$ (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 $\mathsf{tx} = (\textsc{complaint}, h^\star, \mathsf{cert}^\star, j)$, 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 $\mathsf{tx} = (\textsc{standard}, v, \mathsf{addr}_i, \{\mathsf{addr}_j\}, \mathsf{id}, \sigma)$, 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 $\mathsf{tx} = (\textsc{standard}, v, \mathsf{addr}_i, \{\mathsf{addr}_j\}, \mathsf{id}, \sigma)$. 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 $\mathsf{tx}^\star = (\textsc{pledge}, \mathsf{addr}_i, \mathsf{addr}_j, T)$, 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 $\mathsf{tx}' = (\textsc{payback}, \mathsf{addr}_j, \mathsf{tx}, \sigma)$ 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 $\boldsymbol{\Delta t}$.** 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 $\texttt{rew}_{\mathsf{PoR}}$ 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 $\texttt{rew}_{\mathsf{PoS}}$ 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 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*,* $(\mathsf{CheckTreasury}, \mathsf{chain}) < m \cdot \Delta t \cdot \mathtt{rew}_{\mathsf{PoR}}$ *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.