Bitcoin-Enhanced Proof-of-Stake Security: Possibilities and Impossibilities

Ertem Nusret Tas
Stanford University and BabylonChain
nusret@stanford.edu

David Tse
Stanford University and BabylonChain
dntse@stanford.edu

Fisher Yu
BabylonChain
fishermanymc@babylonchain.io

Sreeram Kannan
University of Washington, Seattle
ksreeram@uw.edu

Mohammad Ali Maddah-Ali
Stanford University
maddah.ali.ee@gmail.com

ABSTRACT

Bitcoin is the most secure blockchain in the world, supported by the immense hash power of its Proof-of-Work miners. Proof-of-Stake chains are energy-efficient, have fast finality and some accountability, but face several security issues: susceptibility to non-slashable long-range safety attacks, non-accountable transaction censorship and stalling attacks and difficulty to bootstrap PoS chains from low token valuation. We show these security issues are inherent in any PoS chain without an external trusted source, and propose a new protocol Babylon, where an off-the-shelf PoS protocol uses Bitcoin as an external source of trust to resolve these issues. An impossibility result justifies the optimality of Babylon. Our results shed light on the general question of how much security a PoS chain can derive from an external trusted chain by only making succinct commitments to the trusted chain.

1 INTRODUCTION

1.1 From Proof-of-work to proof-of-stake

Bitcoin, the most valuable and arguably the most secure blockchain in the world, is supported by a proof-of-work (PoW) protocol that requires its miners to solve hard math puzzles by computing many random hashes. Many newer blockchain projects eschew the proof-of-work paradigm in favor of proof-of-stake (PoS). A prominent example is Ethereum, which is currently migrating from PoW to PoS, a process 6 years in the making. Other prominent PoS blockchains include single chain ecosystems such as Cardano, Algorand, Solana, Avalanche as well as multi-chain ecosystems such as Polkadot and Cosmos. The Cosmos ecosystem, for example, consists of many application-specific zones all built on top of the Tendermint consensus protocol [10, 11]. Other ecosystems such as the Binance Smart Chain are also evolving into multi-chain ecosystems.

PoS protocols replace computational work with financial stake as the means to participate in the protocol. Thus, to execute the protocol as *validators*, nodes acquire coins of the PoS protocol, and *bond* their stake as collateral in a contract. This enables the PoS protocol to hold protocol violators accountable, and slash, *i.e.*, burn their bonded stake as punishment.

1.2 Proof-of-stake security issues

Security of PoS protocols has traditionally been shown under the honest majority (or super-majority) assumption, which states that the honest parties hold the majority of the stake [7, 11, 17, 18]. Introduced by Buterin and Griffith [13], the concept of *accountable safety enhances* the notion of security under honest majority with the ability to identify the validators who have provably violated the protocol in the event of a safety violation. Thus, accountable safety not only implies security under an honest majority but also the identification of protocol violators if a large quorum of the validators are adversarial and cause a safety violation. In lieu of making an unverifiable honest majority assumption, this approach aims to obtain a *cryptoeconomic* notion of security by holding protocol violators accountable and *slashing* their stake, thus enabling an exact quantification of the penalty for protocol violation. This *trust-minimizing* notion of security is central to the design of PoS protocols such as Gasper [14], the protocol supporting PoS Ethereum, and Tendermint [10, 11], the protocol supporting the Cosmos ecosystem. However, there are several fundamental limitations to the security of PoS protocols:

(1) **Safety attacks are not slashable:** While a PoS protocol with accountable safety can identify attackers, slashing of their stake is not always possible, implying a lack of *slashable safety*. For example, a posterior corruption attack can be mounted using old coins after the stake is already withdrawn and therefore cannot be slashed [7, 12, 18, 19]. These attacks are infeasible in a PoW protocol like Bitcoin as the attacker needs to counter the total difficulty of the existing longest chain. In contrast, they become affordable in a PoS protocol since the old coins have little value and can be bought by the adversary at a small price. Such posterior corruption attacks is a long-known problem with PoS protocols, and several approaches have been proposed to deal with them under the honest majority assumption (Section 2). Theorem 1 in Section 4.1 says that no PoS protocol can provide slashable safety without *external* trust assumptions. A typical external trust assumption used in practice is *off-chain social consensus checkpointing*. As social consensus is a slow process, this type of checkpointing leads to a long stake lock-up period *e.g.*, 21 days for Cosmos zones [2], which reduces the liquidity of the system. Moreover, social consensus cannot be relied upon in smaller blockchains with an immature community.
(2) **Liveness attacks are not accountable or slashable:** Unlike safety attacks where adversary can be identified by its votes on conflicting blocks, attacks such as inactivity or transaction censorship are hard to hold accountable in a PoS protocol. For example, Tendermint and PoS Ethereum attempt to hold inactive validators accountable through a process called *inactivity leak* [9]. However, in Section 5.1, we show an attack, where the

Contact author: DT.

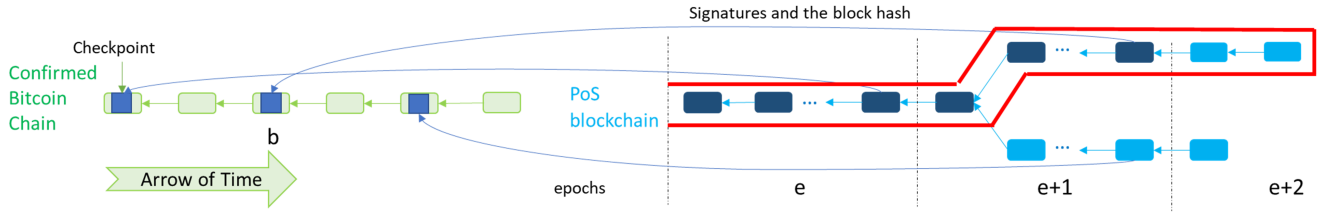

Figure 1: Babylon places hashes of the PoS blocks signed by the PoS validators. Ordering of these hashes enable clients to break ties between alternative PoS chains, and slash adversarial validators' stake before they withdraw in the event of a safety violation. The PoS chain in the view of a client c is shown by the red circle. Dark blue blocks represent the checkpointed chain of PoS blocks in c's view. The fast finalization rule outputs the PoS chain, while the slow finalization rule outputs the checkpointed chain, which is always a prefix of the PoS chain.

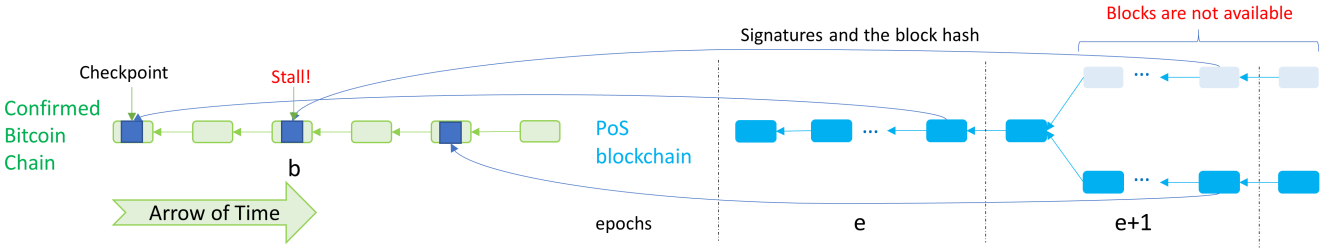

Figure 2: An adversary that controls a supermajority of active validators finalizes PoS blocks on an attack chain (top). It keeps the attack chain private, yet posts the hashes of the private blocks and the corresponding signatures on Bitcoin. Once these hashes and signatures are deep in Bitcoin, adversary helps build a conflicting chain (bottom) in public, and posts the hashes of its blocks and the corresponding signatures on Bitcoin. At this point, a client that sees the earlier checkpoint for the unavailable blocks, and the later one for the public blocks has two options: (1) It can stop outputting new blocks, or (2) it can ignore the earlier checkpoint and output the public blocks from the bottom chain. However, the adversary can later publish the unavailable blocks, and convince a late-coming client to output the blocks from the top, attack chain, causing a safety violation. Moreover, as the adversary might have withdrawn its stake by the time the blocks in the top, attack chain are published, it cannot be slashed. To avoid this attack, clients choose to stall upon seeing block b, *i.e.* emergency-break, if they see a signed checkpoint for unavailable blocks.

adversary creates a private chain without the honest validators' votes, thereby causing the honest validators to lose their stake. Generalizing the attack, Theorem 4 says that no PoS protocol can guarantee accountable liveness.

(3) **The bootstrapping problem:** Even if a PoS protocol could provide slashable security guarantees, the maximum financial loss an adversary can suffer due to slashing does not exceed the value of its staked coins. Thus, the cryptoeconomic security of a PoS protocol is proportional to its token valuation. Many PoS chains, particularly ones that support one specific application, *e.g.*, a Cosmos zone, start small with a low token valuation. This makes it difficult for new blockchains to support high-valued applications like decentralized finance or NFTs. Moreover, a PoS chain that experiences a significant drop in token valuation will suddenly be vulnerable to attacks.

1.3 Leveraging external trust

The main reason behind the security issues described in Section 1.2 is lack of a reliable *arrow of time*. For instance, posterior corruption attacks exploit the inability of the late-coming clients to distinguish between the canonical chain minted by the honest validators and the adversary's history-revision chain that is published much later [18, 19]. Hence, to guarantee a slashable notion of safety, PoS protocols need an external source of trust that can periodically and publicly timestamp the canonical chain. Social consensus can be viewed as one such source of external trust, but because it is achieved off-chain, the level of security is hard to quantify. In this paper, we explore a more quantifiable approach, which is to use an *existing secure blockchain* as a source of external trust. Given such a trusted blockchain, we ask: *What is the limit to the security enhancement the trusted chain can provide to a PoS chain and what is the optimal protocol that achieves this limit?*

A natural example of such a trusted blockchain is Bitcoin. The main result of the paper is the construction of Babylon, where an off-the-shelf PoS protocol posts succinct information to Bitcoin for security enhancement. Moreover, we show that Babylon achieves the optimal security among all protocols that do not post the entire PoS data to Bitcoin. Indeed, it is trivial to see that if the PoS protocol is allowed to post its entire data onto the trusted chain, the PoS protocol can inherit its full security. But in a chain with low throughput like Bitcoin, posting the entire data is clearly infeasible. Our result shows exactly what the loss of security is from this limitation.

The idea of using a trusted parent chain to provide security to a PoS chain has been used in several industry projects and academic works. Most of these works focus on mitigating specific attack vectors. For example, a recently proposed protocol, BMS [32], uses Ethereum to keep track of the dynamic validator set of a PoS chain to withstand posterior corruption attacks. (That work was later extended to a protocol using Bitcoin instead of Ethereum [5].) In our paper, we broaden the investigation to find out the best security guarantees a trusted public blockchain such as Bitcoin can provide to a PoS chain,

and construct an optimal protocol, Babylon, that achieves these guarantees. A detailed comparison of Babylon and other approaches is described in Table 1 and Section 2.

1.4 Babylon

A PoS protocol, such as Tendermint, is executed by *validators*, which lock up their coins to join the validator set. The design of Babylon specifies the kind of information validators post on Bitcoin and how this information is used by the clients, observers of the protocol, to resolve attacks on the PoS chain (*cf.* Figure 1). Highlights of Babylon are presented below:

Checkpointing. Honest validators act as Bitcoin clients, and sign the hash of the last PoS block of each epoch (*cf.* Figure 1). They subsequently post the hash and the corresponding signatures on Bitcoin as *checkpoints*. Ordering imposed on these checkpoints by Bitcoin enable the clients to resolve safety violations, and identify and slash adversarial validators engaged in long range posterior corruption attacks before they can withdraw their stake.

*Fast finalization Rule (*cf. *Figure 1).* To output a PoS chain, a client c first observes the confirmed prefix of the longest Bitcoin chain in its view. It then uses the sequence of checkpoints on Bitcoin to obtain a *checkpointed chain* of PoS blocks. While constructing the checkpointed chain, PoS blocks with earlier checkpoints take precedence over conflicting blocks with later checkpoints. Once c constructs the checkpointed chain, it obtains the full PoS chain by attaching the remaining PoS blocks that extend the checkpointed chain. It stalls upon observing a fork among the PoS blocks that extend the checkpointed chain.

Since Bitcoin enables each client to resolve earlier forks and obtain a unique checkpointed chain, safety can only be violated for later blocks in c's view. Hence, adversarial validators cannot violate the safety of older PoS blocks through a long range posterior corruption attack after withdrawing their stake. On the other hand, if a safety attack is observed for the recent PoS blocks, the clients can detect the adversarial validators and enforce the slashing of their stake as it would not have been withdrawn. Protocol thus ensures slashable safety.

Emergency Break. If the adversary controls a supermajority of the validators, it can sign hashes that do not correspond to blocks available in clients' views. In this case, clients stop adding new PoS blocks to their PoS chains if they observe a signed checkpoint on Bitcoin, yet the corresponding block is unavailable. This emergency break is necessary to protect against data unavailability attacks (*cf.* Figure 2).

Fallback to Bitcoin. If a transaction is observed to be censored, execution of the PoS protocol is halted, and the hashes of all future PoS blocks and the corresponding signatures on them are posted on Bitcoin, which is directly used to order these blocks. This is analogous to operating the PoS protocol as a *rollup*, where Bitcoin plays the role of the parent chain and the PoS validators act like sequencers. Thus, a PoS chain that uses Bitcoin directly to order the blocks is said to be in the *rollup mode*.

Once the protocol enters the rollup mode, validators batch the PoS transactions into bundles and sign the bundles whose data is available. Once a bundle gathers sufficiently many signatures, validators post its hash and the associated signatures to Bitcoin. The rollup mode thus allows the recovery of liveness, albeit at Bitcoin latency, which is larger then the fast finalization latency of the PoS protocol.

Bitcoin Safety & Slow finalization Rule. Clients can achieve Bitcoin safety for their PoS chains if they adopt a slow finalization rule. In this case, clients only output the checkpointed chain in their views. Thus, they wait until a PoS block or its descendants are checkpointed on Bitcoin before outputting the block as part of its PoS ledger. In this case, the PoS ledgers are always safe (assuming Bitcoin is secure), however, the finalization latency is now as large as Bitcoin latency.

1.5 Security guarantees

Table 1 summarizes the security guarantees achieved by Babylon, assuming that Bitcoin is safe and live. Babylon resolves the three PoS security issues presented in Section 1.2 in the following way:

(1) **Safety:** Under the fast finalization rule, Babylon achieves slashable safety via checkpointing and stalling whenever data is unavailable. Slashable safety is not possible without an external source of trust.
(2) **Liveness:** Without external trust, a PoS protocol which is $n/3$-accountably-safe has no liveness guarantee beyond $n/3$ adversarial validators [31, Appendix B]. Babylon improves this liveness resilience from $n/3$ to $n/2$ by using Bitcoin as a fallback. However, when $f \geq n/2$, liveness cannot be guaranteed, and even worse, liveness violations cannot be held accountable. It turns out that this is not a fault of Babylon, but is inherent in any protocol which does not post the entire PoS transaction data onto Bitcoin (Theorem 4). In this regime, the protocol is susceptible to data unavailability attacks.
(3) **Bootstrapping:** Using the slow finalization rule, Babylon is safe no matter how many adversarial validators there are, as long as Bitcoin is secure. Thus, Babylon achieves Bitcoin safety. This is achieved at the expense of Bitcoin confirmation latency (even for $f < n/3$), but is useful in a bootstrapping mode or for important transactions, where slashable safety is not sufficient.

1.6 Outline

The rest of the paper is organized as follows. In Section 2, we review related work. In Section 3, we present the model and definitions of various notions of security. In Section 4, we first show that slashable safety is not possible in any PoS chain without external trust. Then we present Babylon 1.0, a Bitcoin-checkpointed protocol that provides slashable safety. In Section 5, we show that it is impossible to provide *both* slashable safety and slashable liveness, even when there is a data-limited source of external trust. Moreover, we give a bound on the liveness resilience that can be achieved. We then improve Babylon 1.0 to the full Babylon protocol to provide the optimal liveness resilience. In Section 6, we describe a modified finalization rule that provides Bitcoin safety to the PoS chains.

2 RELATED WORKS

2.1 Posterior corruption attacks

Among all the PoS security issues discussed in Section 1.2, posterior corruption attacks is the most well-known, [7, 12, 18, 19]. In a posterior corruption attack also known as founders' attack, long range attack, history revision attack or costless simulation, adversary acquires the old keys of the validators after they withdraw their stake. It then re-writes the protocol history by building a conflicting chain using these old keys. The conflicting chain forks from the canonical one at a past block, at the time of which the old keys constituted a

	Safety	Liveness			Withdrawal
		$f < n/3$	$n/3 \leq f < n/2$	$f \geq n/2$	
KES [17]	$n/3$-safe	PoS Latency	No guarantee	No guarantee	?
BMS [32]	$n/3$-safe	PoS Latency	No guarantee	No guarantee	Ethereum latency
Babylon: fast finalization	$n/3$-slashable safe	PoS Latency	Bitcoin Latency	No guarantee	Bitcoin latency
Babylon: slow finalization	always safe	Bitcoin Latency	Bitcoin Latency	No guarantee	Bitcoin latency

Table 1: The security guarantees of Babylon compared to other solutions, assuming the security of Bitcoin in the case of Babylon and the security of Ethereum in the case of BMS. Here, f is the number of adversarial validators and n is the total number of validators. m-safe means the protocol is safe whenever $f < m$, m-slashable-safe means that whenever safety is violated, m validators can be slashed (which is a stronger property than m-safe). Stake withdrawals happen with Bitcoin latency on Babylon as long as liveness is satisfied, whereas it happens with Ethereum latency on BMS. In theory, Algorand [17] can grant withdrawal requests on the order of seconds as it uses key-evolving signatures (KES) to recycle keys after every signature, but since KES is highly incentive incompatible, Algorand still uses social consensus checkpointing.

majority of the validator set. On the conflicting chain, the adversary replaces the old validators with new ones under its control and continues the attack. Thus, it can cause clients observing the protocol at different times to output conflicting chains.

Without additional trust assumptions, it is impossible to construct a secure PoS protocol, even if the majority of the active validators stay honest over the protocol execution [18, Theorem 2]. Thus, several solutions have been proposed: 1) checkpointing via social consensus (*e.g.*, [6, 8, 12, 18]); 2) use of key-evolving signatures (*e.g.*, [7, 17, 24]); 3) use of verifiable delay functions, *i.e.*, VDFs (*e.g.*, [34]); 4) timestamping on an existing PoW chain like Ethereum [32] or Bitcoin [5].

2.1.1 Social consensus. Social consensus refers to a trusted committee of observers, potentially distinct from the PoS validators, which periodically checkpoint finalized PoS blocks. It thus attempts to prevent posterior corruption attacks by making the blocks on the private attack chain distinguishable from the checkpointed ones on the canonical chain. For instance, in PoS Ethereum, clients identify the canonical chain with the help of checkpoints received from their peers. Since no honest peer provides a checkpoint on a private chain, posterior corruption attacks cannot confuse new validators [4].

As the trusted peers can be different for different validators, it is often difficult to quantify the trust assumption placed on social consensus. For instance, a small set of peers shared by all validators would imply centralization of trust, making security prone to attacks by a few entities. Conversely, a large set would face the problem of reaching consensus on checkpoints in a timely manner, leading to long withdrawal delays. For instance, Cosmos has a delay of 21 days [2], whereas delays in Ethereum can be as large as 13 days [27, Table 1] (calculated for 130,000 attesters with average balance of 32 ETH to accurately model the targeted attester numbers of PoS Ethereum).

2.1.2 Key-evolving signatures. Use of key-evolving signatures requires validators to forget old keys so that a history revision attack cannot be mounted. Security has been shown for various PoS protocols [7, 17] using key-evolving signatures under the honest majority assumption. This assumption is necessary to ensure that the majority of the active validators willingly forget their old keys. However, this is not necessarily incentive-compatible as there might be a strong incentive for the validators to remember the old keys in case they later become useful. Thus, key-evolving signatures render the honest majority assumption itself questionable by asking honest validators for a favor which they may be tempted to ignore to maximize their future payoffs. This observation is formalized in Section 4.1, which shows that key-evolving signatures are not sufficient to provide slashable safety for PoS protocols.

2.1.3 VDFs. VDFs can enable the clients to distinguish the canonical chain that has existed for a long time from an attack chain that was created much later. Hence, VDFs provide an arrow of time for the clients, which protects the PoS protocol against posterior corruption attacks. However, like key-evolving signatures, VDFs are not sufficient to provide slashable safety for PoS protocols (*cf.* Section 4.1).

Another problem with VDFs is the possibility of finding faster functions [3], which can then be used to mount posterior corruption attacks.

2.1.4 Timestamping the validator set. Posterior corruption attacks can be thwarted by timestamping the validator sets of the PoS protocol on an external public blockchain such as Ethereum [32] and Bitcoin [5]. For instance, Blockchain/BFT Membership Service (BMS) [32] uses a smart contract on Ethereum as a *reconfiguration service* that records the changes in the validator set. When validators request to join or leave the *current* set, the current validators send transactions containing the new validator set to the contract. Upon receiving transactions with the same new validator set from sufficiently many validators from the current set, *e.g.*, from over 1/3 of the current validators, the contract replaces the current set with the new one.

The goal of BMS is to protect the PoS protocol against posterior corruption attacks, where the adversary corrupts old validators, and creates an attack chain. On the attack chain, the old validators are replaced by an alternate set of new validators that are under the adversary's control and distinct from those on the canonical chain. If the honest validators constitute over 2/3 of the current validator set on the canonical chain, BMS enables the late-coming clients to identify and reject the attack chain as the validator changes on the attack chain could not have been recorded by the contract before the changes on the canonical chain. Hence, the PoS protocol using BMS satisfies safety and liveness under a dynamic set of validators if the fraction of adversarial validators, active at any given time, is bounded by 1/3 (*cf.* Table 1, third row, safety column and the liveness

column for $f < n/3$). By preventing posterior corruption attacks, BMS also helps reduce the withdrawal delay of the PoS protocols from weeks [2] to the order of minutes.

BMS requires an honest supermajority assumption on the current, active set of validators to prevent posterior corruption attacks, and as such, does not provide slashable safety. If the adversary controls a supermajority of the current validator set, it can create a private, hidden attack chain simultaneous with the public, canonical one, and post the changes in the validator set of the private chain on the contract before that of the public chain. In this case, late-coming clients would believe that the canonical chain is an attack chain, viewing it as a product of posterior corruption, and adopt the adversary's private chain once it is made public. Hence, late-coming clients would believe the validators on the canonical chain to be protocol violators, which implies the absence of slashable safety (*cf.* Figure 2 for a similar type of attack). Moreover, BMS cannot ensure liveness if the fraction of adversarial active validators exceeds $n/3$ (*cf.* Table 1, third row, liveness column for $n/3 \leq f < n/2$).

Unlike Babylon that can provide Bitcoin safety for bootstrapping chains and important transactions, BMS cannot provide Ethereum safety to the constituent PoS protocols even by adopting a slow finalization rule. This is because the state of the BMS consists of the current set of PoS validators and does not include any information about the PoS transactions. Thus, even if the clients of the PoS protocol wait until the validators that have signed the finalized PoS blocks are verified by the BMS on Ethereum, if the adversary controls a supermajority of the current set, it can finalize conflicting PoS blocks, and cause a safety violation.

2.2 Hybrid PoW-PoS protocols

A PoS protocol timestamped by Bitcoin is an example of a *hybrid PoW-PoS protocol*, where consensus is maintained by both the PoS validators and Bitcoin miners. One of the first such protocols is the Casper FFG finality gadget used in conjunction with a longest chain PoW protocol [13]. The finality gadget is run by PoS validators as an overlay to checkpoint and finalize blocks in an underlay PoW chain, where blocks are proposed by the miners. The finality gadget architecture is also used in many other PoS blockchains, such as PoS Ethereum [14] and Polkadot [33]. Bitcoing timestamping can be viewed as a "reverse" finality gadget, where the miners run an overlay PoW chain to checkpoint the underlay PoS chains run by their validators. Our design that combines Bitcoin with PoS protocols also leverages off insights from a recent line of work on secure compositions of protocols [25, 26, 30]. Babylon uses insights from Thunderella [28], which combines a longest chain protocol with a responsive BFT protocol.

2.3 Timestamping

Timestamping data on Bitcoin has been used for purposes other than resolving the limitations of PoS protocols. Timestamping on a secure distributed ledger, *e.g.*, Bitcoin, was proposed as a method to protect Proof-of-Work (PoW) based ledgers against 51% attacks in [23]. However, [23] requires the Bitcoin network to contain 'observing' miners, which publish timestamps of blocks from the ledger to be secured only if the block data is available. This implies changing the Bitcoin protocol to incorporate data-availability checks, whereas in our work, we analyze the limitations of security that can be achieved by using Bitcoin as is.

Two projects that use Bitcoin to secure PoS and PoW child chains are Veriblock [29] and Komodo [1]. Both projects suggest checkpointing child chains on Bitcoin to help resolve forks. However, they lack proper security proofs, and do not analyze how attacks on PoS chains can be made slashable.

Another usecase of timestamping, analyzed by [22], is posting commitments of digital content to Bitcoin to ensure integrity of the data. In this context, [21] implements a web-based service to help content creators prove their possession of a certain information in the past by posting timestamps of the data on Bitcoin.

3 MODEL

3.1 Validators and clients

In the client-server setting of state machine replication (SMR), there are two sets of nodes: validators and clients. Validators receive transactions as input, and execute a SMR protocol. Their goal is to ensure that the clients obtain the same sequence of transactions, thus, the same end state. We assume that the transactions are batched into *blocks*, and the clients output a totally-ordered sequence, *i.e.*, chain, of blocks, denoted by $\mathcal{L}$. Hence, we will hereafter refer to the SMR protocols as *blockchain protocols*.

To output a chain at a given time, clients query the validators, which reply with a set of consensus messages. Purpose of these messages is to ensure that the clients obtain and output the same, or consistent chains. Upon collecting messages from a subset S of the validators, each client outputs a chain. Clients can query the validators at arbitrary times, and cannot be assumed to have observed the protocol execution in between queries. Honest validators are simultaneously clients of the blockchain protocol, and output a chain. However, the set of clients is not restricted to the honest validators, and can contain external nodes that observe the protocol infrequently.

The blockchain protocol has *external validity*: A transaction in a given chain is valid with respect to its prefix if it satisfies external validity conditions. A block or chain is valid if it only contains valid transaction. Clients output valid chains and ignore invalid blocks.

In a permissioned protocol, the set of validators that execute the protocol stay the same over the protocol execution. This is in contract to PoS protocols that allow changes in the validator set. To distinguish the validators that are currently executing the protocol at a given time from the old validators, we will refer to the current validators as *active* and the older validators as *passive*.

3.2 Blocks and chains

Each block consists of two parts: a block header and transaction data. Transactions are typically organized in vector commitments, *e.g.* Merkle trees. The total order across the blocks in a chain together with the ordering of the transactions in each block gives a total order across all transactions included in the chain. Block headers contain

- a pointer to its parent block, *e.g.*, hash of the parent block,
- a vector commitment to the transactions, *e.g.*, a Merkle root,
- protocol related messages.

For a block B, we say that $B \in \mathcal{L}$ if B is part of the chain $\mathcal{L}$. Similarly, $\mathsf{tx} \in \mathcal{L}$ states that the transaction tx is included in a block that is in $\mathcal{L}$. A block B is said to *extend* B', if B' can be reached from B by following the parent pointers. Conversely, the blocks B and B' are said to *conflict* with each other if B' cannot be reached from B and

vice versa. The notation $\mathcal{L}_1 \prec \mathcal{L}_2$ denotes that $\mathcal{L}_1$ is a strict prefix of $\mathcal{L}_2$, whereas $\mathcal{L}_1 \preceq \mathcal{L}_2$ denotes that $\mathcal{L}_1$ is either a prefix of $\mathcal{L}_2$, or is the same as $\mathcal{L}_2$. The chains $\mathcal{L}_1$ and $\mathcal{L}_2$ are said to conflict with each other if they contain conflicting blocks.

3.3 Environment and adversary

Transactions are input to the validators by the environment $\mathcal{Z}$. Adversary $\mathcal{A}$ is a probabilistic polynomial time algorithm. Validators *corrupted* by the adversary are called *adversarial*. These validators surrender their internal states to the adversary and can deviate from the protocol arbitrarily (Byzantine faults) under the adversary's control. The remaining validators are called *honest* and follow the blockchain protocol as specified. Time is slotted, and the validators are assumed to have synchronized clocks[1].

3.4 Networking

Validators can send messages to each other and the clients, which then send each received message to every other client, *i.e.*, broadcast the messages. Messages are delivered by the adversary, which can observe a message sent by an honest validator before it is received. Network is synchronous, *i.e.*, the adversary is required to deliver all messages sent by the honest validators to other honest validators or clients within Δ slots. Here, Δ is a known parameter.

If a client observes the hash or header of a block before slot $r - \Delta$, yet has not seen the whole block by slot r, then that block is deemed to be *unavailable* in the client's view at slot r. Otherwise, the block is said to be *available*. Clients only output available blocks as part of their chains $\mathcal{L}$.

3.5 Security

Let $\mathcal{L}_r^{\mathsf{c}}$ denote the chain outputted by a client c at slot r. Let T_{fin} be a polynomial function of λ, security parameter of the blockchain protocol. We say that the protocol is T_{fin}-secure if the following properties are satisfied:

- **Safety:** For any slots r, r' and clients c, c', either $\mathcal{L}_r^{\mathsf{c}}$ is a prefix of $\mathcal{L}_{r'}^{\mathsf{c}'}$ or vice versa. For any client c, $\mathcal{L}_r^{\mathsf{c}}$ is a prefix of $\mathcal{L}_{r'}^{\mathsf{c}}$ for all slots r and r' such that $r' \geq r$.
- **$\mathbf{T_{\text{fin}}}$-Liveness:** If $\mathcal{Z}$ inputs a transaction tx to an honest validator at some slot r, then, $\mathsf{tx} \in \mathcal{L}_{r'}^{\mathsf{c}}$ for any slot $r' \geq r + T_{\text{fin}}$ and for any client c.

We will alternatively refer to $\mathcal{L}_r^{\mathsf{c}}$ as the PoS chain when we talk about PoS protocols.

3.6 Accountable security

We adopt the model in [31] to formalize accountable safety. During the protocol execution, validators exchange messages, *e.g.*, blocks or votes, and each validator records its view of the protocol, *e.g.*, all of the protocol-specific messages it received, in an execution transcript. If a client observes a safety violation *e.g.*, conflicting chains, it invokes a forensic protocol. The forensic protocol takes transcripts of (some of) the validators as input, and *except with probability negligible in the security parameter*, outputs a proof that a subset of the validators with size at least f have irrefutably violated the protocol rules. This proof is sufficient evidence to convince any client, including those that observe the system at a later slot, that the validators identified by the forensic protocol are protocol violators. *With overwhelming probability*, forensic protocol does not identify any honest validator as a protocol violator.

To invoke the forensic protocol, the client sends at least two conflicting blocks within the output chains to the validators. If honest validators have information needed to create the proof, they send their transcripts to the client. The client then invokes the forensic protocol with these transcripts and constructs the proof, which is subsequently sent to all other clients.

Definition 1. *Accountable safety resilience of a protocol is the minimum number f of validators identified by the forensic protocol as protocol violators when safety is violated. Such a protocol provides f-*accountable-safety.

Accountable safety resilience of f implies that, with overwhelming probability, the forensic protocol identifies f or more adversarial validators and does not identify any honest validator, as a protocol violator, in the event of a safety violation.

We next extend the notion of accountability to liveness violations using the same formalism. If a client observes that T_{fin}-liveness is violated, *i.e.*, a transaction input to an honest validator at slot r by $\mathcal{Z}$ is not in the client's output chain by slot $t + T_{\text{fin}}$, it again invokes the forensic protocol with the transcripts received from the validators[2]. The forensic protocol then outputs a proof that irrefutably identifies a subset of the adversarial validators as protocol violators with overwhelming probability.

Definition 2. *T_{fin}-accountable liveness resilience of a protocol is the minimum number f of adversarial validators identified by the forensic protocol as protocol violators when T_{fin}-liveness is violated. Such a protocol provides f-T_{fin}-*accountable-liveness.

Accountable liveness resilience of f implies that, with overwhelming probability, the forensic protocol identifies f or more adversarial validators and does not identify any honest validator, as a protocol violator, in the event of a liveness violation.

If there exists an adversary $\mathcal{A}$ such that the forensic protocol cannot irrefutably identify any adversarial validator as a protocol violator with overwhelming probability in the event of a safety or liveness violation, or identifies an honest validator with a non-negligible probability, then the protocol is said to provide 0-accountable safety or liveness resilience.

3.7 Proof-of-Stake protocols

In a *Proof-of-Stake* (PoS) protocol, nodes *stake* coins to become validators and participate in the protocol. Staked coins are locked in a *contract* executed on the chain. In our model, one locked coin corresponds to a single active validator that is equipped with a unique cryptographic identity.

Protocol execution starts with an initial committee of n validators with n coins staked in the contract[3]. At all slots, there is a non-empty queue of nodes waiting to stake their coins. However, the contract allows at most n coins to be staked at any given slot, implying that there are exactly n active validators at any slot.

[1] Bounded clock offset can be captured as part of the network delay

[2] Honest validators can ensure that the clients detect a liveness violation by sharing the transactions input to them by the environment, thus informing the client when a transaction was input to an honest validator.

[3] For simplicity, we assume n is 1 modulo 3, and $n/3$ is an integer smaller than n divided by 3.

The PoS protocol proceeds in epochs measured by the clients in the number of blocks on their chains. For instance, if each epoch is scheduled to last for m blocks and a client observes a chain of $5m + 3$ PoS blocks (excluding the genesis block), then the first m blocks belong to the first epoch, the second m blocks to the second one, and so on, until the last 3 blocks, which are part of the on-going epoch 6.

During an epoch, the active validator set is fixed and the execution of the PoS protocol mimicks that of a *permissioned* blockchain protocol, where the validators are the n active validators of that epoch. However, the PoS protocol supports changes in the active validator set through withdrawals. An active validator can send a *withdrawal* transaction to the protocol to leave the active set and retrieve its staked coin. At the end of each epoch, clients inspect their chains and identify the validators whose withdrawal transactions have been included in the portion of the chain corresponding to the epoch. Then, starting with the next epoch, these validators are no longer in the active set, replaced with new ones from the staking queue. Let r denote the first slot such that an epoch boundary is reached and an active validator v has left the active set according to $\mathcal{L}_r^{\mathsf{c}}$ in some client c's view. Then, we say that v has become *passive* in c's view at slot r. From slot r on, v is no longer eligible to execute the blockchain protocol in c's view.

When a validator becomes passive according to some chain $\mathcal{L}$, its coin is *not* necessarily released by the contract immediately. Different PoS protocols can have different *withdrawal delays*. For example, Cosmos blockchains have a withdrawal delay of 21 days [2]. Withdrawal delay mechanism is central to security, and will be analyzed in subsequent sections.

If the withdrawing validator's coin is first released in the view of a client c at slot r, the validator is said to have *withdrawn its stake* in c's view at slot r. In a live PoS protocol, if an honest validator sends a withdrawal transaction at slot r, it should be able to withdraw its stake in the view of all clients by slot $r + T$ with high probability, where T is a finite number.

Before the protocol execution starts, adversary can corrupt a certain number of coins. When a validator with a corrupt coin becomes active, it is called an adversarial active validator. We assume that once a validator becomes passive, it immediately becomes adversarial if it has not been corrupted before. Let f denote the upper bound on the number of adversarial active validators over the execution of the protocol. A PoS protocol provides f_{s}-*safety* if it satisfies safety whenever $f \leq f_{\mathsf{s}}$. Similarly, a PoS protocol provides f_{l}-T_{fin}-*liveness* if it satisfies T_{fin}-liveness whenever $f \leq f_{\mathsf{l}}$.

To output a chain at some slot r, clients query the validators active in their view at slot r. Upon receiving responses from sufficiently many validators, clients output a chain. For instance, if the PoS protocol provides f_{l}-T_{fin}-liveness, the clients wait until they hear from $n - f_{\mathsf{l}}$ active validators since it is possible that the adversarial ones do not reply. Clients can query the validators at arbitrary times and might be offline in between queries. When we talk about events that happen on the PoS protocol, *e.g.* a safety violation, we refer to the clients that query the protocol after the event as *late-coming clients*.

3.8 Slashable security

A useful feature of the PoS protocols is the ability to impose financial punishments for protocol violators through the slashing, *i.e.*, burning of their locked coins. In this context, slashable security extends the notion of accountability to PoS protocols.

A validator v is said to be *slashable* in the view of a client c if,

(1) c has received or generated a proof through the forensic protocol at slot r such that v is irrefutably identified as a protocol violator,
(2) v has not withdrawn its stake in c's view by slot r.

If v is observed to be slashable by one client, no transaction that spends the coin staked by v will be viewed as valid by that client. In practice, once the contract that locks v's coin receives a proof for v's protocol violation, it will attempt to slash v's coin if its is still locked. However, once security is violated, the chain could stop executing new transactions, which would prevent the contract from slashing v's stake. Indeed, bootstrapping a new, secure chain after a security violation is a tricky problem, and often requires intervention from outside the protocol. Consequently, in our definition, we opted to use the word 'slashable' to indicate the conditional nature of *slashing* on the resumption of chain activity after the security violation.

Definition 3. *Slashable safety resilience of a PoS protocol is the minimum number f of validators that become slashable in the view of all clients when safety is violated. Such a protocol provides f-slashable-safety.*

Definition 4. *T_{fin}-slashable liveness resilience of a PoS protocol is the minimum number f of validators that become slashable in the view of all clients when T_{fin}-liveness is violated. Such a protocol provides f-T_{fin}-slashable-liveness.*

A slashable safety or liveness resilience of f implies that, with overwhelming probability, in the event of a safety or liveness violation, all clients identify f or more adversarial validators as protocol violators before the validators withdraw their staked coins, and no client identifies any honest validator as a protocol violator.

PoS protocol that provides f-slashable-safety or f-T_{fin}-slashable-liveness satisfies safety or T_{fin}-liveness if the number of adversarial active validators stay below f throughout the protocol execution. Since a slashable validator has to be irrefutably identified as a protocol violator, f-slashable safety or liveness implies f'-accountable safety or liveness for some $f' \geq f$.

3.9 Model for Bitcoin

We model Bitcoin using the formalism of [20] and treat it as a black-box blockchain protocol, which accepts transactions and outputs a totally ordered sequence of Bitcoin blocks containing these transactions. To obtain the Bitcoin chain *confirmed* with parameter k at slot r, a client c takes the longest chain of Bitcoin blocks in its view, removes the last k blocks, and adopts the k deep prefix as its Bitcoin chain at slot r. To differentiate the confirmed Bitcoin chain outputted by a client c at slot r from its PoS chain, we denote it by C_r^{c}. If a Bitcoin block b or transaction tx first appears in the confirmed Bitcoin chain, hereafter called the Bitcoin chain, of a client c at slot r, we say that tx or b has become *confirmed* in c's view at slot r. We say that Bitcoin satisfies security with parameter k if the clients' Bitcoin chains confirmed with parameter k satisfy safety and T_{fin}-liveness. Here, we require T_{fin} to satisfy the following proposition:

Proposition 1. *Suppose Bitcoin is secure with parameter k with overwhelming probability. Then, for any client* c*, if a transaction* tx

is sent to Bitcoin at slot r such that $|C^{\mathsf{c}}_{r-2\Delta}| = \ell$, $\mathsf{tx} \in C^{\mathsf{c}}_{r'}$ for any $r' \geq r + T_{\text{fin}}$, and $|C^{\mathsf{c}}_{r'}| \leq \ell + k$ with overwhelming probability.

If the adversarial fraction of the mining power is less than $1/2 - \epsilon$ for some $\epsilon > 0$, then there exists a parameter k polynomial in the security parameter λ such that Bitcoin is secure with parameter k and satisfies the above proposition [20]. Hence, in the rest of the paper, we assume that the value of k is polynomial in λ.

We assume that the PoS validators can send arbitrary data to Bitcoin by using the OP_RETURN opcode, which allows 80 bytes of arbitrary data to be recorded in an unspendable transaction. Due to the limitations on the amount of data allowed in each Bitcoin transaction, we will aim to reduce the Bitcoin footprint of the protocols in the subsequent sections.

3.10 Notation

Given a positive integer m, we denote the set $\{1, 2, \ldots, m\}$ by $[m]$. We denote PoS blocks by capital B and the Bitcoin blocks by lowecase b.

4 OPTIMAL SAFETY

In this section, we present and analyze a simplified version of Babylon, Babylon 1.0, which achieves optimal safety guarantees but suboptimal in liveness guarantees. Liveness guarantees will be optimized in the full Babylon in Section 5.

4.1 Safety is not slashable without external trust

Without additional trust assumptions, no PoS protocol can provide slashable safety. Suppose there is a long range attack and a late-coming client observes two conflicting chains. As the client could not have witnessed the attack in progress, it cannot distinguish the attack chain from the canonical one. Hence, it cannot irrefutably identify any validator that is active on either chain as a protocol violator. Although the client might see that the passive validators that have initiated the long range attack violated the protocol rules, *e.g.*, by voting for conflicting blocks, these validators are not slashable as they have already withdrawn their stake. Hence, no validator becomes slashable in the client's view. This fact is formalized by the following theorem:

Theorem 1. *Assuming common knowledge of the initial set of active validators, without additional trust assumptions, no PoS protocol provides both f_{s}-slashable-safety and f_{l}-T_{fin}-liveness for any $f_{\mathsf{s}}, f_{\mathsf{l}} > 0$ and $T_{\text{fin}} < \infty$.*

Proof is given in Appendix A.

Key-evolving Signatures and VDFs. Although key-evolving signatures and VDFs prevent long range attacks, as Theorem 1 indicates, they are not sufficient to provide slashable safety. To emphasize this point, we present the following attack. Suppose the adversary controls a supermajority of the active validators. In the case of key-evolving signatures, the adversarial active validators can record their old keys, and use them to stage a long range attack after withdrawing their stake. Adversary can thus cause a safety violation, yet, the adversarial validators cannot be slashed as their stake is withdrawn. Similarly, in the case of VDFs, adversarial active validators can construct a private attack chain while they work on the canonical one, and run multiple VDF instances simultaneously for both chains. After withdrawing their stake, they can publish the attack chain with the correct VDF proofs. Thus, the adversary can again cause a safety violation without any slashing of its stake. These examples indicate that slashable safety cannot be achieved by cryptographic primitives that can be used by the adversary within its private execution.

4.2 Babylon 1.0 protocol with fast finalization

Algorithm 1 The function used by the client c to find the canonical PoS chain. It takes the blocktree $\mathcal{T}$, the confirmed Bitcoin chain C, the sequence of checkpointed block hashes $\mathbf{h}$ in c's view as input, and outputs $\mathcal{L}^{\mathsf{c}}_r$. The function GetActiveVals takes a blocktree $\mathcal{T}$, a block B and an epoch number ep, and outputs the active validators for the given epoch as determined by the prefix of the given block. The function IsSigned checks if there are signatures on the given hash value h from over $n/3$ of the given set of active validators. The function GetBlocks returns the blocks within the given blocktree that correspond to the preimage of the given hash and its prefix. It returns $\perp$ if one of these blocks is unavailable. The function IsLast returns true iff the given PoS block is the last block of the given epoch. The function IsFinal returns true iff the given sequence of blocks is finalized by the active validators (that have not violated the protocol rules) of epochs up until the given epoch. The function GetChildren returns the children of the given block within the blocktree.

```
1:  function OutputPosChain(𝒯, h, C)
2:      h_1, ..., h_m ← h
3:      ckpt ← B_0
4:      ep ← 1
5:      active_val ← GetActiveVals(𝒯, B_0, ep)
6:      for i = 1 to m                          ▷ Obtaining the checkpointed chain
7:          if IsSigned(C, h_i, active_val)
8:              B_i ← GetBlocks(𝒯, h_i)
9:              if B_i ≠ ⊥ ∧ ckpt[−1] ⪯ B_i ∧ IsFinal(𝒯, B_i, ep)
10:                 ▷ Adding blocks for epoch ep to ckpt
11:                 ckpt ← B_i                  ▷ Chain ending at B_i
12:                 if IsLast(𝒯, B_i, ep)
13:                     ep ← ep + 1
14:                     active_val ← GetActiveVals(𝒯, B_i, ep)
15:                 end if
16:             else if B_i = ⊥ ∨ ¬IsFinal(𝒯, B_i, ep)
17:                 ▷ Send a checkpoint for the current PoS chain ℒ to Bitcoin.
18:                 return ckpt             ▷ Emergency Break: Data Unavailable
19:             end if
20:         end if
21:     end for
22:     ch ← GetChildren(𝒯, ckpt[−1])
23:     ℒ ← ckpt
24:     while |ch| = 1
25:         ℒ ← ℒ ‖ ch
26:         ch ← GetChildren(𝒯, ch)
27:     end while
28:     return ℒ
29: end function
```

To provide slashable safety to PoS protocols, Bitcoin can be used as the additional source of trust. Babylon 1.0 is a Bitcoin checkpointing protocol which can be applied on any PoS blockchain protocol with accountable safety to upgrade the accountability guarantee to slashable safety. Examples of such protocols include PBFT [15], Tendermint [11], HotStuff [35], and Streamlet [16]. For concreteness, we focus on Tendermint which provides $n/3$-accountable safety.

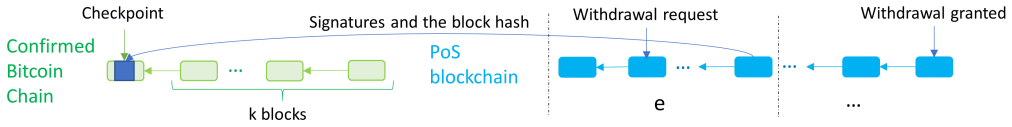

Figure 3: There is an epoch e block containing a withdrawal request and the hash of the last epoch e block and the signatures from the corresponding active validator set appear in a confirmed Bitcoin block in a client's view. The validator is granted permission to withdraw its stake once the Bitcoin block with the checkpoint becomes k deep in the confirmed Bitcoin chain.

Algorithm 2 The function used by the client c to verify stake withdrawal requests. It takes the requesting validator, the checkpointed chain and the confirmed Bitcoin chain in c's view, and returns true iff the withdrawal request for the specified validator is to be granted in the view of the client running the function. It sends a message to the on-chain contract if there is a fraud proof. The function ReturnReqBlock returns the block within the given checkpointed chain that includes the withdrawal request of the given validator. The function ReturnChkptBlock returns the Bitcoin block that contains the checkpoint transaction for the given PoS block or one of its descendants. The function IsClean returns true iff there are no fraud proofs or checkpoints for conflicting PoS blocks accusing the given validator in the Bitcoin chain.

```
1:  function GrantWithdrawal(v, ckpt, C)
2:      B ← ReturnReqBlock(v, ckpt)
3:      b ← ReturnChkptBlock(C, B)
4:      if b ∈ C[: −k] ∧ IsClean(v, C)
5:          return True              ▷ Withdrawal is granted in c's view.
6:      else if ¬IsClean(v, C)
7:          return False             ▷ c requests the on-chain contract to slash v.
8:      end if
9:      return False
10: end function
```

Let c denote a client, potentially a late-coming client or an honest validator, whose goal is to output the *canonical* PoS chain $\mathcal{L}$ that is consistent with the chains of all other clients. We assume that c downloads the PoS block headers and the corresponding transaction data upon observing the protocol. It also downloads the Bitcoin blocks and outputs a Bitcoin chain C, confirmed with parameter k.

In the description below, *finalized* blocks refer to the PoS blocks outputted by the PoS protocol. A finalization by the PoS protocol does not necessarily imply that the finalized block is included in the PoS chain outputted by any client. In fact, when the adversary controls a large fraction of the active validators, or engages in a posterior corruption attack, there might be conflicting finalized PoS blocks, and the clients could be forced to choose a subset of these finalized blocks. Algorithm 1 describes how clients use Bitcoin to output (potentially a subset of) these finalized blocks as part of their PoS chains (*cf.* Algorithm 1).

Checkpointing the PoS Chain. At the end of each epoch, the honest active validators sign the hash of the last finalized PoS block of the epoch. Then, an honest active validator v sends a Bitcoin transaction called the *checkpoint transaction*. The checkpoint transaction contains the hash and a quorum of signatures on the hash from over $n/3$ active validators of the epoch. These signatures are distinct from those that have finalized the block at the end of the epoch.

Suppose v observes multiple finalized and conflicting PoS blocks. As Tendermint provides $n/3$-accountable safety, it can generate a proof which irrefutably identifies $n/3$ adversarial PoS validators as protocol violators. In this case, v sends a transaction called the *fraud proof*, which contains this proof, to Bitcoin.

Fork-choice Rule. (Algorithm 1) To output the PoS chain $\mathcal{L}_r^{\mathsf{c}}$ at some slot r, c first downloads all the PoS block headers and transactions previously seen by all other clients. Using this data, it constructs a PoS blocktree.

Let h_j, $j \in [m]$, denote the sequence of block hashes, *i.e.* checkpoints, within the checkpoint transactions, listed from the genesis Bitcoin block to the tip of C_r^{c}, the confirmed Bitcoin chain in c's view at slot r. Define B_0 as the genesis PoS block. Then, starting at B_0, c constructs a *checkpointed* chain ckpt of PoS blocks by sequentially going through the checkpoint transactions. Let B_i denote the block at the preimage of h_i if the block is available in c's view. Suppose B_i from epoch e_i is the last PoS block appended to the checkpointed chain and c has gone through the sequence of checkpoints until h_j, $j \geq i$. Let $\tilde{e} = e_i + 1$ if B_i was the last block of its epoch; and $\tilde{e} = e_i$ otherwise.

(1) (*cf.* Algorithm 1, Line 9) If (i) the block B_{j+1} is from epoch $\tilde{e}$, (ii) B_{j+1} and every block in its prefix are available and finalized in c's view by the active validators of their respective epochs, (iii) B_{j+1} extends B_i, and (iv) h_{j+1} was signed by more than $n/3$ active validators of the epoch $\tilde{e}$, then, c sets B_{j+1} and its prefix as the checkpointed chain.

(2) **Emergency Break:** (Figure 2, *cf.* Algorithm 1, Line 16) If (i) B_{j+1} or a block in its prefix is either unavailable or not finalized by its respective validators in c's view, and (ii) h_{j+1} was signed by more than $n/3$ validators of epoch $\tilde{e}$, then, c stops going through the sequence h_j, $j \in [m]$, and outputs B_i and its prefix as the checkpointed chain[4]. This premature stalling of the fork-choice rule is necessary to prevent the data availability attack described by Figure 2.

(3) If both of the cases above fail, c skips h_{j+1} and moves to h_{j+2} and its pre-image block as the next candidate.

Unless case (2) happens, c sifts through all the checkpoints h_j, $j \in [m]$, and subsequently outputs the checkpointed chain $\mathsf{ckpt}_r^{\mathsf{c}}$. If case (2) happens, then c outputs B_i and its prefix as $\mathsf{ckpt}_r^{\mathsf{c}}$, and sends a checkpoint transaction for the block at the tip of its PoS chain $\mathcal{L}_r^{\mathsf{c}}$ along with the associated signatures. However, if c had previously outputted finalized blocks extending B_i as part of its PoS chain, it does not roll-back these blocks. Instead, it freezes and sets its old

[4]The client c always knows the active validator set for all epochs $e \leq \tilde{e}$. This is because, by definition, the last block B_i in its checkpointed chain and every block in B_i's prefix are available in c's view. If B_i is the last block of epoch e_i, c can infer the active validator set of epoch $e_i + 1$ from B_i and the blocks in its prefix.

PoS chain to be $\mathcal{L}^{\mathsf{c}}_r$, and sends a checkpoint transaction for the block at the tip of the frozen chain with a subset of signatures that have finalized the block.

Finally, suppose c outputs a checkpointed chain with the block B at its tip. Then, starting at B, c traverses a path to the leaves of the blocktree (*cf.* Algorithm 1, Line 23 onward). If there is a single chain from B to a leaf, c outputs the leaf and its prefix as the PoS chain $\mathcal{L}^{\mathsf{c}}_r$. Otherwise, c identifies the last PoS block B' (potentially the same as B) in the subtree of B, which has no conflicting siblings within the subtree, and outputs B' and its prefix as $\mathcal{L}^{\mathsf{c}}_r$.

Since c attaches the latest finalized PoS blocks to the tip of its output chain $\mathcal{L}^{\mathsf{c}}_r$, as long as there are no forks among finalized PoS blocks, the time for transactions to enter $\mathcal{L}^{\mathsf{c}}_r$ matches the latency of the PoS protocol, hence the name *fast finalization*.

Stake Withdrawals. (Figure 3, Algorithm 2) To withdraw its stake, a validator v first sends a PoS transaction called the *withdrawal transaction*. It is granted permission to withdraw its stake in the client c's view at slot r if

(1) The withdrawal transaction appears in a PoS block B in $\mathsf{ckpt}^{\mathsf{c}}_r$.
(2) Hash, *i.e.*, checkpoint of B or one of its descendants appears in a checkpoint transactions within a Bitcoin block that is at least k deep in C^{c}_r.
(3) The client has not observed any fraud proof in C^{c}_r that irrefutably identifies v as a protocol violator. Similarly, the client has not observed any checkpoint, *i.e.*, hash, in C^{c}_r for finalized and available PoS blocks that conflict with B. (Signatures by v on conflicting PoS blocks constitute a fraud proof.)

Once the above conditions are satisfied in the validator v's view, it sends a *withdrawal transaction* to the PoS chain. An honest active validator includes the withdrawal transaction by v in its PoS block proposal if the above conditions are satisfied in its view. Upon observing a PoS proposal containing a withdrawal transaction, the honest active validators wait for Δ slots before they decide to sign the block. At this point, they sign for the proposal if the above conditions are also satisfied in their views. By synchrony, if they are satisfied in an honest proposer's view at the time of proposal, then they are satisfied in the view of all honest active validators at the time of signing. Thus, the Δ delay for proposals carrying withdrawal transactions ensure that v's transaction is finalized by the PoS chain despite potential, short-lived split views among the honest active validators. Once the transaction is finalized, the on-chain contract releases v's locked coin.

Slashing and Slashable Validators. Suppose a validator v has provably violated the protocol rules. Then, the contract on the PoS chain can slash v's locked coins upon receiving a fraud proof incriminating v if the PoS chain is live and v has not received its coins back.

If the client c observes a fraud proof incriminating v in its confirmed Bitcoin chain C^{c}_r at slot r, c does not consider v's signatures on future checkpoints as valid. It also does not consider v's signatures as valid when verifying the finality of the PoS blocks 'checkpointed' in Bitcoin for the first time after the fraud proof. For instance, suppose c observes a checkpoint h_j for a PoS block B_j in its Bitcoin chain. While verifying whether B_j and the blocks in its prefix (that have not yet been checkpointed) are finalized, c considers signatures only by the active validators that have *not* been identified as protocol violators by a fraud proof appearing in the prefix of h_{j+1}.

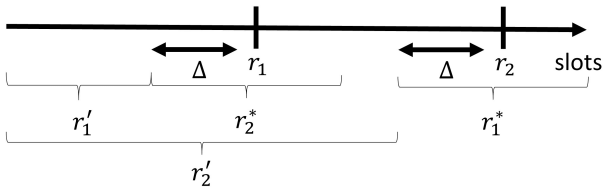

Figure 4: Parameters r_1, r_2, r_1', r_2', r_1^* and r_2^* defined for the proof of Theorem 2 and shown relative to each other.

4.3 Security analysis

Proposition 2. *Suppose Bitcoin is safe with parameter k with overwhelming probability. Then, the checkpointed chains held by the clients satisfy safety with overwhelming probability.*

Proof uses the fact that the safety of Bitcoin implies consensus on the sequence of checkpointed blocks, and is provided in Appendix C.

Theorem 2. *Suppose Bitcoin is secure with parameter k with overwhelming probability, and there is one honest active validator at all times. Then, the Babylon 1.0 protocol (Section 4.2) with fast finalization satisfies $n/3$-slashable safety with overwhelming probability.*

Proof. Suppose there are two clients $\mathsf{c}_1, \mathsf{c}_2$, and slots $r_1, r_2 \geq r_1$ such that $\mathcal{L}^{\mathsf{c}_1}_{r_1}$ conflicts with $\mathcal{L}^{\mathsf{c}_2}_{r_2}$. Let B_1 and B_2 denote the earliest conflicting blocks in $\mathcal{L}^{\mathsf{c}_1}_{r_1}$ and $\mathcal{L}^{\mathsf{c}_2}_{r_2}$ respectively. As B_1 and B_2 share a common parent, they also share the same active validator set.

Define $\mathsf{ckpt}_1 = \mathsf{ckpt}^{\mathsf{c}_1}_{r_1}$ and $\mathsf{ckpt}_2 = \mathsf{ckpt}^{\mathsf{c}_2}_{r_2}$. By Proposition 2, either $\mathsf{ckpt}_1 \preceq \mathsf{ckpt}_2$ or $\mathsf{ckpt}_2 \preceq \mathsf{ckpt}_1$. Since $\mathsf{ckpt}_1 \preceq \mathcal{L}^{\mathsf{c}_1}_{r_1}$ and $\mathsf{ckpt}_2 \preceq \mathcal{L}^{\mathsf{c}_2}_{r_2}$, $B_2 \notin \mathsf{ckpt}_1$ and $B_1 \notin \mathsf{ckpt}_2$. By Proposition 2, for all clients c and slots $r_1' \leq r_1 - \Delta$ and $r_2' \leq r_2 - \Delta$, it holds that $B_2 \notin \mathsf{ckpt}^{\mathsf{c}}_{r_1'}$ and $B_1 \notin \mathsf{ckpt}^{\mathsf{c}}_{r_2'}$ (*cf.* Figure 4).

Let $r_2^* > r_1 - \Delta$ denote the first slot B_2 appears in the checkpointed chain held by a client c_1^* (If there is no such slot, $r_2^* = \infty$.). Similarly, let $r_1^* > r_2 - \Delta$ denote the first slot B_1 appears in the checkpointed chain held by a client c_2^* (*cf.* Figure 4. If there is no such slot, $r_1^* = \infty$.). If $r_1^* < \infty$ or $r_2^* < \infty$, define b_1 and b_2 as the Bitcoin blocks containing the hashes of the first checkpoints that are either equal to B_1 and B_2 or extend B_1 and B_2 in the checkpointed chains observed by the clients c_2^* and c_1^* at slots r_1^* and r_2^* respectively. Then, for any client c, if $r_1^* < \infty$, $b_1 \in C^{\mathsf{c}}_{r_1^*+\Delta}$; if $r_2^* < \infty$, $b_2 \in C^{\mathsf{c}}_{r_2^*+\Delta}$. In this case, b_1 and b_2 are again the first confirmed Bitcoin blocks containing the hashes of the checkpoints that are either equal to B_1 and B_2 or extend B_1 and B_2 in the checkpointed chains observed by c respectively.

As the clients broadcast their consensus messages, by slot $r_0 = \min(\max(r_1, r_2), \max(r_1^*, r_2^*), \max(r_1, r_2^*), \max(r_1^*, r_2)) + \Delta$, both B_1 and B_2 are observed by all clients. Since B_1 and B_2 are finalized and conflicting blocks, an honest validator must have sent a fraud proof that incriminates at least $n/3$ of the validators in the common active validator set of B_1 and B_2, by slot r_0. Here, $r_0 \leq \max(r_1, r_2) + \Delta \leq r_2 + \Delta \leq r_1^* + 2\Delta$, and $r_0 \leq \max(r_1, r_2^*) + \Delta \leq r_2^* + 2\Delta$.

By Proposition 1, for any client c, the fraud proof is in $C^{\mathsf{c}}_{r_0'}$, where r_0' satisfies $|C^{\mathsf{c}}_{r_0'}| = |C^{\mathsf{c}}_{r_0-2\Delta}| + k$. As $r_0 \leq r_1^* + 2\Delta, r_2^* + 2\Delta$, if $r_1^* < \infty$ and the checkpoint in b_1 has always been available, the earliest slot b_1 can appear in c's Bitcoin chain is $r_0 - 2\Delta$. Similarly, in this case, if $r_2^* < \infty$ and the checkpoint in b_2 has always been available, the earliest slot b_2 can appear in c's Bitcoin chain is $r_0 - 2\Delta$. This implies that if b_1 (or b_2) appears in c's Bitcoin chain at all and contain available

checkpoints, the fraud proof will be included in either the k-th block extending b_1 (or b_2), or in its prefix. Since the active validators for the blocks B_1 and B_2 cannot withdraw their stake in c's view before b_1 or b_2 become k deep in c's Bitcoin chain, and as $n/3$ of these validators will be irrefutably identified as protocol violators by c by slot r'_0, at least $n/3$ validators become slashable in c's view.

Without loss of generality, we next consider the case the checkpoint in b_1 is not available in the view of a client c when c first observes b_1 in its Bitcoin chain. Since the slashability of $1/3$ of validators is implied by the proof above in the case $b_2 \preceq b_1$, we assume $b_1 \preceq b_2$ in the arguments below. Upon observing b_1, c stalls its PoS chain, and sends a checkpoint for the block at the tip of its PoS chain. Recall that b_1 and b_2 are the first confirmed Bitcoin blocks containing the hashes of the checkpoints that are either equal to B_1 and B_2 or extend B_1 and B_2 in the checkpointed chains observed by the clients respectively. Let $\tilde{r}_2 \leq r_2 < \infty$ be the first time B_2 appears in the pos chain of any client c. Since $b_1 \preceq b_2$ and B_1 and B_2 conflict, c could not have observed either the block b_1 or the checkpoint for B_1 as available before slot $\tilde{r}_2$. Suppose c first observes b_1 at slot $\tilde{r}_1 > \tilde{r}_2$. If the checkpoint in b_1 is unavailable in c's view, c sends a checkpoint transaction for B_2 or a block extending it by slot $\tilde{r}_1$, which appears within k Bitcoin blocks of b_1 in every client's confirmed Bitcoin chain. Hence, as soon as a client outputs B_1 as part of its PoS chain, *e.g.*, by some slot r'_0, and ensures its visibility by all clients, $n/3$ of the active validators for the blocks B_1 and B_2 are irrefutably identified as protocol violators by all clients by slot $\max(r'_0, \tilde{r}_1) + \Delta$, and at least $n/3$ validators become slashable in all views. On the other hand, if the checkpoint in b_1 is available in c's view at slot $\tilde{r}_1$, then, a fraud proof is generated and sent to Bitcoin by slot $r'_0 = \tilde{r}_1 + \Delta$. Again, $n/3$ of the active validators for the blocks B_1 and B_2 are irrefutably identified as protocol violators by all clients by slot r'_0, and at least $n/3$ validators become slashable in all views. □

Theorem 3. *Suppose Bitcoin is secure with parameter k with overwhelming probability, and the number of adversarial active validators is less than $n/3$ at all times. Then, the Babylon 1.0 protocol (Section 4.2) with fast finalization satisfies T_{fin}-liveness with overwhelming probability, where $T_{\text{fin}} = \Theta(\lambda)$.*

Proof. By Theorem 2, the Bitcoin checkpointing protocol satisfies $n/3$-slashable safety. Hence, if the number of active adversarial validators is less than $n/3$ at all slots, it satisfies safety. Suppose a transaction tx is first input to an honest validator at some slot r by $\mathcal{Z}$. Then, from slot r and on, each honest validator v will include tx in its proposal until v observes a PoS block containing tx become finalized. Let c' be the client that holds the longest PoS chain among all clients at slot r. As the number of active adversarial validators is less than $n/3$, clients never observe an unavailable or non-finalized PoS block become checkpointed, thus never stop outputting new PoS blocks as part of their checkpointed, and PoS chains (*cf.* clause (2) in the fork-choice rule of Section 4.2). Hence, by network synchrony and the safety of the PoS protocol, for every client c, $\mathcal{L}_r^{\mathsf{c}'} \preceq \mathcal{L}_{r+\Delta}^{\mathsf{c}} \preceq \mathcal{L}_{r+2\Delta}^{\mathsf{c}'}$. Then, for every client c, either $\mathcal{L}_r^{\mathsf{c}'} = \mathcal{L}_{r+\Delta}^{\mathsf{c}}$ or $\mathcal{L}_r^{\mathsf{c}'} \prec \mathcal{L}_{r+2\Delta}^{\mathsf{c}'}$.

In the former case, every client agrees on the validator set at slot $r + \Delta$. By [11, Lemma 7], there exists a finite T_{tm} that is polynomial in the security parameter λ such that if every client agrees on the validator set, a new block that extends $\mathcal{L}_r^{\mathsf{c}'}$ is finalized and becomes part of the PoS chain in the clients' views by slot $r + T_{\text{tm}}$ except with probability $\text{negl}(\lambda)$. In the latter case, a new block that extends the longest PoS chain, thus all PoS chains held by the clients at slot r, is finalized in the view of c' by slot $r + 2\Delta$, and becomes part of the PoS chains in all clients' views by slot $r + 3\Delta$ by synchrony.

Finally, when the number of adversarial validators is less than $n/3$, with probability at least $2/3$, each block finalized after slot r must have been proposed by an honest validator. Then, for any given integer $m > 1$, by slot $r + (m+1)\max(T_{\text{tm}}, 3\Delta)$, the transaction tx will appear in each client's PoS chain except with probability $m\,\text{negl}(\lambda) + (1/3)^m$. Setting $m = \Theta(\lambda)$, it holds that $m\,\text{negl}(\lambda) + (1/3)^m = \text{negl}(\lambda)$. Consequently, for $m = \Theta(\lambda)$, liveness is satisfied with parameter T_{fin} that is linear in λ, except with probability $\text{negl}(\lambda)$. □

5 OPTIMAL LIVENESS

Babylon 1.0 enables Tendermint to provide slashable safety by checkpointing. However, the protocol guarantees liveness only when a supermajority of the active validators is honest. Moreover, it does not provide any accountable liveness guarantees. In this section, we explore how Babylon 1.0 can be improved to achieve optimal liveness guarantees.

5.1 No accountable liveness

Our first result is that accountable liveness is not possible even with the help of Bitcoin, or for that matter any timestamping service, if the entire PoS data is not uploaded to the service. In particular, the adversary can execute unaccountable liveness attacks whenever it controls more than half the validators.

Timestamping Service. Timestamping service is a consensus protocol that accepts messages from the validators and provides a total order across these messages. All messages sent by the validators at any slot r are outputted (in some order determined by the service) and observed by all validators and clients at slot $r + 1$. If a client queries the service at slot r, it receives the sequence of messages outputted by the timestamping service until slot r. The service imposes limitations on the total size of the messages that can be sent during the protocol execution.

Theorem 4. *Consider a PoS or permissioned protocol with n validators that provides f_{s}-accountable-safety for some $f_{\text{s}} > 0$ and has access to a timestamping service. Suppose each validator is given an externally valid input of m bits by $\mathcal{Z}$ but the number of bits written to the timestamping service is less than $m\lfloor n/2 \rfloor - 1$ bits. Then under a Δ-synchronous network, the protocol:*

(1) cannot provide f_{a}-T_{fin}-accountable-liveness for any $f_{\text{a}} > 0$ and $T_{\text{fin}} < \infty$;
(2) cannot provide f_{l}-T_{fin}-liveness for any $f_{\text{l}} \geq n/2$ and $T_{\text{fin}} < \infty$.

To provide some intuition to this theorem, we analyze *inactivity leak* [9], used in Cosmos zones and also proposed for PoS Ethereum to slash adversarial validators to recover from liveness attacks. We show that when the adversarial fraction of validators is more than $1/2$, inactivity leak can result in a gradual slashing of the *honest* validators' stake in the view of *late-coming* clients with non-negligible probability. Consider an accountably safe consensus protocol with the setup on Figure 5. Half of the validators are adversarial, and build an attack chain that is initially kept private. They do not communicate with the honest validators or vote for the blocks on the

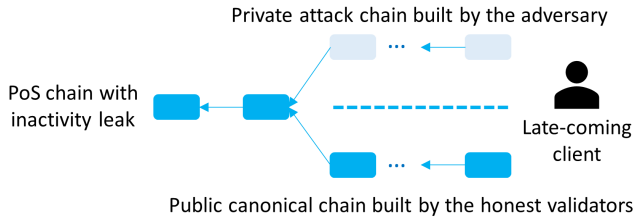

Figure 5: Inactivity leak attack. At the top is adversary's private attack chain. At the bottom is the public canonical chain built by the honest validators. Due to inactivity leak, honest and adversarial validators lose their stake on the attack and canonical chains respectively. A late-coming client cannot differentiate the canonical and attack chains.

public, canonical chain. As the honest validators are not privy to the adversary's actions, they also cannot vote for the blocks on the attack chain. Since only half of the validators are voting for the public blocks, liveness is temporarily violated for the public canonical chain. At this point, inactivity leak kicks in, and gradually slashes the stake of the adversarial validators on the public canonical chain to recover liveness. Similarly, the honest validators lose their stake on the private attack chain due to inactivity leak.

Finally, the adversary publishes the attack chain, which is subsequently seen by a late-coming client. As there are two conflicting chains in the client's view, there is a safety violation, and the client identifies at least one validator as a protocol violator as the protocol is accountably safe. Since the client could not have observed the attack in progress, it cannot distinguish the attack chain from the canonical one. Thus, with non-negligible probability, it identifies an honest validator on the canonical chain as protocol violator, which is a contradiction.

In the attack above, the data-limited timestamping service cannot help the late-coming client distinguish between the canonical and attack chains. This is because the honest validators cannot timestamp the entirety of public blocks early on as the timestamping service is data-limited. Thus, they cannot prove to a late-coming client that their canonical chain was public at the beginning. This enables the adversary to plausibly claim that the public, canonical chain was initially private, and its private attack chain was the public one.

The proof of Theorem 4 is given in Appendix B, and generalizes the attack on inactivity leak to any PoS, or permissioned protocol. As in the attack above, the proof exploits the indistinguishability, by a client, of two worlds with different honest and adversarial validators when the adversary can control over half of the validators, and the timestamping service is data-limited.

5.2 Optimal liveness resilience: full Babylon protocol with fast finalization

Theorem 4 states that no PoS protocol with accountable safety provides a positive accountable liveness resilience unless the transaction data within PoS blocks are posted on Bitcoin. As accountable liveness is impossible in this setting, we next focus on whether Bitcoin, despite its data limitation, can help increase the liveness resilience of Babylon 1.0

Babylon 1.0, presented in Section 4.2, provides $n/3$-slashable-safety and $n/3$-liveness. On the other hand, Theorem 4 only says that the liveness resilence of an accountably safe protocol cannot exceed $n/2$. We now show how to improve Babylon 1.0 to achieve

Algorithm 3 The function used by the client c to find the canonical PoS chain at a given time slot. It takes the blocktree, the confirmed Bitcoin chain, the sequence of checkpointed block hashes, bundle hashes and liveness transactions in c's view as input, and outputs $\mathcal{L}_r^c$. Here, t_i denotes the type of the transaction on Bitcoin, which can either be a checkpoint transaction, liveness transaction or a bundle. If t_i is a checkpoint transaction or bundle, h_i denotes the block or bundle hash, whereas if t_i is a liveness transaction, h_i denotes the censored transaction itself. The functions are defined in the caption of Algorithm 1 except for GetHeight, which returns the height of the Bitcoin block containing the given block or bundle.

function OutputPosChain($\mathcal{T}$, **h**, C)
  $(t_1, h_1), \ldots, (t_m, h_m) \leftarrow$ **h**
  ckpt, ep, active_val $\leftarrow$ B_0, 1, GetActiveVals($\mathcal{T}$, B_0, ep)
  bmode, censored, censoredtx, ht $\leftarrow$ False, False, $\perp$, -1
  for $i = 1$ to m
    if censored $\wedge$ GetHeight(h_i) $\geq$ ht + $2k$ ▷ *Enter BTC mode*
      bmode, censored, censoredtx $\leftarrow$ True, False, $\perp$
    else if bmode $\wedge$ GetHeight(h_i) $\geq$ ht + $2k + T_{\text{btc}}$ ▷ *Exit*
      bmode, ht $\leftarrow$ False, -1
    end if
    ▷ *Obtaining the checkpointed chain*
    if t_i = checkpoint $\wedge$ $\neg$bmode $\wedge$ IsSigned(C, h_i, active_val)
      $B_i \leftarrow$ GetBlocks($\mathcal{T}$, h_i)
      if $B_i \neq \perp \wedge$ ckpt$[-1] \preceq B_i \wedge$ IsFinal($\mathcal{T}$, B_i, ep)
        ckpt $\leftarrow$ $B_i.\mathcal{L}$ ▷ *Chain ending at* B_i
        if IsLast($\mathcal{T}$, B_i, ep)
          ep, active_val $\leftarrow$ ep + 1, GetActiveVals($\mathcal{T}$, B_i, ep)
        end if
        if censored $\wedge$ censoredtx $\subseteq$ ckpt
          censored, censoredtx, ht $\leftarrow$ False, $\perp$, -1
        end if
      else if $B_i = \perp \vee \neg$IsFinal($\mathcal{T}$, B_i, ep)
        return ckpt ▷ *Emergency Break: Data Unavailable*
      end if
    else if t_i = liveness $\wedge$ $\neg$bmode ▷ *Liveness block detected*
      tx, ht $\leftarrow$ h_i, GetHeight(h_i)
      if tx $\notin$ ckpt $\wedge$ $\neg$censored
        censored, censoredtx $\leftarrow$ True, {tx}
      else if tx $\notin$ ckpt $\wedge$ censored
        censoredtx $\leftarrow$ censoredtx $\cup$ {tx}
      end if
      ▷ *Bundle detected*
    else if t_i = bundle $\wedge$ bmode $\wedge$ IsSigned(C, h_i, active_val)
      $B_i \leftarrow$ GetBlocks($\mathcal{T}$, h_i)
      if $B_i \neq \perp$
        ckpt $\leftarrow$ ckpt $\|$ B_i
      else if $B_i = \perp$
        return ckpt ▷ *Emergency Break: Data Unavailable*
      end if
    end if
  end for
  $\mathcal{L}$, ch $\leftarrow$ ckpt, GetChildren($\mathcal{T}$, ckpt$[-1]$)
  while |ch| = 1
    if bmode $\vee$ ($\neg$censored $\wedge$ $|C| \geq$ ht + $k \wedge$ IsLast($\mathcal{T}$, ch, ep))
      Break
    end if
    $\mathcal{L}$, ch $\leftarrow$ $\mathcal{L}$ $\|$ ch, GetChildren($\mathcal{T}$, ch)
  end while
  return $\mathcal{L}$
end function

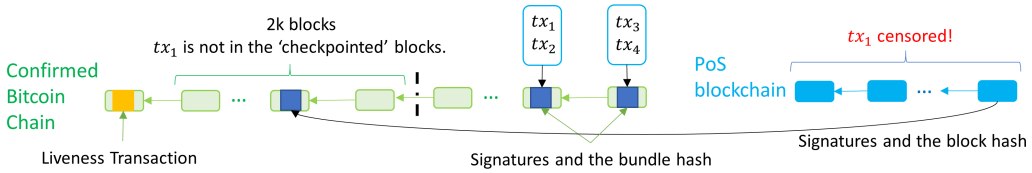

Figure 6: If tx_1 is observed to be censored by an honest validator v, it sends a liveness transaction to Bitcoin. Once liveness transaction becomes $2k$ deep in v's and the clients' views, they enter the rollup mode. In the rollup mode, validators group transactions into bundles and post signed hashes of these bundles on Bitcoin.

the optimal liveness resilience of $n/2$. Note that by [31, Appendix B], the liveness resilience f_l of a PoS protocol that provides $n/3$-accountable-safety cannot exceed $n/3$ *in the absence of external trust*, so the improvement of resilience from $n/3$ to $n/2$ depends crucially on the use of the data-limited timestamping service. Indeed, if the adversary controls $f \in [n/3, n/2)$ of the active validators and violates liveness, the improved protocol uses Bitcoin as a fallback mechanism to guarantee eventual liveness.

The full Babylon protocol proceeds in two modes: the normal mode and the rollup mode, where Bitcoin plays a more direct role in the ordering of the PoS blocks. Execution starts and continues in the normal mode as long as no PoS transaction is censored. If a transaction is observed to be censored, clients can force the execution to switch to the rollup mode. During the normal mode, checkpointing of the PoS chain, fork-choice rule, stake withdrawals and slashing work in the same way as described in Section 4.2, except for one difference: The minimum number of signatures required by a checkpoint transaction on the hash of the PoS blocks is set to be over $n/2$, instead of $n/3$.

We next focus on how censorship is detected and communicated by the clients, and the protocol execution afterwards. Algorithm 3 provides the full algorithm for the fork-choice rule.

Checkpointing. If a transaction tx input to an honest validator by $\mathcal{Z}$ at slot r has not appeared in $\mathcal{L}^{\mathsf{v}}_{r+T_{\mathsf{tm}}}$ in an honest validator v's view, v sends a *liveness transaction* to Bitcoin. The liveness transaction contains the censored tx, and signals a liveness violation in v's view. Here, T_{tm} represents the finalization latency of the Tendermint PoS protocol.

Suppose there is a block b within its Bitcoin chain that contains a liveness transaction for some tx. Upon observing b become k deep, v sends a checkpoint transaction for the block at the tip of its PoS chain, even if the block is not the last block of its epochs. If tx is not in its checkpointed chain, v also stops executing Tendermint. When b becomes $2k$ deep in its Bitcoin chain, if tx is still not in v's checkpointed chain, v enters the *rollup mode.*

Once in the rollup mode, v collects and orders transactions into *bundles* that are broadcast to all other validators. Upon observing a bundle of externally valid transactions, it signs the hash of the bundle. If it observes a bundle whose hash has been signed by over $n/2$ validators, it sends the hash of the bundle as well as the signatures to Bitcoin.

Fork-choice Rule (Figure 6, Algorithm 3). Consider a client c that observes the protocol at some slot $r' \geq r + T_{\mathsf{tm}}$. Suppose there is a block b within $C^{\mathsf{c}}_{r'}$ that contains a liveness transaction for some tx. Once b becomes k deep in c's Bitcoin chain, if tx is still not in c's checkpointed chain, c freezes its PoS chain. At this point, c also sends a checkpoint transaction for the block at the tip of its PoS chain, even if the block is not the last block of its epoch. Afterwards, c outputs new PoS blocks as part of its PoS chain, only if these new blocks are also part of its checkpointed chain (*cf.* Algorithm 3 Line 43. At this point in the algorithm, if c was previously awake and has already outputted blocks outside its checkpointed chain when the chain was frozen, it does not roll back its older blocks). If c observes tx within its checkpointed chain by the time b becomes $2k$ deep, it resumes outputting the new blocks that are not part of its checkpointed chain. Otherwise, once b becomes $2k$ deep in c's Bitcoin chain, if tx is not yet in c's checkpointed chain, c enters the *rollup mode* (*cf.* Figure 6, Algorithm 3, Line 6).

Once in the rollup mode, c first constructs the checkpointed chain by observing the prefix of its Bitcoin chain that ends at the $2k$-th block extending b. Suppose B_i from epoch e_i is the last PoS block appended to the checkpointed chain and it is followed by a sequence h_j, $j \in [m]$, of signed hash values corresponding to bundles in c's Bitcoin chain. Let $\tilde{e} = e_i + 1$ if B_i is the last block of epoch e_i, and $\tilde{e} = e_i$ otherwise. Then, c sifts through these values iteratively, and for each $j \in [m]$, acts as follows:

(1) (*cf.* Algorithm 3, Line 35) If h_j is signed by over $n/2$ active validators of epoch $\tilde{e}$, and its pre-image bundle is available in c's view, then c attaches the bundle to its PoS chain.
(2) (*cf.* Algorithm 3, Line 37) If h_j is signed by over $n/2$ active validators of epoch $\tilde{e}$, and its pre-image bundle is not available in c's view, then c stops going through the sequence h_j, $j \in [m]$, and outputs its current PoS chain.
(3) If neither of the conditions above are satisfied, c skips h_j and moves to h_{j+1} and its pre-image bundle as the next candidate.

Each client c leaves the rollup mode when it sees the T_{btc}-th Bitcoin block extending b (*cf.* Alg. 3 Line 8). Here, T_{btc} is a pre-determined parameter of the protocol that sets the duration of the rollup mode. After exiting the rollup mode, validators treat the hash of the last bundle as the parent hash for the new PoS blocks, and subsequently execute the protocol in the normal mode[5].

Stake withdrawals and slashing for safety attacks work as described by Algorithm 2 and Section 4.2.

5.3 Safety Analysis

Theorem 5. *Suppose Bitcoin is secure with parameter k with overwhelming probability and there is one honest active validator at all*

[5] Although not explicitly mentioned, it is possible to support stake withdrawals during the rollup mode once the bundles containing withdrawal transactions and signed by over half of the active validators of epoch $\tilde{e}$ become k deep in Bitcoin.

times. Then, the Babylon protocol (Section 5.2) with fast finalization satisfies $n/3$-slashable safety.

Theorem 5 follows from Theorem 2 and the safety of the checkpoints ordered by Bitcoin. It is presented in Appendix C.

5.4 Slashing and Liveness after Safety Violation

Theorem 5 states that as long as Bitcoin is secure, at least $n/3$ adversarial validators become slashable in the view of all clients when there is a safety violation. However, the theorem does not specify whether the validators that have become slashable can be slashed at all. Indeed, slashing can only be done if the PoS chain is live, a condition that might not be true after a safety violation.

When blocks on two or more conflicting chains are finalized by Tendermint, the chain with the earlier checkpoint in Bitcoin is chosen as the canonical one. However, it might not be possible for the honest active validators to unlock from the conflicting chains they have previously signed, and start extending the canonical chain; as that would require them to sign conflicting blocks. Thus, due to the absence of signatures from these stuck validators, the PoS chain might stall after a safety violation. In this case, even though the adversarial validators that have caused the safety violation become slashable, the on-chain contract might not be able to *slash* them since the chain itself is not live.

This issue of liveness recovery after a safety violation is present in many BFT protocols which strive to support accountability. For example, Cosmos chains enter into a panic state when safety is violated and need a manual reboot based on social consensus with the slashing done off chain. With Bitcoin, however, this problem can be solved to an extent. As in the case of stalling or censorship attacks, after which the protocol switches to the rollup mode, the honest validators can use Bitcoin to unstuck from their respective forks, bootstrap the PoS chain, and slash the adversarial validators if the honest validators constitute over *half* of the active PoS validators. For this purpose, they use the same process described in Section 5.2 for entering the rollup mode: Once the PoS chain loses liveness, an honest validator sends a liveness transaction for the censored PoS transactions, to Bitcoin. Soon afterwards, the honest PoS validators enter the rollup mode. This is because new checkpoints appearing on Bitcoin and signed by the slashable validators will not be considered as valid by the honest validators (*cf.* paragraph on slashing and slashable validators in Section 4.2), and cannot prevent the protocol from switching to the rollup mode. Once in the rollup mode, with their majority, the honest PoS validators can sign for new bundles and put the bundle hashes along with the signatures to Bitcoin. Through these new bundles, they can finalize the censored transactions, and slash the adversarial validators that have previously become slashable, using the on-chain contract.

After $n/3$ adversarial active validators are slashed, the $n/2$ honest active validators would constitute a supermajority of the remaining active validators. Hence, by treating the last bundle of the rollup mode as the new genesis PoS block, they can switch back to the normal mode and continue finalizing new PoS blocks through Tendermint (*cf.* Section 5.2, Line 8 of Alg. 3). This way, the PoS chain can bootstrap liveness and eventually return to the normal mode with fast finalization after safety violations.

5.5 Liveness Analysis

With the ability to bootstrap liveness after a safety violation, we can state the following theorem:

Theorem 6. *Suppose Bitcoin is secure with parameter k and the number of adversarial active validators is less than $n/2$ at all times. Then, the Babylon protocol (Section 5.2) with fast finalization satisfies T_{fin}-liveness with overwhelming probability, where T_{fin} is a polynomial in the security parameter λ.*

If the number of adversarial active validators f is less than $n/3$ at all times, liveness follows from Theorem 3. Otherwise, if $n/3 \leq f < n/2$, Bitcoin ensures the liveness of PoS transactions through the finalization of signed bundle hashes. Proof is presented in Appendix C.

6 BABYLON WITH SLOW FINALIZATION: BITCOIN SAFETY

So far in the paper, we have focused on the scenario where the clients of the PoS chain use the native *fast finalization rule*, where blocks are considered finalized immediately after voted upon by the validators of the PoS chains. Since Bitcoin confirmation operates at a slower time-scale, Bitcoin cannot protect the PoS chain against safety attacks under the fast finalization rule. What Bitcoin does is to make these attacks *slashable* by not allowing the attackers to withdraw funds after they double-signed, To achieve *Bitcoin safety* for some transactions, a client can choose to use a *slow finalization rule* where a PoS block is considered confirmed if in addition to being finalized on the PoS chain, its checkpoint is also confirmed in Bitcoin. More specifically, a client c using the slow finalization rule sets its PoS chain to be the same as its checkpointed chain at any time slot: $\mathcal{L}_r^{\mathsf{c}} = \mathsf{ckpt}_r^{\mathsf{c}}$. The major drawback of this scheme is that c now waits until the PoS blocks are checkpointed on Bitcoin, *i.e.*, until their hashes and the corresponding signatures are k deep in Bitcoin, before it can output them as part of its PoS chain.

Corollary 1. *Suppose Bitcoin is secure with parameter k with overwhelming probability, and there is an honest active validator at all times. Then, the Babylon protocol with slow finalization satisfies safety with overwhelming probability.*

Proof follows from Proposition 2. Corollary 1 holds for any number of adversarial active validators less than n.

Corollary 2. *Suppose Bitcoin is secure with parameter k with overwhelming probability and the number of active adversarial validators is less than $n/2$ at all times. Then, the Babylon protocol with slow finalization satisfies T_{fin}-liveness with overwhelming probability, where T_{fin} is a polynomial in the security parameter λ.*

Proof is given in Appendix C.

7 CONCLUSION

PoS protocols pioneered a stronger, accountable notion of safety that goes beyond the honest majority assumption, but were observed to be susceptible to non-slashable long-range attacks, non-accountable transaction censorship and stalling attacks and difficulties in bootstrapping PoS chains from low token valuation. To overcome these limitations, we have constructed and analyzed a protocol, Babylon, where an accountable PoS chain, *e.g.* a Tendermint chain, uses Bitcoin as a timestamping service, and operates in two nodes: In the *fast*

finality mode called the normal mode, clients of the PoS chain use the native fast finalization rule of the chain, and the role of Bitcoin is to provide slashable safety. In the second mode, clients use a slow finalization rule, and rely on Bitcoin as the consensus layer, thus giving this mode its name, the rollup mode. Clients can choose which mode they want to operate at. When there is censorship or stalling, the PoS chain can also switch to the rollup mode and rely on Bitcoin to regain liveness.

We have also proven Babylon's security and shown its optimality by characterizing the limitations of Bitcoin as a timestamping service for PoS protocols. Just as Babylon accommodates a general set of accountable PoS protocols, the advantages and limitations provided for these protocols by Bitcoin are not specific to Bitcoin, but can be offered by any trusted public blockchain. Hence, we can replace Bitcoin with any other trusted public blockchain that allows checkpointing PoS data, and all of the claimed improvements in the consensus security of PoS protocols would carry over with minor changes to the Babylon protocol.

ACKNOWLEDGEMENTS

We thank Joachim Neu, Lei Yang and Dionysis Zindros for several insightful discussions on this project.

REFERENCES

[1] Komodo. Advanced blockchain technology, focused on freedom. https://docs.komodoplatform.com/whitepaper/introduction.html#intoduction-to-komodo, 2018. Accessed: 2022-07-10.
[2] Launch communications — june community update. https://blog.cosmos.network/launch-communications-june-community-update-e1b29d66338, 2018. Accessed: 2022-04-17.
[3] VDF Alliance. VDF Alliance FPGA Competition. https://supranational.atlassian.net/wiki/spaces/VA/pages/36569208/FPGA+Competition, 2019.
[4] Aditya Asgaonkar. Weak Subjectivity in Eth2.0. https://notes.ethereum.org/\spacefactor\@m{}adiasg/weak-subjectvity-eth2#Distributing-Weak-Subjectivity-Checkpoint-States, 2019.
[5] Sarah Azouvi. Securing membership and state checkpoints of bft and pos blockchains by anchoring onto the bitcoin blockchain. https://www.youtube.com/watch?v=k4SacbLrypc, 2021. ConsensusDays 21.
[6] Sarah Azouvi, George Danezis, and Valeria Nikolaenko. Winkle: Foiling long-range attacks in proof-of-stake systems. In *AFT*, pages 189–201. ACM, 2020.
[7] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros Genesis: Composable proof-of-stake blockchains with dynamic availability. In *CCS*, pages 913–930. ACM, 2018.
[8] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better - how to make bitcoin a better currency. In *Financial Cryptography*, volume 7397 of *Lecture Notes in Computer Science*, pages 399–414. Springer, 2012.
[9] Carl Beekhuizen. Validated, staking on eth2: #1 - incentives. https://blog.ethereum.org/2020/01/13/validated-staking-on-eth2-1-incentives/, 2020. Accessed: 2021-11-3.
[10] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains, 2016.
[11] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *arXiv:1807.04938*, 2018.
[12] Vitalik Buterin. Proof of stake: How i learned to love weak subjectivity. https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/, 2014.
[13] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv:1710.09437*, 2019.
[14] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining GHOST and Casper. *arXiv:2003.03052*, 2020.
[15] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186. USENIX Association, 1999.
[16] Benjamin Y. Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains. In *AFT*, pages 1–11. ACM, 2020.
[17] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.
[18] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography*, volume 11598 of *Lecture Notes in Computer Science*, pages 23–41. Springer, 2019.
[19] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725, 2019.
[20] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.
[21] Bela Gipp, Norman Meuschke, and Andre Gernandt. Decentralized trusted timestamping using the crypto currency bitcoin. In *Proceedings of the iConference 2015*, 2015.
[22] Thomas Hepp, Patrick Wortner, Alexander Schönhals, and Bela Gipp. Securing physical assets on the blockchain: Linking a novel object identification concept with distributed ledgers. In *CRYBLOCK@MobiSys*, pages 60–65. ACM, 2018.
[23] Dimitris Karakostas and Aggelos Kiayias. Securing proof-of-work ledgers via checkpointing. In *IEEE ICBC*, pages 1–5. IEEE, 2021.
[24] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388. Springer, 2017.
[25] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *IEEE Symposium on Security and Privacy*, pages 446–465. IEEE, 2021.
[26] Joachim Neu, Ertem Nusret Tas, and David Tse. The availability-accountability dilemma and its resolution via accountability gadgets. In *Financial Cryptography and Data Security*, FC '22, 2022.
[27] Daejun Park and Aditya Asgaonkar. Analysis on weak subjectivity in ethereum 2.0, 2021.
[28] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2018.
[29] Maxwell Sanchez and Justin Fisher. Proof-of-proof: A decentralized, trustless, transparent, and scalable means of inheriting proof-of-work security. https://veriblock.org/wp-content/uploads/2018/03/PoP-White-Paper.pdf, 2018.
[30] Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. Blockchain CAP theorem allows user-dependent adaptivity and finality. In *Financial Cryptography (2)*, volume 12675 of *Lecture Notes in Computer Science*, pages 84–103. Springer, 2021.
[31] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. BFT protocol forensics. In *CCS*, pages 1722–1743. ACM, 2021.
[32] Selma Steinhoff, Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolic. BMS: Secure Decentralized Reconfiguration for Blockchain and BFT Systems. *arXiv:2109.03913*, 2021.
[33] Alistair Stewart and Eleftherios Kokoris-Kogia. GRANDPA: A Byzantine finality gadget. *arXiv:2007.01560*, 2020.
[34] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0.8.13. https://solana.com/solana-whitepaper.pdf, 2019.
[35] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356. ACM, 2019.

A PROOF OF THEOREM 1

Proof. Towards contradiction, suppose there exists a PoS protocol Π that provides f_l-T_{fin}-liveness and f_a-slashable-safety for some integers $f_l, f_a > 0$ and $T_{\text{fin}} < \infty$.

Let n be the number of active validators at any given slot. Let P, Q' and Q'' denote disjoint sets of distinct validators: $P = \{\mathsf{v}_i, i = 1, .., n\}$, $Q' = \{\mathsf{v}'_i, i = 1, .., n\}$ and $Q'' = \{\mathsf{v}''_i, i = 1, .., n\}$. Let $T < \infty$ denote the time it takes for a validator to withdraw its stake after its withdrawal transaction is finalized by the PoS protocol. We consider the following four worlds:

World 1: The initial set of active validators is P. Validators in P and Q' are honest. At slot 0, $\mathcal{Z}$ inputs transactions tx'_i, $i = 1, .., n$, to the validators in P. Here, tx'_i is the withdrawal transaction for v_i. Validators in P execute the PoS protocol, and record the consensus messages they observe in their transcripts. Suppose $\mathcal{Z}$ replaces each v_i with $\mathsf{v}'_i \in Q'$ as the new active validator.

At slot $T_{\text{fin}} + T$, $(\mathcal{A}, \mathcal{Z})$ spawns the client c_1. Upon querying the validators, c_1 receives messages from the validators in Q'. By T_{fin}-liveness, for all $i \in [n]$, $\mathsf{tx}'_i \in \mathcal{L}^{\mathsf{c}_1}_{T_{\text{fin}}+T}$. Moreover, by $T_{\text{fin}} + T$, all validators in P have withdrawn their stake in c_1's view, and the set of active validators is Q'.

World 2: The initial set of active validators is P. Validators in P and Q'' are honest. At slot 0, $\mathcal{Z}$ inputs transactions tx_i'', $i = 1, .., n$, to the validators in P. Here, tx_i'' is the withdrawal transaction for v_i. Validators in P execute the PoS protocol, and record the consensus messages they observe in their transcripts. Suppose $\mathcal{Z}$ replaces each v_i with $\mathsf{v}_i'' \in Q''$ as the new active validator.

At slot $T_{\text{fin}}+T$, $(\mathcal{A}, \mathcal{Z})$ spawns client c_2. Upon querying the validators, c_2 receives messages from the validators in Q''. By T_{fin}-liveness, for all $i \in [n]$, $\mathsf{tx}_i'' \in \mathcal{L}^{\mathsf{c}_2}_{T_{\text{fin}}+T}$. Moreover, by $T_{\text{fin}} + T$, all validators in P have withdrawn their stake in c_2's view, and the set of active validators is Q''.

World 3: The initial set of active validators is P. Validators in Q' are honest. Validators in P and Q'' are adversarial.

At slot 0, $\mathcal{Z}$ inputs transactions tx_i', $i = 1, .., n$, to the validators in P. Validators in P execute the PoS protocol, and record the consensus messages they observe in their transcripts.

Simultaneous with the execution above, $(\mathcal{A}, \mathcal{Z})$ creates a simulated execution in its head, where a different sequence of transactions, tx_i'', $i \in [n]$, are input to the validators in P at slot 0. In the simulated execution, $\mathcal{Z}$ replaces each v_i with $\mathsf{v}_i'' \in Q''$ as the new active validator. As in the real execution, validators in P execute the PoS protocol, and record the consensus messages they observe in their transcripts.

Finally, $(\mathcal{A}, \mathcal{Z})$ spawns two clients c_1 and c_2 at slot $T_{\text{fin}} + T$. Upon querying the validators, c_1 receives messages from the validators in Q' whereas c_2 receives messages from the validators in Q''. Since the worlds 1 and 3 are indistinguishable by c_1 except with negligible probability, for all $i \in [n]$, $\mathsf{tx}_i' \in \mathcal{L}^{\mathsf{c}_1}_{T_{\text{fin}}+T}$ with overwhelming probability. Since the worlds 2 and 3 are indistinguishable by c_2 except with negligible probability, for all $i \in [n]$, $\mathsf{tx}_i'' \in \mathcal{L}^{\mathsf{c}_2}_{T_{\text{fin}}+T}$ with overwhelming probability. Similarly, for all $i \in [n]$, $\mathsf{tx}_i' \notin \mathcal{L}^{\mathsf{c}_2}_{T_{\text{fin}}+T}$, and $\mathsf{tx}_i'' \notin \mathcal{L}^{\mathsf{c}_1}_{T_{\text{fin}}+T}$. Thus, $\mathcal{L}^{\mathsf{c}_1}_{T_{\text{fin}}+T}$ and $\mathcal{L}^{\mathsf{c}_2}_{T_{\text{fin}}+T}$ conflict with each other with overwhelming probability. Moreover, at slot $T_{\text{fin}} + T$, in the view of c_1 and c_2, the set of active validators are Q' and Q'' respectively, and all validators in P have withdrawn their stake.

As there is a safety violation and $f_{\text{a}} > 0$, at least one validator must have become slashable in the view of both clients. By definition of the forensic protocol, with overwhelming probability, a validator from the set Q'' becomes slashable in the clients' views as (i) the validators in P have withdrawn their stake in the clients' view and (ii) those in Q' are honest.

World 4: World 4 is the same as world 3, except that the validators in Q' are adversarial, those in Q'' are honest, and the real and simulated executions are run with the transactions tx_i'' and tx_i' respectively. As the worlds 3 and 4 are indistinguishable in the views of the clients except with negligible probability, they again identify a validator from Q'' as slashable in world 2 with non-negligible probability. However, the validators in Q'' are honest in world 2, which is a contradiction with the definition of the forensic protocol. □

B PROOF OF THEOREM 4

Proof. **Proof of part (1)** Towards contradiction, suppose there exists a PoS protocol Π that provides f_{s}-accountable-safety, and f_{a}-T_{fin}-accountable-liveness for some integers $f_{\text{a}}, f_{\text{s}} > 0$ and $T_{\text{fin}} < \infty$. Let f_{l} denote the T_{fin}-liveness resilience of Π.

We first analyze the case $f_{\text{l}} \geq n/2$. Let P and Q denote two sets that partition the validators into two groups of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ respectively. Consider the following worlds, where $\mathcal{Z}$ inputs externally valid bit strings, tx_i^P, $i \in [\lceil n/2 \rceil]$, and tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, to the validators in P and Q respectively at the beginning of the execution. Here, each validator i in P receives the unique string tx_i^P, and each validator j in Q receives the unique string tx_j^Q. Each string consists of m bits, where m is a polynomial in the security parameter λ.

World 1: There are two clients c_1 and c_2. Validators in P are honest, and those in Q are adversarial. In their heads, the adversarial validators simulate the execution of $\lfloor n/2 \rfloor$ honest validators that do not receive any messages from those in P over the network. They also do not send any messages to P and c_1, but reply to c_2.

Validators in Q send messages to the timestamping service I as dictated by the protocol Π. There could be messages on I sent by the validators in P that require a response from those in Q. In this case, the validators in Q reply as if they are honest validators and have not received any messages from those in P over the network.

As $|Q| = \lfloor n/2 \rfloor \leq f_{\text{l}}$, by the f_{l}-liveness of Π, clients c_1 and c_2 both output tx_i^P, $i \in [\lceil n/2 \rceil]$ as part of their chains by slot T_{fin}. Since there can be at most $m\lfloor n/2 \rfloor - 1$ bits of data on I, and tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, consists of $m\lfloor n/2 \rfloor$ bits, c_1 does not learn and cannot output all of tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, as part of its chain by slot T_{fin} with overwhelming probability.

World 2: There are again two clients c_1 and c_2. Validators in P are adversarial, and those in Q are honest. In their heads, the adversarial validators simulate the execution of the $\lceil n/2 \rceil$ honest validators from world 1, and pretend as if they do not receive any messages from those in Q over the network. They also do not send any messages to Q and c_1, but reply to the queries by c_2. They send the same messages to I as those sent by the honest validators within world 1.

As $|P| = \lceil n/2 \rceil \leq f_{\text{l}}$, by the f_{l}-liveness of Π, clients c_1 and c_2 both output tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, as part of their chains by slot T_{fin}. Since there can be at most $m\lfloor n/2 \rfloor - 1$ bits of data on I, and tx_i^P, $i \in [\lceil n/2 \rceil]$, consists of $m\lceil n/2 \rceil$ bits, c_1 does not learn and cannot output all of tx_i^P, $i \in [\lceil n/2 \rceil]$, as part of its chain by slot T_{fin} with overwhelming probability

As the worlds 1 and 2 are indistinguishable by c_2 except with negligible probability, it outputs the same chain containing tx_i^P, $i \in [\lceil n/2 \rceil]$, and tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, in both worlds with overwhelming probability. However, c_1's chain contains tx_i^P, $i \in [\lceil n/2 \rceil]$, but not tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, in world 1, and tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, but not tx_i^P, $i \in [\lceil n/2 \rceil]$, in world 2. This implies that there is a safety violation in either world 1 or world 2 or both worlds with non-negligible probability. Without loss of generality, suppose there is a safety violation in world 2. In this case, c_1 asks the validators for their transcripts, upon which the adversarial validators in P reply with transcripts that omit the messages received from the set Q. As $f_{\text{s}} > 0$, by invoking the forensic protocol with the transcripts received, c_1 identifies a non-empty subset $S \subseteq P$ of the adversarial validators, and outputs a proof that the validators in S have violated the protocol Π. However, in this case, an adversarial validator in world 1 can emulate the behavior of c_1 in world 2, and ask the validators for their transcripts. It can then invoke the forensic protocol with the transcripts, and output a proof that identifies the same subset $S \subseteq P$ of validators as protocol violators. Since the two worlds are indistinguishable by c_2 except

with negligible probability, upon receiving this proof, it identifies the honest validators in $S \subseteq P$ as protocol violators in world 1 as well with non-negligible probability, which is a contradiction. By the same reasoning, if the safety violation happened in world 1, an adversarial validator in world 2 can construct a proof accusing an honest validator in world 2 in c_2's view with non-negligible probability, again a contradiction.

We next analyze the case $f_{\mathsf{l}} < n/2$.

World 3: There are two clients, c_1 and c_2. Validators in P are honest, and those in Q are adversarial. Adversarial validators behave as described in world 1. Since there can be at most $m\lfloor n/2 \rfloor - 1$ bits of data on I, and tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, consists of $m\lfloor n/2 \rfloor$ bits, c_1 does not learn and cannot output all of tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, as part of its chain by slot T_{fin}. As there are at least f_{l} adversarial validators, either of the following cases can happen:

- c_1 outputs tx_i^P, $i \in [\lceil n/2 \rceil]$, as part of its chain by slot T_{fin}.
- c_1 does not output tx_i^P, $i \in [\lceil n/2 \rceil]$, by slot T_{fin}.

World 4: There are again two clients, c_1 and c_2. Validators in P are adversarial, and those in Q are honest. Adversarial validators behave as described in world 2. Since there can be at most $m\lfloor n/2 \rfloor - 1$ bits of data on I, and tx_i^P, $i \in [\lceil n/2 \rceil]$, consists of $m\lceil n/2 \rceil$ bits, c_1 does not learn and cannot output all of tx_i^P, $i \in [\lceil n/2 \rceil]$, as part of its chain by slot T_{fin}. As there are at least f_{l} adversarial validators, either of the following cases can happen:

- c_1 outputs tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, as part of its chain by slot T_{fin}.
- c_1 does not output tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, by slot T_{fin}.

As the worlds 3 and 4 are indistinguishable by c_2 except with negligible probability, it outputs the same, potentially empty, chain in both worlds by T_{fin} with overwhelming probability. Suppose c_2 did not output all of tx_i^P, $i \in [\lceil n/2 \rceil]$, as part of its chain by slot T_{fin}. As this implies a violation of T_{fin}-liveness in world 3, it asks the validators for their transcripts, upon which the adversarial validators in Q reply with transcripts that omit the messages received from the set P. As $f_{\mathsf{a}} > 0$, by invoking the forensic protocol with the transcripts received, c_1 identifies a non-empty subset $S \subseteq Q$ of the adversarial validators, and outputs a proof that the validators in S have violated the protocol Π. However, in this case, an adversarial validator in world 4 can emulate the behavior of c_2 in world 3, and ask the validators for their transcripts. It can then invoke the forensic protocol with the transcripts, and output a proof that identifies the same subset $S \subseteq Q$ of validators as protocol violators. Since the two worlds are indistinguishable by c_2 except with negligible probability, upon receiving this proof, it would identify the honest validators in $S \subseteq Q$ as protocol violators in world 4 as well with non-negligible probability, which is a contradiction. By the same reasoning, if c_2 does not output tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, as part of its chain by slot T_{fin}, the adversary can construct a proof accusing an honest validator in world 3 with non-negligible probability, again a contradiction.

Next, suppose c_1 did not output all of tx_i^P, $i \in [\lceil n/2 \rceil]$, as part of its chain by slot T_{fin} in world 3. As this implies a violation of T_{fin}-liveness in world 3, it asks the validators for their transcripts, upon which the adversarial validators in Q reply with transcripts that omit the messages received from the set P. As $f_{\mathsf{a}} > 0$, by invoking the forensic protocol with the transcripts received, c_1 identifies a non-empty subset $S \subseteq Q$ of the adversarial validators, and outputs a proof that the validators in S have violated the protocol Π. However, in this case, an adversarial validator in world 4 can emulate the behavior of c_1 in world 3, and ask the validators for their transcripts. It can then invoke the forensic protocol with the transcripts, and output a proof that identifies the same subset $S \subseteq Q$ of validators as protocol violators. Since the two worlds are indistinguishable by c_2 except with negligible probability, upon receiving this proof, it would identify the honest validators in $S \subseteq Q$ as protocol violators in world 4 with non-negligible probability, which is a contradiction. By the same reasoning, if c_1 does not output tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, as part of its chain by slot T_{fin}, the adversary can, with non-negligible probability, construct a proof accusing an honest validator in world 3 in c_2's view, again a contradiction.

Finally, if c_1 outputs tx_i^P, $i \in [\lceil n/2 \rceil]$, and tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, as part of its chain respectively in worlds 3 and 4, and c_2 outputs both tx_i^P, $i \in [\lceil n/2 \rceil]$, and tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, as part of its chain in both worlds, if $f_{\mathsf{s}} > 0$, we reach a contradiction by the same reasoning presented for the worlds 1 and 2. Consequently, under the given conditions, no PoS protocol can provide a positive accountable safety and liveness resilience simultaneously even with access to a timestamping service.

Proof of part (2) Part (2) can be proved using just the worlds 1 and 2 above, which shows a contradiction given the assumptions $f_{\mathsf{l}} \geq n/2$ and $f_{\mathsf{s}} > 0$. □

C SECURITY PROOFS

Proof of Proposition 2. Since Bitcoin is safe with parameter k, without loss of generality, suppose $C_{r_1}^{\mathsf{c}_1} \preceq C_{r_2}^{\mathsf{c}_2}$. Let h_i, $i \in [m_1]$, and h_j, $j \in [m_2]$, $m_1 \leq m_2$, denote the sequence of hash values within checkpoint transactions in c_1's and c_2's views at slots r_1 and r_2 respectively. Note that the sequence observed by c_1 is a subset of the sequence observed by c_2. Let B_1 denote the first PoS block in $\mathsf{ckpt}_{r_1}^{\mathsf{c}_1}$ that is not available or not finalized in c_2's view at slot r_2, and define i_1 as the index of the hash of the block that extends or is the same as B_1. (If there is no such block B_1, $i_1 = \infty$.) Similarly, let B_2 denote the first PoS block in $\mathsf{ckpt}_{r_2}^{\mathsf{c}_2}$ that is not available or not finalized in c_1's view at slot r_1, and define i_2 as the index of the hash of the block that extends or is the same as B_2. (If there is no such block B_2, $i_2 = \infty$.) Note that if $i_1 < \infty$, $i_2 = \infty$, and if $i_2 < \infty$, $i_1 = \infty$, due to Line 16 of Algorithm 1. In the former case, *i.e.*, if $i_1 < i_2$, $\mathsf{ckpt}_{r_2}^{\mathsf{c}_2} \prec \mathsf{ckpt}_{r_1}^{\mathsf{c}_1}$. In the latter case, *i.e.*, if $i_2 \leq i_1$, $\mathsf{ckpt}_{r_1}^{\mathsf{c}_1} \preceq \mathsf{ckpt}_{r_2}^{\mathsf{c}_2}$.

If $r_2 \geq r_1 + \Delta$, any PoS block available in c_1's view at slot r_1 becomes available in c_2's view by slot r_2. Similarly, by the safety of Bitcoin with parameter k, if $r_2 \geq r_1 + \Delta$, $C_{r_1}^{\mathsf{c}_1} \preceq C_{r_2}^{\mathsf{c}_2}$. In this case, $i_1 = \infty$ and $\mathsf{ckpt}_{r_1}^{\mathsf{c}_1} \preceq \mathsf{ckpt}_{r_2}^{\mathsf{c}_2}$. Finally, by the safety of Bitcoin with parameter k, $C_{r_1}^{\mathsf{c}} \preceq C_{r_2}^{\mathsf{c}}$ for any $r_2 \geq r_1$. Thus, $\mathsf{ckpt}_{r_1}^{\mathsf{c}} \preceq \mathsf{ckpt}_{r_2}^{\mathsf{c}}$. □

Proof of Theorem 5. Suppose there are two clients c_1, c_2, and slots r_1, r_2 such that $\mathcal{L}_{r_1}^{\mathsf{c}_1}$ conflicts with $\mathcal{L}_{r_2}^{\mathsf{c}_2}$. Let B_1 and B_2 denote the earliest conflicting PoS blocks or bundles in $\mathcal{L}_{r_1}^{\mathsf{c}_1}$ and $\mathcal{L}_{r_2}^{\mathsf{c}_2}$ respectively. Without loss of generality, let r_1 and r_2 be the first slots B_1 and B_2 appear in c_1's and c_2's PoS chains respectively.

By the safety of Bitcoin, $C_{r_1}^{\mathsf{c}_1}$ is a prefix of $C_{r_2}^{\mathsf{c}_2}$ or vice versa with overwhelming probability. By Proposition 2, $\mathsf{ckpt}_{r_1}^{\mathsf{c}_1}$ is a prefix of $\mathsf{ckpt}_{r_2}^{\mathsf{c}_2}$ or vice versa.

We first consider the case where at least one of the blocks is a bundle. Without loss of generality, let B_1 be a bundle and B denote the common parent of B_1 and B_2. Let b denote the Bitcoin block with the liveness transaction that triggered the rollup mode, during which h_1, the hash of B_1, and the corresponding $n/2$ signatures appeared in $C^{\mathsf{c}_1}_{r_1}$. At slot r_1, the prefix of c_1's PoS chain ending at B_1 consists of two pieces: (i) a checkpointed chain outputted using the prefix of $C^{\mathsf{c}_1}_{r_1}$ that ends at the $2k$-th block extending b, (ii) bundles extending the checkpointed chain until B_1. If B is also a bundle, the next block in $\mathcal{L}^{\mathsf{c}_2}_{r_2}$ following B, *i.e.* B_2, has to be the same block as B_1 due to the consistency of $C^{\mathsf{c}_1}_{r_1}$ and $C^{\mathsf{c}_2}_{r_2}$. However, as $B_2 \neq B_1$, B cannot be a bundle.

If B is not a bundle, it must be the last PoS block in c_1's checkpointed chain preceding B_1, implying that B_1 is the first bundle in $\mathcal{L}^{\mathsf{c}_1}_{r_1}$. However, this again implies $B_1 = B_2$ since c_1 and c_2 agree on the first block of the rollup mode whenever $C^{\mathsf{c}_1}_{r_1}$ and $C^{\mathsf{c}_2}_{r_2}$ are consistent. As this is a contradiction, with overwhelming probability, neither of the blocks B_1 or B_2 can be a bundle.

Finally, if neither of B_1 and B_2 is a bundle, proof of slashable safety proceeds as given for Theorem 2. □

Proof of Theorem 6. As the number of honest active validators is $> n/2$ at all times, no PoS block or bundle hash with an unavailable preimage can acquire signatures from over $n/2$ active validators of the corresponding epoch. Hence, there cannot be any emergency break and the clients do not stop outputting new PoS blocks or bundles as part of their PoS chains while new checkpoints for available and finalized PoS blocks continue to appear in Bitcoin.

Consider a transaction tx input to the honest validators at some slot r by $\mathcal{Z}$. If tx does not appear in $\mathcal{L}^{\mathsf{v}}_{r+T_{\mathsf{tm}}}$ in an honest validator v's view, v sends a liveness transaction to Bitcoin containing tx at slot $r + T_{\mathsf{tm}}$. Let R, polynomial in the security parameter λ, denote the confirmation latency of Bitcoin with parameter k. Then, by the security of Bitcoin, with overwhelming probability, for all clients c, the liveness transaction appears in $C^{\mathsf{c}}_{r_1}$ within the same Bitcoin block b by slot $r_1 = r + T_{\mathsf{tm}} + R$,

Once a client c observes b become k deep in its Bitcoin chain, which happens by some slot less than $r_1 + R$, it sends a checkpoint transaction for the block at the tip of its PoS chain. Subsequently, b becomes at least $2k$ deep in c's Bitcoin chain by some slot $r_2 \leq r_1 + 2R$ with overwhelming probability. In this case, there are two possibilities: (1) $\mathsf{tx} \in \mathsf{ckpt}^{\mathsf{c}}_{r_2}$, or (2) $\mathsf{tx} \notin \mathsf{ckpt}^{\mathsf{c}}_{r_2}$. If (1) happens, then for all clients c, it holds that $\mathsf{tx} \in \mathcal{L}^{\mathsf{c}}_{r_2}$. If (2) happens, then each client c enters the rollup mode by slot r_2. Once in the rollup mode, an honest validator v prepares a bundle of transactions containing tx by slot r_2, which is viewed by all clients and signed by all honest validators by slot $r_2 + \Delta$. Upon gathering these signatures, *i.e.*, by slot $r_2 + 2\Delta$, v sends the hash of the bundle and the signatures to Bitcoin. By the security of Bitcoin, the hash and the signatures appear in the Bitcoin chain of each client c at the same position by slot $r_3 = r_2 + 2\Delta + R$ with overwhelming probability. Since the PoS protocol is accountable, an honest validator can never be identified as a protocol violator and can never become slashable in the view of any client. This implies that the signatures from the honest validators suffice to pass the $n/2$ threshold. Consequently, $\mathsf{tx} \in \mathcal{L}^{\mathsf{c}}_{r_3}$ for each client c.

Finally, setting $T_{\mathsf{fin}} = r_3 - r = 2\Delta + 4R + T_{\mathsf{tm}}$, which is polynomial in λ, we observe that unless there is a safety violation, T_{fin}-liveness holds for all clients. □

Proof of Corollary 2. By Theorem 6, if the number of adversarial active validators is less than $n/2$ at all times, the Babylon protocol of Section 5.2 with fast finalization satisfies T_{fin}-liveness, where T_{fin} is a polynomial in the security parameter λ. Thus, if a transaction tx is input to an honest validator at some slot r, then for all clients c that follow the fast finalization rule, tx will be in $\mathcal{L}^{\mathsf{c}}_{r+T_{\mathsf{fin}}}$. If tx was included in a bundle, then once tx enters $\mathcal{L}^{\mathsf{c}}_{r+T_{\mathsf{fin}}}$, it is also in $\mathsf{ckpt}^{\mathsf{c}}_{r+T_{\mathsf{fin}}}$. On the other hand, if tx was included in a finalized and available PoS block, it might be the case that the block extends $\mathsf{ckpt}^{\mathsf{c}}_{r+T_{\mathsf{fin}}}$, but is not checkpointed yet.

At the end of each epoch, an honest validator v sends a checkpoint transaction for the finalized and available PoS blocks extending its checkpointed chain. As the PoS protocol is accountable, an honest validator can never be identified as a protocol violator and can never become slashable in the view of any client. Thus, the signatures from the honest validators on v's checkpoint are always viewed as valid by all clients. Let R, polynomial in the security parameter λ, denote the confirmation latency of Bitcoin with parameter k. Let T, polynomial in the security parameter λ, denote an upper bound on the duration of epochs.

Suppose the validator v sent its signed checkpoint for a finalized and available PoS block containing tx at the end of the epoch where it observed tx in its PoS chain, *e.g.*, at some slot $r' < r + T_{\mathsf{fin}} + T$. Then, with overwhelming probability, for any client c, v's signed checkpoint is in $C^{\mathsf{c}}_{r'+R}$. As the valid signatures on the checkpoint by the honest validators pass the $n/2$ threshold, with overwhelming probability, either $\mathsf{tx} \in \mathsf{ckpt}^{\mathsf{c}}_{r'+R}$ for all clients c, or there is a checkpoint in the prefix of v's checkpoint for conflicting PoS blocks. In the latter case, the protocol enters the rollup mode as described in Section 5.4, in which case tx would be included in a bundle. Consequently, Babylon with slow finalization satisfies T'_{fin}-liveness, where $T'_{\mathsf{fin}} \leq T_{\mathsf{fin}} + T + R$ is a polynomial in the security parameter λ. □