Babylon:
Reusing Bitcoin Mining to Enhance Proof-of-Stake Security

Ertem Nusret Tas
Stanford University
nusret@stanford.edu

David Tse
Stanford University
dntse@stanford.edu

Fisher Yu
Hash Laboratories
fisher.yu@hashlabs.cc

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

ABSTRACT

Bitcoin is the most secure blockchain in the world, supported by the immense hash power of its Proof-of-Work miners, but consumes huge amount of energy. Proof-of-Stake chains are energy-efficient, have fast finality and accountability, but face several fundamental security issues: susceptibility to non-slashable long-range safety attacks, non-slashable transaction censorship and stalling attacks and difficulty to bootstrap new PoS chains from low token valuation. We propose Babylon, a blockchain platform which combines the best of both worlds by reusing the immense Bitcoin hash power to enhance the security of PoS chains. Babylon provides a data-available timestamping service, securing PoS chains by allowing them to timestamp data-available block checkpoints, fraud proofs and censored transactions on Babylon. Babylon miners merge mine with Bitcoin and thus the platform has zero additional energy cost. The security of a Babylon-enhanced PoS protocol is formalized by a cryptoeconomic security theorem which shows slashable safety and liveness guarantees.

1 INTRODUCTION

1.1 From Proof-of-Work to Proof-of-Stake

Bitcoin, the most valuable blockchain in the world, is secured by a Proof-of-Work protocol that requires its miners to solve hard math puzzles by computing many random hashes. As of this writing, Bitcoin miners around the world are computing in the aggregate roughly 1.4×10^{21} hashes per second. This hash power is the basis of Bitcoin's security, as an attacker trying to rewrite the Bitcoin ledger or censor transactions has to acquire a proportional amount of hash power, making it extremely costly to attack the protocol. However, this security also comes at a tremendous energy cost.

Many newer blockchains eschew the Proof-of-Work paradigm in favor of energy-efficient alternatives, the most popular of which is 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 Cardano, Algorand, Solana, Polkadot, Cosmos Hub and Avalanche among others. In addition to energy-efficiency, another major advantage of many PoS blockchains is their potential to hold protocol violators accountable and slash their stake as punishment.

Contact author: DT.

1.2 Proof-of-Stake Security Issues

Early attempts at proving the security of PoS protocols were made under the assumption that the majority or super-majority of the stake belongs to the honest parties (*e.g.*, [12, 16, 22, 23]). However, modern PoS applications such as cryptocurrencies are increasingly run by *economic agents* driven by financial incentives, that are not a priori honest. To ensure that these agents follow the protocol rules, it is crucial to incentivize honest behavior through economic rewards and punishments for protocol-violating behavior. Towards this goal, Buterin and Griffith [18] advocated the concept of *accountable safety*, the ability to identify validators who have provably violated the protocol in the event of a safety violation. In lieu of making an unverifiable honest majority assumption, this approach aims to obtain a *cryptoeconomic* notion of security for these protocols 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 important PoS protocols such as Gasper [19], the protocol supporting Ethereum 2.0, and Tendermint [15], the protocol supporting the Cosmos ecosystem. However, there are several fundamental limitations to achieving such trust-minimizing cryptoeconomic security for 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. thus implying a lack of *slashable safety*. For example, a long-range history-revision attack can be mounted using old coins after the stake is already withdrawn and therefore cannot be slashed [12, 17, 23, 24]. 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 long-range attacks is a long-known problem with PoS protocols, and there have been several approaches to deal with them (Section 2). In Section 4.1, we show a negative result: 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*. But since this type of checkpointing cannot be done very frequently, the stake lock-up period has to be set very long (*e.g.*, 21 days is a typical lock-up period for Cosmos zones), reducing 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:** Examples of these attacks include protocol stalling and transaction censorship. Unlike safety attacks where adversary double-signs conflicting blocks, such attacks are hard to be held accountable in a PoS protocol. For example, Ethereum 2.0 attempts to hold protocol stalling accountable by slashing non-voting attesters through a process called *inactivity leak* [14]. However, as we discuss in Section 4.2, an attacker can create an alternative chain and make it public much later, in which the honest attesters would not be voting and would therefore be slashed. Moreover, there is no known mechanism to hold the censoring of specific transactions accountable. In this context, we show in Section 4.2 that accountable liveness, let alone slashable liveness, is impossible for any PoS protocol without external trust assumptions.

(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.

1.3 Reusing Bitcoin Mining to Provide External Trust

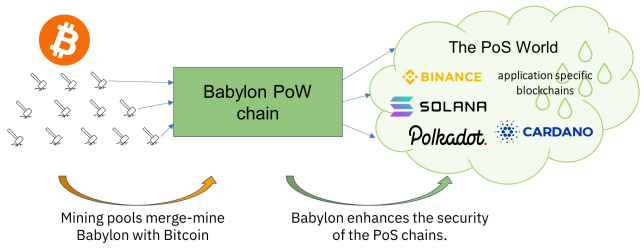

Figure 1: The Babylon architecture. Babylon is a PoW chain merge mined by Bitcoin miners and used by the PoS protocols to obtain slashable security.

The PoS security issues above cannot be resolved without an external source of trust. But a strong source of trust already exists in the blockchain ecosystem: Bitcoin mining. Built on this observation, we propose Babylon, a blockchain platform which *reuses* the existing Bitcoin hash power to enhance the security for any PoS chain which uses the platform (Figure 1). Babylon is a PoW blockchain on which multiple PoS chains can post information and use the ordering and availability of that information to obtain cryptoeconomic security guarantees while retaining all of their desirable features such as fast finalization. Babylon is mined by the Bitcoin miners via a technique called *merge mining* [2, 3, 5], which enables the reuse of the same hash power on multiple chains (cf. Appendix B). Thus, by reusing the existing Bitcoin hash power, Babylon enhances the security of PoS chains at no extra energy cost.

1.4 Babylon: A Data-available Timestamping Service

A major reason behind the security issues of PoS protocols described in Section 1.2 is the lack of a reliable *arrow of time*. For example, long-range attacks exploit the inability of late-coming nodes to distinguish between the original chain and the adversary's history-revision chain that is publicized much later [23, 24]. Babylon resolves these security limitations by providing a *data-available timestamping service* to the PoS chains.

To obtain slashable security guarantees, full nodes of the PoS protocols post commitments of protocol-related messages, *e.g.*, finalized PoS blocks, fraud proofs, censored PoS transactions, onto the Babylon PoW chain (Figure 2). Babylon checks if the messages behind these commitments are available and provides a timestamp for the messages by virtue of the location of its commitments in the Babylon chain. This enables PoS nodes, including the late-coming ones, to learn the time and order in which each piece of data was first made public. PoS nodes can then use the timestamps on this data in conjunction with the consensus logic of the native PoS protocol to resolve safety conflicts, identify protocol violators and slash them before they can withdraw their stake in the event of safety or liveness violations. For example, whenever there is a safety violation in a PoS protocol causing a fork, timestamps on Babylon can be used to resolve the fork by choosing the branch with the earlier timestamp (Figure 2). Whenever there is a proof of double-signing or liveness violation recorded on Babylon, responsible PoS participants can be irrefutably identified and slashed using the information on Babylon.

To resolve the security issues of PoS protocols, Babylon, *in addition* to timestamping PoS data, has to guarantee that this data is *available*, *i.e.*, has been publicized to the honest PoS nodes, when it is timestamped. Otherwise an adversary that controls the majority stake can mount a non-slashable long range attack by posting succinct commitments of finalized, yet private PoS blocks on Babylon and releasing the block data to the public much later, after the adversarial stake is withdrawn (Figure 3). Thus, Babylon must also provide the additional functionality of checking for the availability of the PoS data it is timestamping. This functionality cannot be satisfied by solutions that timestamp PoS data by posting its succinct commitments directly on Bitcoin or Ethereum [32], whereas posting all of the data raises scalability concerns (*cf.* Section 2 for more discussion). Thus, it necessitates a new PoW chain, Babylon, whose miners are instructed to check for the availability of the timestamped data in the view of the PoS nodes.

Babylon is *minimalistic* in the sense that it provides a data-available timestamping service *and no more*. It does not execute the transactions on the PoS chains, does not keep track of their participants and in fact does not even need to understand the semantics of the PoS block content. It also does not store the PoS data. All Babylon needs to do is to check the availability of the PoS data it is timestamping and make this data public to the PoS nodes, which can be done efficiently (eg. [7, 36]). This minimalism allows the scalability of Babylon to support the security of many PoS chains simultaneously.

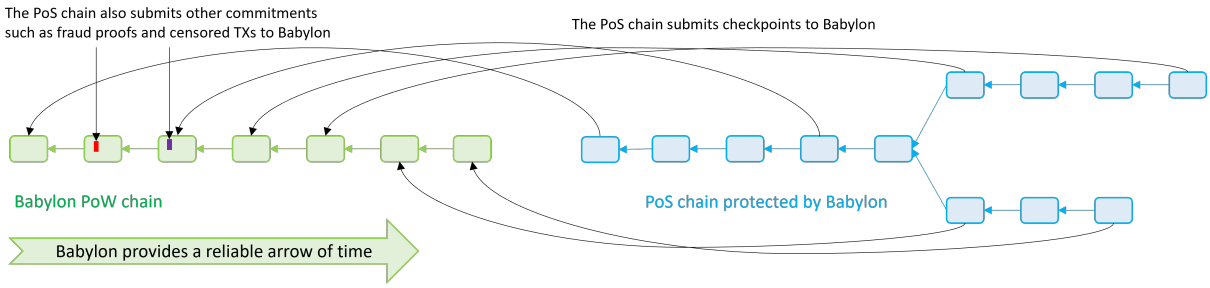

Figure 2: Timestamping on Babylon. Babylon PoW chain provides a record of the times events happen on the PoS chains, thus enabling PoS nodes to resolve safety violations on the PoS chain. For instance, since the checkpoints of the blocks in the top branch of the PoS chain appears earlier on Babylon than the checkpoints of the blocks in the bottom branch, the canonical PoS chain in this case follows the top branch.

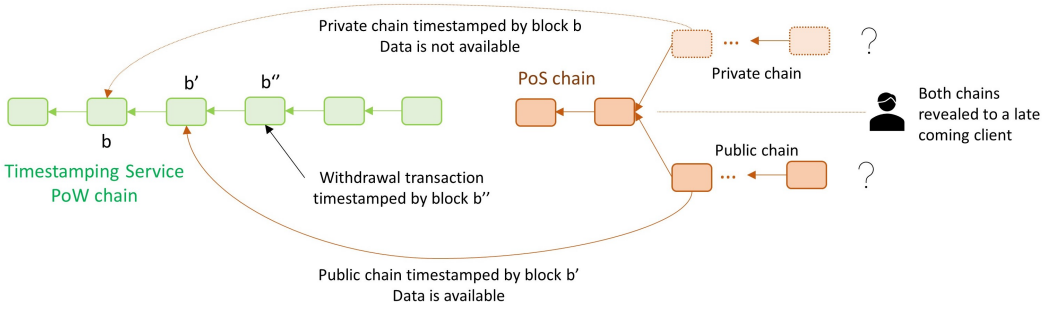

Figure 3: An adversary that controls a super-majority of stake finalizes PoS blocks on two conflicting chains. It keeps the blocks on one of the chains, the attack chain, private, and builds the other, the canonical one in public. It also posts succinct commitments of blocks from both chains to a timestamping service, *e.g.*, Bitcoin or Ethereum. Commitments from the attack chain are ignored by the nodes since their data is not available and might be invalid. Finally, after withdrawing its stake and timestamping it on the PoW chain, the adversary publishes the private attack chain. From the perspective of a late-coming node, the attack chain *is* the canonical one as it has an earlier timestamp on block b and its contents are now available. However, this is a safety violation. Moreover, as the adversary has withdrawn its stake, it cannot be slashed or financially punished.

1.5 High-level Description of the Protocol

The Babylon architecture consists of two major components: the Babylon PoW chain (Babylon for short), merge-mined by Bitcoin miners, hereafter referred to as Babylon miners, and the Babylon-enhanced PoS protocols each maintained by a distinct set of PoS nodes (nodes for short). The Babylon-enhanced PoS protocol is constructed on top of a standard accountably-safe PoS protocol which takes PoS transactions as input and outputs a chain of *finalized* blocks called the PoS chain.

Babylon miners are *full nodes* towards Babylon, *i.e.*, download and verify the validity of all Babylon blocks. They also download the data whose commitments are sent to Babylon for timestamping, but do not validate it. PoS full nodes download and verify the validity of all PoS blocks in their respective chains. All PoS nodes act as *light clients* towards Babylon. Thus, they rely on Babylon miners to hand them valid Babylon blocks, and download only the messages pertaining to their PoS protocol from Babylon. A subset of PoS full nodes, called validators, lock their funds for staking and run the PoS consensus protocol by proposing and voting for blocks[1].

PoS nodes can send different types of messages to Babylon for timestamping (Figure 2). These messages are typically succinct commitments of PoS data such as finalized PoS blocks, censored transactions and fraud proofs identifying misbehaving PoS validators. Finalized PoS blocks whose commitments are included in Babylon are said to be checkpointed by the Babylon block that includes the commitment.

PoS nodes use the timestamped information on Babylon to resolve safety violations and slash protocol violators as described below (*cf.* Section 5 for details):

(1) **Fork-choice rule:** If there is a fork on the PoS chain due to a safety violation, then the canonical chain of the Babylon-enhanced PoS protocol follows the fork whose first checkpointed block has the earlier timestamp on Babylon (Figure 2). Thus, Babylon helps resolve safety violations on the PoS chains.

[1]Not every PoS full node is necessarily a validator. Full nodes that are not validators still download PoS blocks and process their transactions to obtain the latest PoS state.

(2) **Stake withdrawal and slashing for double-signing:** A validator can input a withdrawal request into the PoS protocol to withdraw its funds locked for staking. A stake withdrawal request is granted, as a transaction in a later PoS block, if the PoS block containing the request is timestamped, *i.e.* checkpointed, by a Babylon block that is at least k_w-block deep in the longest Babylon chain and no *fraud proof* of the validator double-signing has appeared on Babylon (Figure 5). On the other hand, if a fraud proof against this validator exists, then the validator is slashed. Here, k_w determines the stake withdrawal delay.

(3) **Slashing for transaction censoring:** If a node believes that a transaction is being censored on the PoS chain, it can submit the transaction to Babylon along with a *censorship complaint*. Upon observing a complaint on the Babylon chain, validators stop proposing or voting for PoS blocks that do not contain the censored transaction. PoS blocks excluding the censored transaction and checkpointed on Babylon after the censorship complaint are labeled as *censoring*. Validators that propose or vote for censoring blocks are slashed.
Although Babylon is a timestamping service, granularity of time as measured by its blocks depends on the level of its security. For instance, if the adversary can reorganize the last k_c blocks on the Babylon chain, it can delay the checkpoints of PoS blocks sent to the miners before a censorship complaint until after the complaint appears on Babylon. Thus, to avoid slashing honest validators that might have voted for these blocks, PoS blocks checkpointed by the first k_c Babylon blocks following a censorship complaint are not labelled as censoring. Since this gives the adversary an extra k_c blocktime to censor transactions, k_c is an upper bound on the worst-case finalization latency of transactions when there is an active censorship attack.

(4) **Slashing for stalling:** If a PoS node believes that the growth of the PoS chain has stalled due to missing proposals or votes, it submits a *stalling evidence* to Babylon. Upon observing a stalling evidence, validators record proposals and votes exchanged over the next round of the PoS protocol on Babylon. Those that fail to propose or vote, thus whose protocol messages do not appear on Babylon within k_c blocks of the stalling evidence, are slashed. Again, the grace period of k_c blocks protects honest validators from getting slashed in case the adversary delays their messages on Babylon.

We note that the Babylon checkpoints of finalized PoS blocks are primarily used to resolve safety violations and slash adversarial validators in the event of safety and liveness attacks. Hence, Babylon does not require any changes to the native finalization rule of the PoS protocols using its services, and preserves their fast finality propert in the absence of censorship or stalling attacks.

Majority of the Babylon-specific add-ons used to enhance PoS protocols treat the PoS protocol as a blackbox, thus are applicable to any propose-vote style accountably-safe PoS protocol. In fact, the only part of the Babylon-specific logic in Section 5 that uses the Tendermint details is the part used to slash stalling attacks on liveness. Hence, we believe that Babylon can be generalized to apply to PoS protocols such as PBFT [20], HotStuff [35], and Streamlet [21].

1.6 Security Theorem

Using Babylon, accountably-safe PoS protocols can overcome the limitations highlighted in Section 1.2 and obtain slashable security. To demonstrate this, we augment Tendermint [16] with Babylon-specific add-ons and state the following security theorem for Babylon-enhanced Tendermint. Tendermint was chosen as it provides the standard accountable safety guarantees [15].

The Babylon chain is said to be secure for parameter r, if the r-deep prefixes of the longest Babylon chains in the view of honest nodes satisfy safety and liveness as defined in [25]. A validator v is said to become slashable in the view of an honest node if v was provably identified as a protocol violator and has not withdrawn its stake in the node's view. Formal definitions of safety and liveness for the Babylon-enhanced PoS protocols, slashability for the validators and security for the Babylon chain are given in Section 3.

Theorem 1. *Consider a synchronous network where message delays between all nodes are bounded, and the average time between two Babylon blocks is set to be much larger than the network delay bound. Then, Babylon-enhanced Tendermint satisfies the following security properties if there is at least one honest PoS node at all times:*

- *Whenever the safety of the PoS chain is violated, either of the following conditions must hold:*
  - ***S1:** More than* $1/3$ *of the active validator set becomes slashable in the view of all honest PoS nodes.*
  - ***S2:** Security of the Babylon chain is violated for parameter* $k_w/2$.
- *Whenever the liveness of the PoS chain is violated for a duration of more than* $\Theta(k_c)$ *block-time as measured in mined Babylon blocks, either of the following conditions hold:*
  - ***L1:** More than* $1/3$ *of the active validator set becomes slashable in the view of all honest PoS nodes.*
  - ***L2:** Security of the Babylon chain is violated for parameter* $k_c/2$.

Proof of Theorem 1 is given in Appendix D. Note that this is a *cryptoeconomic* security theorem as it explicitly states the slashing cost to the attacker to cause a safety or liveness violation (conditions **S1** and **L1** respectively). There is no trust assumption on the PoS validators such as having an honest majority. There are trust assumptions on the Babylon miners (as reflected by conditions **S2** and **L2**), but these trust assumptions are also quantifiable in terms of the economic cost of the attacker to acquire the hash power to reorganize certain number of Babylon blocks.

Specific implications of the theorem are:

(1) **Slashable safety:** Conditions **S1** and **S2** together say that, when the PoS chain is supported by Babylon, the attacker must reorganize $k_w/2$ blocks on Babylon if it does not want to be slashed for a safety attack on the PoS chain. Since k_w is the stake withdrawal delay and determines the liquidity of the staked funds, **S2** quantifies the trade-off between stake liquidity and the attacker's cost. When reorganization cost of Babylon is high as is the case for a chain merge-mined with Bitcoin, this

trade-off also implies much better liquidity than in the current PoS chains (*e.g.*, 21 days in Cosmos).

(2) **Slashable liveness:** Conditions **L1** and **L2** together say that, with Babylon's support, the attacker must reorganize $k_c/2$ blocks on Babylon if it does not want to be slashed for a liveness attack on the PoS chain. Since k_c is the worst-case latency for the finalization of transactions under an active liveness attack, **L2** quantifies the trade-off between the worst-case latency under attack and the attacker's cost.

1.7 Bootstapping New PoS Chains

In a PoS protocol with slashable security, the attack cost is determined by the token value (*cf.* Section 1.2). On protocols with low initial valuation, this low barrier to attack pushes away high-valued applications that would have increased the token value. To break this vicious cycle, new PoS protocols can use Babylon as a second layer of finalization. For instance, PoS nodes can require the checkpoint of a finalized PoS block to become, *e.g.*, k blocks deep in the Babylon PoW chain before they consider it finalized. Then, to violate the security of the k-deep and finalized PoS blocks, the attacker must not only forgo its stake due to slashing, but also acquire the hash power necessary to reorganize Babylon for k blocks. For this purpose, it has to control over half of the total hash power for a duration $\Theta(k)$ blocks[2] [9]. Thus, by increasing k, the attack cost can be increased arbitrarily. Through this extra protection provided by Babylon, newer PoS protocols can attract high value applications to drive up their valuation.

Note that a large k comes at the expense of finalization latency, which no longer benefits from the fast finality of the PoS protocol. This tradeoff between latency and the parameter k can be made individually or collectively by the PoS nodes in a manner that suits the nodes' or the protocol's security needs. Moreover, once the valuation of the protocol grows sufficiently large, the parameter k can be decreased in proportion to the slashing costs, and eventually removed altogether, enabling the PoS protocols to regain fast finality after a quick bootstrapping period.

1.8 Outline

Section 2 surveys the related work and analyzes alternative timestamping solutions in terms of their ability to provide slashable security. Section 3 introduces the model and the formal definitions used throughout the paper. Section 4 formalizes the impossibility results for slashable safety and accountable liveness of PoS protocols. Sections 5 and 6 give a detailed description of a Babylon-enhanced PoS protocol, and discuss Babylon's potential for scalability. Finally, Section 7 provides a reference design for Babylon-enhanced Tendermint using Cosmos SDK.

2 RELATED WORKS

2.1 Long-range Attacks

Among all the PoS security issues discussed in Section 1.2, long range history revision attacks is the most well-known, [12, 17, 23, 24] and several solutions have been proposed: 1) checkpointing via social consensus (*e.g.*, [11, 13, 17, 23]); 2) use of key-evolving signatures (*e.g.*, [12, 22, 27]); 3) use of verifiable delay functions, *i.e.*, VDFs (*e.g.*, [34]); 4) timestamping on an existing PoW chain like Ethereum [32] or Bitcoin [10].

2.1.1 Social Consensus. Social consensus refers to a trusted committee of observers, potentially distinct from the PoS nodes, which periodically checkpoint finalized PoS blocks that have been made available. It thus attempts to prevent long range attacks by making the adversarial blocks that are kept private distinguishable from those on the canonical PoS chain that contain checkpoints.

Social consensus suffers from vagueness regarding the size and participants of the checkpointing committee. For instance, a small oligarchy of trusted nodes would lead to a centralization of trust, anathema to the spirit of distributed systems. Conversely, a large committee would face the problem of reaching consensus on checkpoints in a timely manner. Moreover, the question of who belongs in the committee complicates the efforts to quantify the trust assumptions placed on social consensus, in turn making any security valuation prone to miscalculations. For instance, a re-formulation of Theorem 1 in this setting would claim slashable security as long as the social consensus checkpoints are 'trustworthy', without much insight on how to value this trust in economic terms. In comparison, the trust placed on Babylon is quantifiable and equals the cost of acquiring the hash power necessary to reorganize the Babylon PoW chain, which is well-known [4].

2.1.2 Key-evolving Signatures. Use of key-evolving signatures requires validators to forget old keys so that a history revision attack using old coins cannot be mounted. However, an adversarial majority can always record their old keys and use them to attack the canonical chain by creating a conflicting history revision chain once they withdraw their stake. This way, they can cause a safety violation, yet upon detection, avoid any slashing of the stake as it was already withdrawn. Hence, key-evolving signatures cannot prevent long range attacks without an honest majority assumption, thus cannot provide slashable security.

Security has been shown for various PoS protocols [12, 22] using key-evolving signatures under the honest majority assumption, which ensures that the majority of 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 become useful later on. 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.

2.1.3 VDFs. As was the case with key-evolving signatures, VDFs cannot prevent long range attacks without the honest majority assumption, thus cannot provide slashable security. For instance, an adversarial majority can build multiple conflicting PoS chains since the beginning of time, and run multiple VDF instances simultaneously for both the public PoS chain and the attack chains that are kept private. After withdrawing their stakes, these validators can publish the conflicting attack chains with the correct VDF proofs. Thus, VDFs cannot prevent an adversarial majority from causing a safety violation at no slashing cost.

[2]Reorganizing one Bitcoin block costs about USD $0.5M, as of this writing [4]. Perhaps more importantly, 0% of this hash power is available on nicehash.com.

Another problem with VDFs is the possibility of finding faster functions [8], which can then be used to mount a long range attack, even under an honest majority assumption.

2.1.4 Timestamping on Bitcoin or Ethereum. Timestamping directly on an existing PoW chain, *e.g.*, Bitcoin [10] or Ethereum [32] suffers from the fact that these PoW chains do not check for the availability of the committed data in the view of the PoS nodes, thus, as a solution, is vulnerable to the attack on Figure 3. To mitigate the attack, either all of the committed PoS data, *e.g.*, all the PoS blocks, must be posted on the PoW chain to guarantee their availability, or an honest majority must certify the timestamps to prevent unavailable PoS blocks from acquiring timestamps on the PoW chain.

The first mitigation creates scalability issues since in this case, miners must not only verify the availability of the PoS data, which can be done through lightweight methods such as data availability sampling [7, 36], but also store the data of potentially many different PoS protocols indefinitely.

The second mitigation was implemented by [32] through an Ethereum smart contract which requires signatures from over 2/3 of the PoS validators to timestamp changes in the validator sets. Since [32] assumes honest majority, signatures from 2/3 of the validators imply that the signed changes in the validator set are due to transactions within available and valid PoS blocks. However, the second mitigation cannot be used to provide cryptoeconomic security without trust assumptions on the validators. In contrast, Babylon miners are modified to do data availability checks, which enables Babylon to rely on the miners themselves rather than an honest majority of PoS validators for data availability.

2.2 Hybrid PoW-PoS Protocols

A Babylon-enhanced PoS protocol is an example of a *hybrid PoW-PoS protocol*, where consensus is maintained by both PoS validators and PoW miners. One of the first such protocols is the Casper FFG finality gadget used in conjunction with a longest chain PoW protocol [18]. 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 Ethereum 2.0 [19] and Polkadot [33]. Babylon 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 of Babylon that combines an existing PoW protocol with PoS protocols also leverages off insights from a recent line of work on secure compositions of protocols [28, 29, 31].

2.3 Blockchain Scaling Architectures

Scaling blockchains is a longstanding problem. A currently popular solution on Ethereum and other platforms is the shift of transaction execution from a base blockchain to *rollups*, which execute state transitions and post state commitments on the blockchain. Emerging projects like Celestia [6] take this paradigm further by removing execution entirely from the base blockchain and having it provide only data availability and ordering. In contrast, the main goal of the present work is not on scalability but on enhancing existing or new PoS protocols with slashable security. While rollups derive their security entirely from the base blockchain, the PoS protocols are autonomous and have their own validators to support their security. In this context, the main technical challenge of this work is how to design the architecture such that the Babylon PoW chain augments the existing security of the PoS protocols with slashable security guarantees. Nevertheless, to scale up our platform to support many PoS protocols, we can leverage off scaling techniques such as efficient data availability checks [7, 36] and sharding [1]. More discussions can be found in Section 6.

3 MODEL

Validators: PoS nodes that run the PoS consensus protocol are called validators. Each validator is equipped with a unique cryptographic identity. Validators are assumed to have synchronized clocks.

There are two sets of validators: passive and active. Validators *stake* a certain amount of coins to become active and participate in the consensus protocol. Although staked coins cannot be spent, active validators can send *withdrawal requests* to withdraw their coins. Once a withdrawal request by an active validator is finalized by the PoS protocol, *i.e.*, included in the PoS chain, the validator becomes passive and ineligible to participate in the consensus protocol. The passive validator is granted permission to withdraw its stake and spend its funds once a *withdrawal delay* period has passed following the finalization of the withdrawal request.

Let n denote the total number of validators that are active at any given time. The number of passive validators is initially zero and grows over time as active validators withdraw their stakes and become passive.

Environment and Adversary: Transactions are input to the validators by the environment $\mathcal{Z}$. Adversary $\mathcal{A}$ is a probabilistic poly-time algorithm. $\mathcal{A}$ gets to corrupt a certain fraction of the validators when they become active, which are then called *adversarial* validators. It can corrupt any passive validator.

Adversarial validators surrender their internal state 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 PoS protocol as specified.

Networking: Validators can send each other messages. Network is synchronous, *i.e.* $\mathcal{A}$ is required to deliver all messages sent between honest validators, miners and nodes within a known upper bound Δ.

Accountability: We assume that the PoS protocol supported by Babylon has an accountable safety resilience of f_{a} (parameter d as defined in [26]), *i.e.*, f_{a} adversarial validators (that are potentially passive) are irrefutably identified by all PoS nodes as having violated the protocol in the event of a safety violation, and no honest validator can be identified as a protocol violator. Moreover, for the culpable validators, PoS nodes can create an irrefutable fraud proof showing that they violated the protocol.

Safety and Liveness for the PoS Protocols: Let PoSLOG_i^t denote the chain of finalized PoS blocks, *i.e.*, the PoS chain, in the view of a node i at time t. Then, safety and liveness for the PoS protocols are defined as follows:

Definition 1. *Let T_{fin} be a polynomial function of the security parameter σ of the PoS protocol Π. We say that Π is T_{fin}-secure if the PoS chain satisfies the following properties:*

- ***Safety:*** *For any time slots t, t' and honest PoS nodes i, j, either* PoSLOG_i^t *is a prefix of* $\mathsf{PoSLOG}_j^{t'}$ *or vice versa. For any honest PoS node i,* PoSLOG_i^t *is a prefix of* $\mathsf{PoSLOG}_i^{t'}$ *for all times t and t' such that $t \leq t'$.*
- $\mathsf{T}_{\mathsf{fin}}$***-Liveness:*** *If $\mathcal{Z}$ inputs a transaction* tx *to the validators at some time t, then,* tx *appears at the same position in* $\mathsf{PoSLOG}_i^{t'}$ *for any time $t' \geq t + T_{\mathsf{fin}}$ and for any honest PoS node i.*

A PoS protocol is said to satisfy f_s-safety or f_l-T_{fin}-liveness if it satisfies safety or T_{fin}-liveness whenever the number of active adversarial validators is less than or equal to f_s or f_l respectively.

Safety and Liveness for the Babylon Chain: Let $\mathsf{PoWChain}_i^t$ denote the longest, *i.e.*, canonical, Babylon chain in the view of a miner or PoS node i at time t. Then, safety and liveness for the Babylon chain are defined as follows:

Definition 2 (From [25]). *Babylon is said to be secure for parameter $r \geq 1$, $r \in \mathbb{Z}$, if it satisfies the following two properties:*

- ***Safety:*** *If a transaction* tx *appears in a block which is at least r-deep in the longest Babylon chain of an honest node or miner, then,* tx *will eventually appear and stay at the same position in the longest Babylon chain of all honest nodes or miners forever.*
- ***Liveness:*** *If a valid transaction* tx *is received by all honest miners for more than r block-time, then* tx *will eventually appear at an r-deep block in the longest Babylon chain of all honest nodes or miners. No invalid transaction ever appears at an r-deep block in the longest Babylon chain held by any honest node or miner.*

Thus, if Babylon satisfies security for the parameter r, r-deep prefixes of the longest chains held by the honest nodes are consistent with each other, grow monotonically, and transactions received by the honest miners for more than r block-time enter and stay in the longest Babylon chain observed by the honest nodes forever.

Slashability: Slashing refers to the process of financial punishment for the active validators detected as protocol violators.

Definition 3. *A validator* v *is said to be* slashable *in the view of a PoS node* c *if,*

(1) c *provably identified* v *as having violated the protocol for the first time at some time t, and,*
(2) v *has not withdrawn its stake in* c*'s view by time t.*

If a validator v is observed to be slashable by all honest PoS nodes, no transaction that spends the coins staked by v will be viewed as valid by the honest PoS nodes.

4 IMPOSSIBILITY RESULTS FOR PROOF-OF-STAKE PROTOCOLS

4.1 Safety Violation is not Slashable

Without additional trust assumptions, PoS protocols are susceptible to various flavors of long range attacks, also known as founders' attack, posterior corruption or costless simulation. In this context, [23, Theorem 2] formally shows that even under the honest majority assumption for the active validators, PoS protocols cannot have safety due to long range attacks without additional trust assumptions. Since slashable safety is intuitively a stronger result than guaranteeing safety under the honest majority assumption,

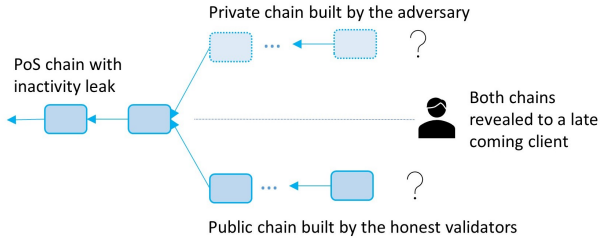

Figure 4: Inactivity leak attack. At the top is adversary's private attack chain. At the bottom is the public canonical chain built by the honest validators. Due to inactivity leak, honest & adversarial validators lose their stake on the attack and canonical chains respectively. A late-coming node cannot differentiate the canonical and attack chains.

[23, Theorem 2] thus rules out any possibility of providing PoS protocols with slashable safety without additional trust assumptions. This observation is formally stated by Theorem 3 in Appendix A.

4.2 Liveness Violation is not Accountable

Without additional trust assumptions, PoS nodes cannot identify any validator to have irrefutably violated the PoS protocol in the event of a liveness violation, even under a synchronous environment. To illustrate the intuition behind this claim, we show that inactivity leak [14], proposed as a financial punishment for inactive Ethereum 2.0 validators, can lead to the slashing of honest validators' stake with non-negligible probability. Consider the setup on Figure 4, where adversarial validators build a private attack chain that forks off the canonical one and stop communicating with the honest validators. As honest validators are not privy to the adversary's actions, they cannot vote for the blocks on the attack chain. Thus, honest validators are inactive from the perspective of the adversary and lose their stake on the attack chain due to inactivity leak. On the other hand, as the adversarial validators do not vote for the blocks proposed by the honest ones, they too lose their stake on the public, canonical chain (Figure 4). Finally, adversary reveals its attack chain to a late-coming node which observes two conflicting chains. Although the nodes that have been active since the beginning of the attack can attribute the attack chain to adversarial action, a late-coming node could not have observed the attack in progress. Thus, upon seeing the two chains, it cannot determine which of them is the canonical one nor can it irrefutably identify any validator slashed on either chains as adversarial or honest.

To formalize the impossibility of accountable liveness for PoS protocols, we extend the notion of accountability to liveness violations and show that no PoS protocol can have a positive *accountable liveness resilience*, even under a synchronous network with a static set of n active validators that never withdraw their stake. For this purpose, we adopt the formalism of [26] summarized below: During the runtime of the PoS protocol, validators exchange messages, *e.g.*, blocks or votes, and each validator records its view of the protocol by time t in an execution transcript. If a node observes that T_{fin}-liveness is violated, *i.e.*, a transaction input to the validators at some time t by $\mathcal{Z}$ is not finalized in the PoS chain in its view by time $t + T_{\mathsf{fin}}$, it invokes a forensic protocol: The forensic protocol takes

transcripts of the validators as input, and outputs an irrefutable proof that a subset of them have violated the protocol rules. This proof is sufficient evidence to convince any node, including late-coming ones, that the validators identified by the forensic protocol are adversarial.

Forensic protocol interacts with the nodes in the following way: Upon observing a liveness violation on the PoS chain, a node asks the validators to send their transcripts. It then invokes the forensic protocol with the transcripts received from the validator. Finally, through the forensic protocol, it constructs the irrefutable proof of protocol violation by the adversarial validators, and broadcasts this proof to all other nodes.

Using the formalization above, we next define *accountable liveness resilience* and state the impossibility theorem for accountable liveness on PoS protocols in the absence of additional trust assumptions:

Definition 4. *T_{fin}-accountable liveness resilience of a protocol is the minimum number f of validators identified by the forensic protocol to be protocol violators when T_{fin}-liveness of the protocol is violated. Such a protocol provides f-T_{fin}-*accountable-liveness.

Theorem 2. *Without additional trust assumptions, no PoS protocol provides both f_{a}-T_{fin}-accountable-liveness and f_{s}-safety for any $f_{\text{a}}, f_{\text{s}} > 0$ and $T_{\text{fin}} < \infty$.*

Proof is presented in Appendix A and generalizes the indistinguishability argument for the conflicting chains from the inactivity leak attack. It rules out any possibility of providing accountable liveness for PoS protocols even under a Δ-synchronous network and a static set of active validators.

A corollary of Theorem 2 is that PoS protocols cannot have a positive *slashable liveness resilience*:

Definition 5. *T_{fin}-slashable liveness resilience of a protocol is the minimum number f of validators that are slashable in the view of all PoS nodes per Definition 3 when T_{fin}-liveness of the protocol is violated. Such a protocol provides f-T_{fin}-slashable-liveness.*

Corollary 1. *Without additional trust assumptions, no PoS protocol provides both f_{s}-safety and f_{l}-T_{fin}-slahable-liveness for any $f_{\text{s}}, f_{\text{l}} > 0$ and $T_{\text{fin}} < \infty$.*

Proof of Corollary 1 follows from the fact that accountable liveness resilience of PoS protocols is zero without additional trust assumptions.

5 PROTOCOL

In this section, we specify how to obtain slashable security for any accountably-safe PoS protocol using Babylon-specific add-ons. Unless stated otherwise, the accountably-safe PoS protocol is treated as a black-box which takes PoS transactions as its input and outputs a chain of finalized PoS blocks containing these transactions. We assume that the consensus-related messages required to verify finalization of PoS blocks can be accessed by viewing the contents of the child blocks.

For concreteness, sections below focus on the interaction between the Babylon chain and a single PoS protocol.

5.1 Handling of Commitments by Babylon

PoS nodes timestamp messages by posting their commitments on Babylon. A commitment h is a succinct representation of a piece of data D[3]. Babylon miners receive commitments from the PoS nodes as pairs (tx, D), where tx is a Babylon transaction that contains the commitment h, and D is the associated data. Upon receiving such a pair, miners validate h against D, *i.e.*, check if h is a succinct commitment of D, on top of other transaction validation procedures for tx. However, since Babylon is a generic data-available timestamping service, miners do not check the syntax or semantics of the data D. If the validation succeeds, miners consider the commitment h valid and include tx in the next Babylon block mined. They do not include the data D in the Babylon blocks.

Whenever a miner propagates a Babylon transaction to its peers, either directly or as part of the block body, it also attaches the associated data, so that the peers receiving the transaction can also validate its availability. Since PoS nodes act as light clients of the Babylon chain and are connected to the peer-to-peer network of the Babylon miners, they also obtain the data broadcast by the miners. This ensures the availability of data across all honest PoS nodes once its commitment is validated and published by the Babylon miners.

Miners merge-mine the Babylon chain following the longest chain rule (*cf.* Appendix B for more details). A Babylon block is said to be *valid* in the view of a miner if the Babylon transactions included in the block are valid in the miner's view.

5.2 Generation and Validation of Commitments

There are two types of commitments: message commitments and checkpoints. Message commitment refers to the hash of the whole message. For example, to timestamp a list of censored transactions, a PoS node sends the hash of the list to the miners as the message commitment h and the whole list as the data D (Algorithm 2). Then, to validate the commitment, miners and PoS nodes check if the hash of the data matches the commitment (Algorithms 1 and 3).

Checkpoints are commitments of finalized PoS blocks. A single checkpoint can commit to multiple consecutive blocks from the same PoS chain. To post a checkpoint on Babylon for consecutive blocks $B_1, ..., B_n$, a PoS node first extracts the transaction roots txr_i, $i = 1, .., n$, from the header B_i.header of each block (Algorithm 2). Then, using a binding hash function H, it calculates the following commitment[4]:

$$h = H(B_1.\mathsf{header}||...||B_n.\mathsf{header}||\mathsf{txr}_1||...||\mathsf{txr}_n). \tag{1}$$

Finally, it sends the commitment h, *i.e.*, the checkpoint, to the miners along with the data D which consists of (i) the block headers B_1.header, .., B_n.header, (ii) the block bodies, and (iii) the transaction roots $\mathsf{txr}_1, .., \mathsf{txr}_n$ separately from the headers.

Upon receiving a checkpoint or observing one on Babylon, miners and PoS nodes parse the associated data D into the block headers, block bodies and transaction roots. Miners view the commitment as

[3] In a real-world implementation, commitments will also carry metadata such as a PoS chain identifier, submitter's signature and public key. The metadata will not be validated by the Babylon miners.

[4] In a real-world application, commitment also contains the header of block B_{n+1} as it contains the signatures necessary to verify the finalization of block B_n. We omit this fact above for brevity.

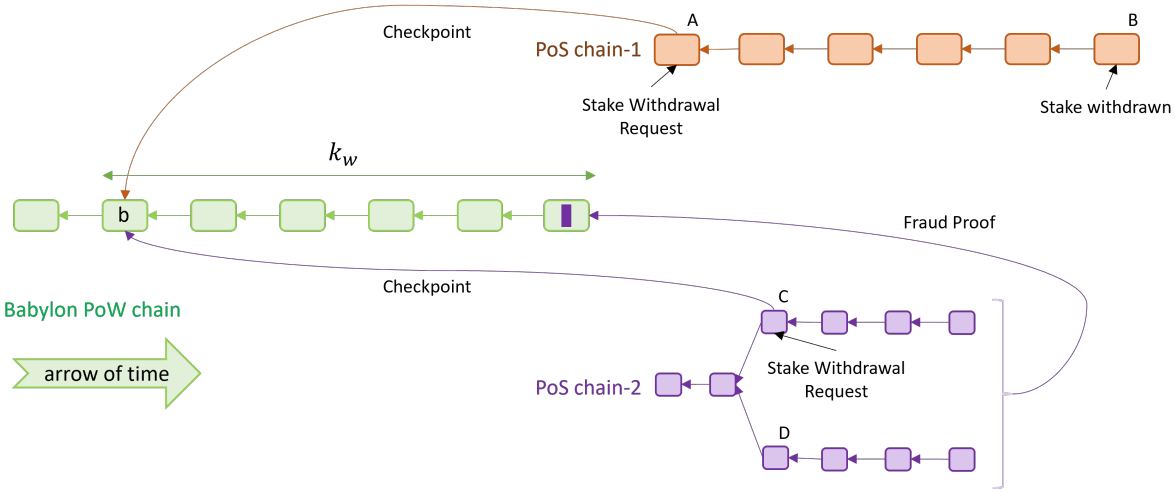

Figure 5: Delayed granting of withdrawal request and slashing. A validator for the PoS chain-1 sends a stake withdrawal request to its chain which is captured by the PoS block A. Block A is in turn checkpointed by the Babylon block b. This stake withdrawal request will only be granted and executed at a later PoS block B, where B is generated by a validator that observes the checkpoint of block A in block b become at least k_w deep in Babylon and that there is no fraud proof. On the other hand, the validator for the PoS chain-2 is not granted its withdrawal request and is slashed, since a fraud proof appears on Babylon before block b becomes k_w-deep.

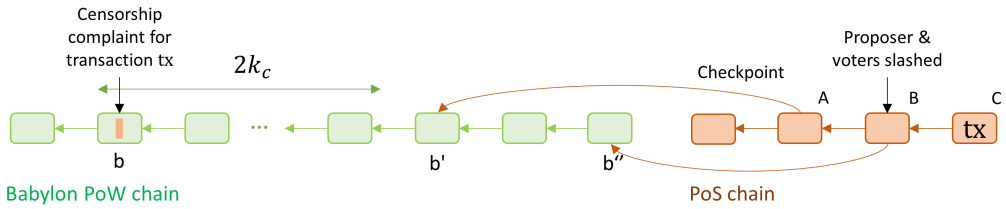

Figure 6: Slashing for censoring. A censored transaction tx is submitted to Babylon through a censorship complaint, and included in a Babylon block b. Suppose A is the last PoS block honest nodes proposed or voted for before they observed the censorship complaint on Babylon. Let b' denote the first Babylon block containing a checkpoint and extending b by at least $2k_c$ blocks. Since finalized PoS blocks are checkpointed frequently on Babylon, A will be checkpointed by b', or a Babylon block in its prefix. Then, any PoS block, *e.g.*, B, checkpointed by a Babylon block following b' must have been proposed or voted upon by the validators after they have observed the censorship complaint, and must include tx. However, B, which is checkpointed by b'' extending b', does not contain tx, thus, is a censoring block. Hence, validators that have proposed and voted for B will be slashed for censorship. Here, the $2k_c$ grace period on Babylon between b and b' ensures that the honest validators are not slashed for voting upon PoS blocks excluding the censored transactions, before they observed the censorship complaint.

valid if (i) expression (1) calculated using B_1.header, .., B_n.header and txr_1, .., txr_n matches the received commitment, and (ii) the roots txr_1, .., txr_n commit to the transactions in the bodies of the blocks B_1, .., B_n (Algorithm 1). PoS full nodes view the commitment as valid if conditions (i) and (ii) above are satisfied, (iii) txr_1, .., txr_n are the same as the transaction roots within the block headers B_1.header, .., B_n.header and (iv) the checkpointed PoS blocks are finalized in the given order within the PoS chain in their view (Algorithm 3). Although each header already contains the respective transaction root, a Babylon miner does not necessarily know the header structure of different PoS protocols. Thus, miners receive transaction roots separately besides the block headers and bodies. Note that miners cannot check if the transaction root txr_i it got for a block B_i is the same as the root within the header B_i.header. However, honest PoS nodes can detect any discrepancy between the transaction roots in the headers and those given as part of the data D, and ignore incorrect commitments.

Checkpoints are designed to enable light clients towards the PoS protocol to identify the checkpointed PoS blocks when they observe a commitment on Babylon. Unlike full nodes, PoS light clients do not download bodies of PoS blocks, thus cannot check if txr_i commits to the body of B_i. However, since they do download PoS block headers, these light clients can extract the transaction roots from the headers, calculate expression (1) and compare it

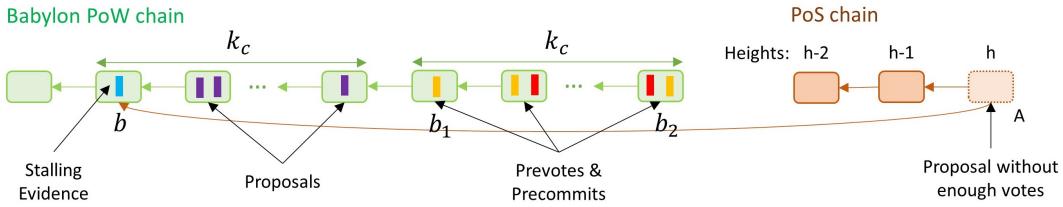

Figure 7: Slashing for stalling. A stalling evidence for height h, where validators failed to finalize any PoS block, is submitted to Babylon and captured by the Babylon block b. Upon observing b with a stalling evidence, validators enter a new Tendermint round whose messages are recorded on Babylon. During this round, they submit their proposals, prevotes and precommits to Babylon. Validators, whose proposals do not appear on Babylon between b and b_1, are deemed to be unresponsive and slashed for stalling the protocol. Similarly, validators, whose votes for the proposal selected from the interval $[b, b_1)$ are missing between b_1 and b_2, are slashed. Here, the k_c block intervals between b, b_1 and b_2 ensure that the proposals, prevotes and precommits submitted on time by the honest validators appear on Babylon in the appropriate interval, thus preventing honest validators from getting slashed for stalling.

against the commitment on Babylon to verify its validity. PoS light clients trust the Babylon miners to check if the transaction roots txr_i indeed commit to the data in the bodies of the checkpointed blocks.

5.3 Checkpointing the PoS Chain

Nodes send checkpoints of all finalized blocks on the PoS chain to the Babylon miners every time they observe the Babylon chain grow by k_c blocks[5]. We say that a Babylon block b checkpoints a PoS block B in the view of a node c (at time t) if (i) B is a finalized & valid block in the PoS chain in c's view, and (ii) b is the first block within the longest Babylon chain in c's view (at time t) to contain a valid checkpoint of B alongside other PoS blocks.

Checkpoints that do not include information about new PoS blocks are ignored by the PoS nodes during the interpretation of the commitments on Babylon. Thus, given two consecutive checkpoints on the Babylon chain that are not ignored by a PoS node, if they do not commit to conflicting PoS blocks, then the latter one must be checkpointing new PoS blocks extending those covered by the earlier one.

Fork-choice Rule: (Figure 2, Algorithm 4) If there are no forks on the PoS chain, *i.e.*, when there is a single chain, it is the canonical PoS chain.

If there are multiple PoS chains with conflicting finalized blocks, *i.e.*, a safety violation, in the view of a node c at time t, c orders these chains by the following recency relation : Chain A is *earlier* than chain B in c's view at time t if the first PoS block that is on A but not B, is checkpointed by an earlier Babylon block than the one checkpointing the first PoS block that is on B but not A, on c's canonical Babylon chain at time t. If only chain A is checkpointed in this manner on c's canonical Babylon chain, then A is earlier. If there are no Babylon blocks checkpointing PoS blocks that are exclusively on A or B, then the adversary breaks the tie for c. The canonical PoS chain PoSLOG_c^t is taken by c to be the earliest chain in this ordering at time t. Thus, Babylon provides a total order across multiple chains when there is a safety violation on the PoS chains.

[5]In reality, PoS nodes do not submit a commitment of all of the blocks on the PoS chain in their view. They submit commitments of only those blocks that were not captured by previous checkpoints on Babylon.

5.4 Stake Withdrawals and Slashing for Safety Violations

Since the PoS protocol provides accountable safety, upon observing a safety violation on the PoS chain, any node can construct a *fraud proof* that irrefutably identifies $n/3$ adversarial validators as protocol violators, and send it to Babylon. Fraud proof contains checkpoints for conflicting PoS blocks along with a commitment, *i.e.*, hash, of the evidence, *e.g.*, double-signatures, implicating the adversarial validators. Hence, it is valid as long as the checkpoints and the commitments are valid, and serves as an irrefutable proof of protocol violation by $n/3$ adversarial validators.

Stake withdrawal: (Figure 5, Algorithm 5) To withdraw its stake, a validator v first sends a special PoS transaction called the *withdrawal request* to the PoS protocol. Given k_w, v is granted permission to withdraw its stake in the view of a PoS node once the node observes that

(1) A block B on its canonical PoS chain containing the withdrawal request is checkpointed by a block b on its longest Babylon chain, *i.e.*, the longest Babylon chain in its view.
(2) There are k_w blocks building on b on its longest Babylon chain, where k_w, chosen in advance, determines the withdrawal delay.
(3) There does not exist a valid fraud proof implicating v in the node's longest Babylon chain.

Once the above conditions are also satisfied in v's view, it submits a *withdrawal transaction* to the PoS protocol, including a reference to the k_w-th Babylon block building on b. Honest nodes consider the withdrawal transaction included in a PoS block B' as *valid* if B' extends B, the block with the withdrawal request, and the above conditions are satisfied in their view.

Slashing for Safety Attacks: Stake of a validator becomes slashable in the view of any PoS node which observes that condition (3) above is violated. In this case, nodes that sent the fraud proofs on Babylon can receive part of the slashed funds as reward by submitting a *reward transaction* to the PoS chain.

5.5 Slashing for Liveness Violations

In the rest of this section, a validator or PoS node's Babylon chain, *i.e.*, the Babylon chain in the view of a PoS node or validator, refers to the $k_c/2$-deep prefix of the longest chain in their view. As a liveness violation can be due to either censorship, *i.e.*, lack of chain quality, or stalling, *i.e.*, lack of chain growth, we analyze these two cases separately:

5.5.1 Censorship Resilience. (Figure 6, Algorithm 6) PoS nodes send commitments of censored PoS transactions to Babylon via *censorship complaints*. A complaint is valid in the miners's view if the commitment matches the hash of the censored transactions.

Upon observing a valid complaint on its Babylon chain, a validator includes the censored PoS transactions within the new blocks its proposes unless they have already been included in the PoS chain or are invalid with respect to the latest PoS state. Similarly, among new PoS blocks proposed, validators vote only for those that include the censored transactions in the block's body or prefix if the transactions are valid with respect to the latest PoS state.

Suppose a censorship complaint appears within some block b on a validator p's Babylon chain (Figure 6). Let b' be the first block on p's Babylon chain that checkpoints a new PoS block and extends b by at least $2k_c$ blocks. Then, a PoS block B is said to be *censoring* in p's view if (i) it is checkpointed by a block b'', $b' \prec b''$ in p's Babylon chain, and (ii) B does not include the censored transactions in neither its body nor its prefix (*cf.* Algorithm 6 for a function that detects the censoring PoS blocks with respect to a censorship complaint).

5.5.2 Slashing for Censorship Attacks. Stake of a validator becomes slashable in an honest PoS node p's view if the validator proposed or voted for a PoS block B that is censoring in p's view (*e.g.*, block B in Figure 6).

5.5.3 Stalling Resilience. (Figure 7, Algorithm 7) A node detects that the PoS protocol has stalled if no new checkpoint committing new PoS blocks appears on its Babylon chain within $2k_c$ blocks of the last checkpoint. In this case, it sends a *stalling evidence* to Babylon. Stalling evidence is labelled with the smallest height h at which a PoS block has not been finalized yet and contains a checkpoint for the PoS blocks from smaller heights. Hence, it is valid in the miners' view if the included checkpoint is valid.

Stalling evidence signals to the validators that they should hereafter publish the PoS protocol messages, previously exchanged over the network, on Babylon until a new PoS block is finalized. For instance, in the case of Babylon-enhanced Tendermint, a stalling evidence on Babylon marks the beginning of a new *round* whose proposals and votes are recorded on Babylon. Thus, upon observing the first stalling evidence that follows the last checkpoint on Babylon by $2k_c$ blocks, validators stop participating in their previous rounds and enter a new, special Tendermint round for height h, whose messages are recorded *on-Babylon*. Each of them then pretends like the next round leader and sends a proposal message to Babylon for the new round.

In the rest of this section, we focus on Tendermint [16] as the Babylon-enhanced PoS protocol for the purpose of illustration. A summary of Tendermint is given in Appendix C.1. The following paragraphs explain how a Tendermint round is recorded on Babylon in the perspective of a validator p (*cf.* Algorithm 7). A detailed description of this can be found in Appendix C.2

Let b denote the Babylon block that contains the first stalling evidence observed by p (Figure 7). Let b_1 and b_2 denote the first blocks in p's Babylon chain that extend b by k_c and $2k_c$ blocks respectively. If a new checkpoint for a PoS block finalized at height h appears between b and b_1, p stops participating in the round on-Babylon and moves to the next height, resuming its communication with the other validators through the network. Otherwise, if there are $\geq 2f + 1$ *non-censoring* proposal messages signed by unique validators between b and b_1, p and every other honest validator selects the message with the largest validRound as the *unique* proposal of the round.

Once p decides on a proposal B and observes b_1 in its Babylon chain, it signs and sends prevote and precommit messages for B to Babylon. Upon seeing b_2 in its Babylon chain, p finalizes B if there are more than $2f + 1$ prevotes and precommits for B, signed by unique validators, between b_1 and b_2. In this case, b_2 is designated as the Babylon block that has checkpointed block B for height h. After finalizing B, p moves to the next height, resuming its communication with other validators through the network.

5.5.4 Slashing for Stalling Attacks. Consider the validator p and the on-Babylon Tendermint round described above and suppose there is no new checkpoint for a PoS block finalized at height h between b and b_1 (Figure 7). Then, if there are less than $2f + 1$ uniquely signed non-censoring proposals between b and b_1, stake of each validator with a censoring or missing proposal becomes slashable in p's view. Similarly, if there are less than $2f + 1$ uniquely signed prevotes or precommits between b_1 and b_2 for the proposal B selected by p, stake of each validator with a missing prevote or precommit for B between b_1 and b_2 becomes slashable in p's view.

To enforce the slashing of the validator's stake in the case of censorship or stalling, PoS nodes can submit a reward transaction to the PoS chain, upon which they receive part of the slashed funds.

No validator is slashed by the slashing rules for censorship or stalling if there is a safety violation on the PoS chains, in which case slashing for safety (*cf.* Section 5.4) takes precedence.

6 SCALABILITY OF THE PROTOCOL

Babylon protocol above can be used by different PoS protocols simultaneously, which raises the question of how much data Babylon miners can check for availability at any given time. To address this, we first review the three physical limits that determine the amount and speed of on-chain data generation by the PoS blockchains:

(1) hot storage capacity, which caps the amount of data generated before cold storage or chain snapshot have to kick in;
(2) execution throughput, which limits the data generation speed to how fast transactions and blocks can be created, validated, and executed;
(3) communication bandwidth, which limits the data generation speed to how fast transactions and blocks can be propagated throughout the P2P network.

Since Babylon does not permanently store[6] any PoS chain data, it does not have to match aggregated storage capacity of the PoS chains to provide data protection. Thus, there is no storage issue for Babylon to scale, namely, to support many PoS protocols.

On the other hand, Babylon's data processing speed, *i.e.* the speed with which miners validate data availability, must match the total data generation speed across the PoS chains. Currently, the data generation speed of individual PoS chains is mostly limited by the execution rather than communication bandwidth. As Babylon only downloads the PoS data without executing it, it should also able to accommodate many PoS chains from the speed perspective.

In the unlikely case that a certain PoS protocol is only limited by the communication bandwidth and thus generates large blocks frequently, Babylon could potentially apply sampling-based probabilistic data availability checks [7, 36] to significantly reduce the amount of data it needs to download and process per block, which is a promising future research direction.

7 REFERENCE DESIGN WITH COSMOS SDK

Cosmos is a well-known open-source blockchain ecosystem that enables customizable blockchains [30]. It also enables inter-blockchain communications by using Cosmos Hub (ATOM) as the trust anchor. Therefore, Cosmos provides both the tools through its SDK and the ecosystem required to demonstrate the implementation of a Babylon-enhanced PoS blockchain protocol. To this end, we first briefly review how essential Cosmos modules work together to protect the security of the Cosmos zones, *i.e.*, its constituent PoS blockchains, and then show how Babylon can enhance the security of these zones via a straightforward module extension.

7.1 Cosmos Overview

Cosmos encapsulates the core consensus protocol and networking in its Tendermint consensus engine, which uses Tendermint BFT underneath. Several interoperable modules have been built to work with this consensus engine together as a complete blockchain system. Each module serves a different functionality such as authorization, token transfer, staking, slashing, etc., and can be configured to meet the requirements of the application. Among these modules, the following are directly related to security:

- evidence module, which enables the submission by any protocol participant, and handling of the evidences for adversarial behaviors such as double-signing and inactivity;
- slashing module, which, based on valid evidence, penalizes the adversarial validators by means such as stake slashing and excluding it from the BFT committee;
- crisis module, which suspends the blockchain in case a predefined catastrophic incident appears, *e.g.*, when the sum of stakes over all the accounts exceed the total stake of the system;
- gov module, which enables on-chain blockchain governance in making decisions such as software updates and spending community funds.

We note that these modules currently are not able to handle the aforementioned attacks such as long range attacks and transaction censorship. Moreover, in case catastrophic incidents such as chain forking appear, system cannot recover from halt by itself. The incident can only be resolved via human intervention, which can be either proactive or reactive: Under proactive human intervention, stakeholders of the system regularly agree on and publish checkpoints on the blockchain to prevent long range attacks. Under reactive human intervention, when a forking incident happens, stakeholders get together to decide on a fork as the canonical chain. Since both types of interventions require stakeholder meetings, they are part of "social consensus".

[6]Babylon may store the data committed in the recent Babylon blocks for the synchronization between Babylon nodes.

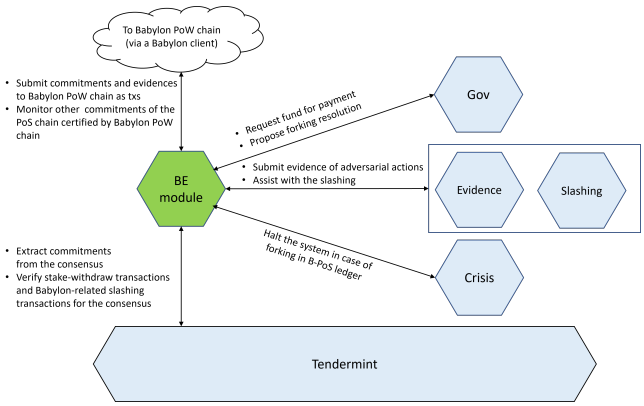

Figure 8: Enhancing Cosmos zones via a new BE (Babylon-enhancement) module.

7.2 Enhancing Cosmos Security with Babylon

To enhance the security of Cosmos PoS chains with Babylon, we add a new module called BE (Babylon-enhancement) to the Cosmos SDK. This module executes the protocol described in Section 5 and only requires straightforward interactions with existing Cosmos modules. Some of the key interactions are as follows (Figure 8):

BE implements the Babylon-specific add-ons such as the fork-choice rule specified in Section 5.3 to output the canonical PoS chain. It monitors the PoS chain and creates the messages specified in Section 5 such as checkpoints, fraud proofs, censorship complaints and stalling evidences. It communicates with the gov module to obtain approval for the expenditure of community funds to pay for the Babylon transaction fees. It submits the messages mentioned above, through a customized client (Figure 10) to the Babylon chain and uses Babylon transactions to pay the miners. It also monitors the existing messages created for the same PoS chain and timestamped on Babylon. In case any adversarial action is detected through the interpretation of the messages on Babylon, it submits the evidences to the evidence module and then works with the slashing module to slash the adversarial validators on the PoS chain. In the case of forking on the PoS chain, it interacts with the crisis module to temporarily suspend the system, and proposes resolution via the gov module to recover the system, where the resolution is derived using the fork-choice rule specified in Section 5.3. When withdrawal requests and Babylon-related PoS transactions are submitted to the Tendermint consensus engine, it helps the engine verify such transactions.

All the above interactions can be supported by existing Cosmos modules via API and data format configurations. These configurations are explained below:

- Tendermint consensus engine: redirect the validation of stake withdrawal transactions and Babylon-related slashing transactions to the BE module.
- evidence module: add evidence types such as fraud proofs, censorship complaint and stalling evidence, corresponding to Babylon-related violations;
- slashing module: define the appropriate slashing rules as described in Sections 5.4, 5.5.2 and 5.5.4;
- crisis module: add handling of safety violations reported by the BE module;
- gov module: add two proposal types (i) to use community funds to pay for Babylon transaction fees and (ii) to execute fork choice decision made by the BE module.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] Ethereum Wiki: On sharding blockchains FAQs. https://eth.wiki/sharding/Sharding-FAQs. Accessed: 2022-01-12.
[2] Merged mining specification. https://en.bitcoin.it/wiki/Merged_mining_specification. Accessed: 2021-11-3.
[3] Namecoin. https://www.namecoin.org/. Accessed: 2021-11-3.
[4] Pow 51% attack cost. https://www.crypto51.app/. Accessed: 2021-11-3.
[5] Rsk. https://www.rsk.co/. Accessed: 2021-11-3.
[6] Mustafa Al-Bassam. Lazyledger: A distributed data availability ledger with client-side smart contracts. 2019.
[7] Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, and Ismail Khoffi. Fraud and data availability proofs: Detecting invalid blocks in light clients. In *Financial Cryptography and Data Security*, FC '21, 2021.
[8] VDF Alliance. VDF Alliance FPGA Competition. https://supranational.atlassian.net/wiki/spaces/VA/pages/36569208/FPGA+Competition, 2019.
[9] Georgia Avarikioti, Lukas Käppeli, Yuyi Wang, and Roger Wattenhofer. Bitcoin security under temporary dishonest majority. In *Financial Cryptography and Data Security*, FC '19, 2019.
[10] 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.
[11] Sarah Azouvi, George Danezis, and Valeria Nikolaenko. Winkle: Foiling long-range attacks in proof-of-stake systems. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, AFT '20, page 189–201, 2020.
[12] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros Genesis: Composable proof-of-stake blockchains with dynamic availability. In *Conference on Computer and Communications Security*, CCS '18, pages 913–930, 2018.
[13] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better — how to make bitcoin a better currency. In Angelos D. Keromytis, editor, *Financial Cryptography and Data Security*, pages 399–414, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
[14] 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.
[15] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains, 2016.
[16] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus, 2018.
[17] Vitalik Buterin. Proof of stake: How i learned to love weak subjectivity, 2014. Accessed: 2021-04-20.
[18] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv:1710.09437*, 2019.
[19] 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.
[20] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186. USENIX Association, 1999.
[21] Benjamin Y. Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains. In *Advances in Financial Technologies*, AFT '20, page 1–11. ACM, 2020.
[22] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.
[23] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security*, FC '19, 2019.
[24] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725, 2019.
[25] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT 2015*, pages 281–310, 2015.
[26] Sreeram Kannan, Kartik Nayak, Peiyao Sheng, Pramod Viswanath, and Gerui Wang. BFT protocol forensics. In *Conference on Computer and Communications Security*, CCS '21, 2021.
[27] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO 2017*, pages 357–388, 2017.
[28] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *Symposium on Security and Privacy*, S&P '21. IEEE, 2021.
[29] 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.
[30] List of modules in Cosmos SDK. https://docs.cosmos.network/master/modules/. Accessed: Nov 2021.
[31] Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. Blockchain cap theorem allows user-dependent adaptivity and finality. In *Financial Cryptography and Data Security*, FC '21, 2021.
[32] Selma Steinhoff, Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolić. BMS: Secure Decentralized Reconfiguration for Blockchain and BFT Systems, 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 *Symposium on Principles of Distributed Computing*, PODC '19, page 347–356, 2019.
[36] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. Coded Merkle tree: Solving data availability attacks in blockchains. In *Financial Cryptography and Data Security*, FC '18, 2018.

A PROOFS FOR SECTION 4

To formalize slashable safety and its absence thereof, we define *slashable safety resilience* for the PoS protocols and state the impossibility theorem for slashable safety in the absence of additional trust assumptions:

Definition 6. *Slashable safety resilience of a protocol is the minimum number f of validators that become slashable in the view of all honest PoS nodes per Definition 3 in the event of a safety violation. Such a protocol provides f-slashable-safety.*

Theorem 3. *Assuming a 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 $f_s, f_l > 0$ and $T_{fin} < \infty$.*

Proof. For the sake of contradiction, suppose there exists a PoS protocol Π that provides f_l-T_{fin}-liveness and f_s-slashable-safety for some $f_l, f_s > 0$ and $T_{fin} < \infty$ without any additional trust assumptions.

Let n be the number of active validators at any given time. Let P, Q' and Q'' denote disjoint sets of validators such that $P := \{v_i, i = 1, .., n\}$, $Q' := \{v'_i, i = 1, .., n\}$ and $Q'' := \{v''_i, i = 1, .., n\}$.

Next, we consider the following two worlds, where the adversarial behavior is designated by $(\mathcal{A}, \mathcal{Z})$:

World 1: $(\mathcal{A}, \mathcal{Z})$ provides P as the initial set of active validators. Validators in Q' are honest. Validators in P and Q'' are adversarial.

At time $t = 0$, $\mathcal{Z}$ inputs transactions tx'_i, $i = 1, .., n$, to the validators in P, where tx'_i causes $\mathsf{v}_i \in P$ to become passive and $\mathsf{v}'_i \in Q'$ to become active. Validators in P emulate a set of honest validators with equal size, except that they record every piece of information in their transcripts. Since $f_{\mathsf{l}} > 0$ and $T_{\mathsf{fin}} < \infty$, there exists a constant time T such that upon receiving transcripts from the set of active validators at time T, clients output a ledger Ledger for which $\mathsf{tx}'_i \in \mathsf{Ledger}$, $i = 1, .., n$. Thus, the set of active validators at time T is Q' in the view of any client. As passive validators withdraw their stake within a constant time T', by time $T + T'$, all validators in P have withdrawn their stake.

In parallel to the real execution above, $(\mathcal{A}, \mathcal{Z})$ creates a simulated execution in its head where a different set of transactions, tx''_i, $i = 1, .., n$, is input to the validators in P at time $t = 0$. Here, tx''_i causes $\mathsf{v}_i \in P$ to become passive and $\mathsf{v}''_i \in Q''$ to become active. Then, upon receiving the transcripts of the simulated execution at time T, clients would output a ledger Ledger' for which $\mathsf{tx}''_i \in \mathsf{Ledger}'$, $i = 1, .., n$. Then, the set of active validators at time T would be Q'' in the view of any client. As passive validators can withdraw their stake within a constant time T', all validators in P withdraw their stake in the simulated execution by time $T + T'$.

Finally, $(\mathcal{A}, \mathcal{Z})$ spawns a PoS client c at time $T + T'$, which receives transcripts from both the simulated and real executions. Since Ledger and Ledger' conflict with each other and $f_{\mathsf{s}} > 0$, there is a safety violation, and c identifies a set of irrefutably adversarial validators by invoking the forensic protocol, a non-empty subset of which is slashable. As the validators in P have withdrawn their stake and those in Q' are honest and did not violate the protocol, this set includes at least one slashable validator from Q''.

World 2: World 2 is the same as World 1, except that (i) validators in Q' are adversarial and those in Q'' are honest, and (ii) the transactions tx'_i and tx''_i, $i = 1, .., n$, are swapped in the description, *i.e.* tx'_i is replaced by tx''_i and vice versa.

***

Finally, as World 1 and 2 are indistinguishable, c again identifies a validator from Q'' as slashable in World 2 with probability at least $1/2$. However, the validators in Q'' are honest in World 2, and could not have been identified as irrefutably adversarial, *i.e.* contradiction. □

Following theorem is used for the proof of Theorem 2.

Theorem 4. *For any SMR protocol that is run by n validators and satisfies f_{s}-safety and f_{l}-T_{fin}-liveness with $f_{\mathsf{s}}, f_{\mathsf{l}} > 0$ (assuming Byzantine faults) and $T_{\mathsf{fin}} < \infty$, it must be the case that $f_{\mathsf{s}} < n - f_{\mathsf{l}}$.*

Proof. For the sake of contradiction, assume that there exists an SMR protocol Π that provides f_{l}-T_{fin}-liveness for some $f_{\mathsf{l}} > 0$, $T_{\mathsf{fin}} < \infty$ and f_{s}-safety for $f_{\mathsf{s}} = n - f_{\mathsf{l}}$. Then, the protocol should be safe when there are $n - f_{\mathsf{l}}$ adversarial validators. Let P, Q and R denote disjoint sets consisting of f_{l}, f_{l} and $n - 2f_{\mathsf{l}} > 0$ validators respectively, where we assume $f_{\mathsf{l}} < n/2$. Next, consider the following worlds with two clients c_1 and c_2 prone to omission faults, where the adversarial behavior is designated by $(\mathcal{A}, \mathcal{Z})$:

World 1: $\mathcal{Z}$ inputs tx_1 to all validators. Those in P and R are honest and the validators in Q are adversarial. There is only one client c_1. Validators in Q do not communicate with those in P and R; they also do not respond to c_1. Since $P \cup R$ has size $n - f_{\mathsf{l}}$ and consists of honest validators, via f_{l}-liveness, upon receiving transcripts from the validators in P and R, c_1 outputs the ledger $[\mathsf{tx}_1]$ by time T_{fin}.

World 2: $\mathcal{Z}$ inputs tx_2 to all validators. Those in Q and R are honest and the validators in P are adversarial. There is only one client c_2. Validators in P do not communicate with those in Q and R; they also do not respond to c_2. Since $Q \cup R$ has size $n - f_{\mathsf{l}}$ and consists of honest validators, via f_{l}-liveness, upon receiving transcripts from the validators in Q and R, c_2 outputs the ledger $[\mathsf{tx}_2]$ by time T_{fin}.

World 3: $\mathcal{Z}$ inputs tx_1 to the validators in P, tx_2 to the validators in Q, and both transactions to the validators in R. Validators in P are honest, those in Q and R are adversarial. There are two clients this time, c_1 and c_2. Validators in Q do not send any message to any of the validators in P; they also do not respond to c_1. $\mathcal{Z}$ also omits any message sent from the validators in P to c_2.

Validators in R perform a split-brain attack where one brain interacts with P as if the input were tx_1 and it is not receiving any message from Q (real execution). Simultaneously, validators in Q and the other brain of R start with input tx_2 and communicate with each other exactly as in world 2, creating a simulated execution. The first brain of R only responds to c_1 and the second brain of R only responds to c_2.

Since worlds 1 and 3 are indistinguishable for c_1 and the honest validators in P, upon receiving transcripts from the validators in P and the first brain of R, c_1 outputs $[\mathsf{tx}_1]$ by time T_{fin}. Similarly, since worlds 2 and 3 are indistinguishable for c_2, upon receiving transcripts from the validators in Q and the second brain of R, c_2 outputs $[\mathsf{tx}_2]$ by time T_{fin}.

Finally, there is a safety violation in world 3 since c_1 and c_2 output conflicting ledgers. However, there are only $f_{\mathsf{s}} = n - f_{\mathsf{l}}$ adversarial validators in Q and R, which is a contradiction.

Proof for $f_{\mathsf{l}} \geq n/2$ proceeds via a similar argument, where sets P, Q and R denote disjoint sets of sizes $n - f_{\mathsf{l}}$, $n - f_{\mathsf{l}}$ and $2f_{\mathsf{l}} - n > 0$ respectively. □

Proof of Theorem 2 is given below:

Proof of Theorem 2. For the sake of contradiction, suppose there exists a PoS protocol Π with a static set of validators that provides f_{a}-T_{fin}-accountable-liveness and f_{s}-safety for some $f_{\mathsf{a}}, f_{\mathsf{s}} > 0$ and $T_{\mathsf{fin}} < \infty$ without any additional trust assumptions. Then, there exists a forensic protocol which takes transcripts sent by the validators as input, and in the event of a liveness violation, outputs a non-empty set of validators which have irrefutably violated the protocol rules.

Let f_{l} denote the liveness resilience of Π. By Theorem 4, $f_{\mathsf{l}} < n - f_{\mathsf{s}}$, *i.e.*, $f_{\mathsf{l}} \leq n - 2$ as $f_{\mathsf{s}} > 0$. By definition of accountable liveness resilience, $f_{\mathsf{l}} \geq f_{\mathsf{a}} > 0$. Let $m \geq 1$ denote the maximum integer less than $n - f_{\mathsf{l}} \geq 2$ that divides n. Let P_i, $i = 1, .., n/m$ (*i.e.* $i \in [n/m]$) denote sets of size m that partition the n validators into n/m disjoint, equally sized groups. We next consider the following worlds indexed by $i \in [n/m]$ where $\mathcal{Z}$ inputs a transaction tx to all validators at time $t = 0$ and the adversarial behavior is designated by $(\mathcal{A}, \mathcal{Z})$:

World *i*: Validators in P_i are honest. Validators in each set P_j, $j \neq i, j \in [n/m]$, are adversarial and simulate the execution of m honest validators in their heads without any communication with

the validators in the other sets. Validators in each P_j, $j \in [n/m]$ generate a set of transcripts such that upon receiving transcripts from the set of validators in P_j at time T_{fin}, a client outputs a (potentially empty) ledger Ledger_j, $j \in [n/m]$. As $|P_j| < n - f_{\text{l}}$, validators in P_i do not hear from the validators in P_j, $j \neq i$, and the validators in P_j, $j \neq i$ simulate the execution of the honest validators in world j respectively, $\mathsf{tx} \notin \mathsf{Ledger}_j$ for any $j \in [n/m]$.

Finally, $(\mathcal{A}, \mathcal{Z})$ spawns a client at time T_{fin}, which receives transcripts from both the real and the multiple simulated executions. Since $\mathsf{tx} \notin \mathsf{Ledger}_j$ for any $j \in [n/m]$, there is a liveness violation in the client's view. As $f_{\text{a}} > 0$, by invoking the forensic protocol with the transcripts received, client identifies a subset S_i of validators as irrefutably adversarial.

***

Finally, by definition of S_i, it should be the case that $S_i \subseteq \bigcup_{j \in [n/m], j \neq i} P_j$. However, as worlds i, $i \in [n/m]$ are indistinguishable for the client, there exists a world i^*, $i^* \in [n/m]$, such that a node from P_{i^*} is identified as adversarial in world i^* with probability at least $m/n \geq 1/n$, which is non-negligible. This is a contradiction. □

Note that when the number of validators n is large and $f_{\text{l}} = n - 2$, probability that the forensic protocol for accountable liveness makes a mistake and identifies an honest validator as adversarial can be small. However, assuming that n is polynomial in the security parameter of the PoS protocol, this probability will not be negligible in the security parameter.

B MERGE MINING AND CLIENT APPLICATIONS

Miners merge-mine the Babylon chain following the longest chain rule. To merge-mine Babylon blocks, miners calculate hashes of blocks containing both Bitcoin and Babylon transactions. Whenever a miner finds a block with its hash falling into the *Babylon range*, it shares the Babylon transactions in this block with its *Babylon client*, which extracts a Babylon block from the received contents (*cf.* Figure 9). Hash of this block is then sent over Bitcoin's peer-to-peer network to be included as a Bitcoin transaction [2]. Babylon blocks have the same structure as Bitcoin blocks. Size of the Babylon range determines the chain difficulty, in turn, the growth rate λ for the Babylon chain.

Babylon client is run by the Bitcoin miners in parallel with the Bitcoin client. Besides exchanging nonces and hashes with the mining software for merge-mining, Babylon client also records the commitments submitted by the PoS chains and checks for data availability. Thus, miners follow the same longest chain mining protocol as regular Bitcoin clients, except for the fact that they also check for the availability of the PoS blocks before accepting their commitments.

Similar to miners, each PoS node using Babylon runs a special *Babylon-embedded* PoS chain client (Figure 10). This client is built on top of an existing PoS client, but augmented with Babylon-specific add-ons to allow the PoS node to post commitments and checkpoints to Babylon as well as interpret the timestamps of these messages.

C STALLING RESILIENCE

C.1 Tendermint Summary

Tendermint consensus proceeds in heights and rounds. Each height represents a new consensus instance and the validators cannot move on to the next height before a unique block is finalized for the previous one. Heights consist of rounds, each with a unique leader that proposes a PoS block. Goal of each round is to finalize a block for its height.

Rounds are divided into three steps: propose, prevote and precommit. An honest round leader proposes a block for its round at the beginning of the propose step. Then, during the respective steps, validators send prevote and precommit messages for the proposed block or a *nil* block, depending on the proposal and their internal states. Each honest validator maintains four variables which affect its decision whether to prevote for a proposal: lockedValue, lockedRound, validValue and validRound. lockedValue denotes the most recent non-nil block for which the validator sent a precommit message. validValue denotes the most recent non-nil block for which the validator has observed $2f + 1$ prevotes. Recency of a block is determined by the round it was proposed for by the leader of that round. Thus, lockedRound and validRound refer to the rounds for which lockedValue and validValue were proposed respectively. At the beginning of each height, lockedValue, lockedRound, validValue and validRound are reset to $\perp$, -1, $\perp$ and -1 respectively.

C.1.1 Propose. If the leader of a round r, height h, is honest, it broadcasts the following proposal message at the beginning of the round if its validRound ≥ 0: $\langle \text{PROPOSAL}, h, r, v = \text{validValue}, vr = \text{validRound} \rangle$. Otherwise, it proposes a new valid PoS block B: $\langle \text{PROPOSAL}, h, r, v = B, vr = -1 \rangle$. Similarly, upon receiving a proposal message $\langle \text{PROPOSAL}, h, r, v, vr \rangle$ (from the round leader) during the propose step of round r and height h, an honest validator broadcasts the following prevote message $\langle \text{PREVOTE}, h, r, id(v) \rangle$ for the proposal if either (i) v is the same block as its lockedValue, or (ii) vr is larger than its lockedRound. Otherwise, it sends a prevote for a nil block: $\langle \text{PREVOTE}, h, r, nil \rangle$. Thus, by proposing its validValue instead of a new block when $vr \neq -1$, an honest leader ensures that honest validators locked on blocks from previous rounds will be prevoting for its proposal instead of nil blocks.

If the honest validator does not observe any proposal message within a timeout period of its entry to the propose step, it sends a prevote for a nil block. After sending its prevote, it leaves the propose step and enters the prevote step.

C.1.2 Prevote. Once in the prevote step, the honest validator waits until it receives $2f + 1$ prevotes, for potentially different blocks, upon which it activates a prevote countdown. If it observes $2f + 1$ prevotes for a valid block B proposed for round r and height h during this time, it sends the following precommit message $\langle \text{PRECOMMIT}, h, r, id(B) \rangle$ and enters the precommit step. It also updates its lockedValue, lockedRound, validValue and validRound to B, r, B and r respectively. If the honest validator receives $2f + 1$ prevotes for nil blocks, it sends a precommit message for a nil block: $\langle \text{PRECOMMIT}, h, r, nil \rangle$.

If the honest validator does not receive $2f + 1$ prevotes for a valid block B before the countdown expires, it sends a precommit for a

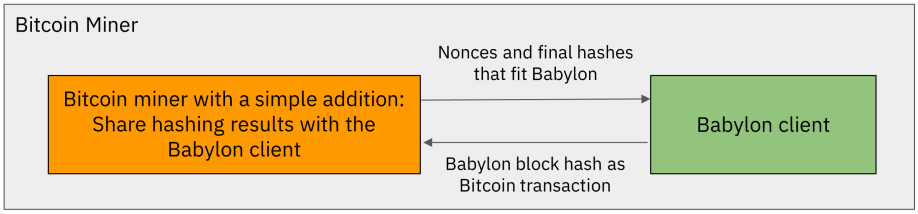

Figure 9: Interaction of the Babylon client run by Babylon miners with the mining software in the context of merge-mining. Babylon client uses the same hashing results generated by the Bitcoin miners as a Bitcoin client, but the criterion of mining a new Babylon block based on those results is different from that of mining a Bitcoin block.

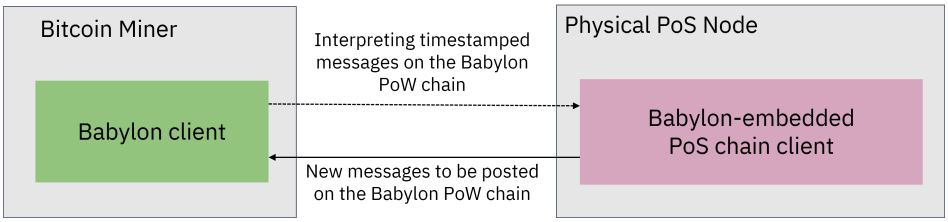

Figure 10: Interaction between the Babylon-embedded PoS chain client and the Babylon client in the context of timestamping PoS block commitments.

nil block. After sending its precommit, it leaves the prevote step and enters the precommit step.

C.1.3 Precommit. Finally, during the precommit step, our honest validator waits until it receives $2f + 1$ precommit messages, for potentially different blocks, upon which it activates a precommit countdown. If it observes $2f + 1$ precommit messages for a valid block B proposed for round r and height h, it finalizes B for height h and moves on to the next height $h + 1$. Otherwise, if the countdown expires or there are $2f + 1$ precommit messages for nil blocks, validator enters the next round $r + 1$ without finalizing any block for height h.

Timeout periods for proposal, prevote and precommit steps are adjusted to ensure the liveness of Tendermint under Δ synchrony when there are at least $2f + 1$ honest validators. On the other hand, the two step voting process along with the locking mechanism guarantees its safety by preventing conflicting blocks from receiving more than $2f + 1$ prevotes for the same round and more than $2f + 1$ precommits for the same height.

C.2 Details of Stalling Resilience through Babylon

This section presents a detailed description of how a Tendermint round is recorded on Babylon and interpreted by the nodes when the PoS chain is stalled. For the rest of this section, we assume that the Babylon chain in the view of a node refers to the $k_c/2$-deep prefix of the longest Babylon chain in its view.

To clarify the connection between censorship and stalling, we extend the definition of censoring blocks presented in Section 5.5.1 to proposals recorded on Babylon: Consider an honest PoS node p and let b be a Babylon block containing a valid censorship complaint in p's Babylon chain, *i.e.*, the longest Babylon chain in p's view. Define b' as the first block on p's Babylon chain that contains a checkpoint and extends b by at least $2k_c$ blocks. Then, a proposal message (*cf.* Appendix C.1.1) for a block B is said to be *censoring* in p's view if (i) the proposal was sent in response to a stalling evidence within a block b'' such that $b' \prec b''$ and comes after the checkpoint in b' in p's Babylon chain, and (ii) B does not include the censored transactions neither in its body nor within its prefix. The $2k_c$ lower bound on the gap between b and b' ensures that all finalized PoS blocks which exclude the censored transactions and were proposed or voted upon by honest validator are checkpointed by b' or other Babylon blocks in its prefix, thus leaving no room to accuse an honest validator for censorship.

Next, we describe a Tendermint round recorded on-Babylon in the perspective of an honest validator p. Suppose there is a stalling evidence for some height h on p's Babylon chain and the evidence is at least $2k_c$ blocks apart from the last preceding checkpoint. Then, upon observing the first such stalling evidence recorded by a Babylon block b, p enters a new Tendermint round for height h, whose messages are recorded on-Babylon, and freezes the parameters lockedValue, lockedRound, validValue and validRound in its view. If p has observed a new valid Tendermint block become finalized at the height h by that time, it sends a new checkpoint to Babylon for that block. Otherwise, p signs and sends a proposal message to Babylon, pretending as the leader of the new round. Since the round is recorded on Babylon, its number round_p is set to a special value, Babylon. Thus, p's PROPOSAL message is structured as $\langle \text{PROPOSAL}, h, \text{Babylon}, H(v), vr \rangle$, where either (i) $(v, vr) = (\text{validValue}, \text{validRound})$ held by p if p's validValue ≥ 0, or (ii) $(v, vr) = (B, -1)$, where B is a new PoS block created by p, if p's validValue $= -1$ (*cf.* Appendix C.1.1). If $vr \geq 0$ and p proposed

its validValue as the proposal v, it also sends a commitment of the $2f+1$ prevote messages for v to Babylon along with the proposal. This is to convince late-coming PoS nodes that the $2f+1$ prevote messages for v were indeed seen by p before it proposed v.

Note that p only includes the hash of the proposed block v in the proposal message unlike the proposals in Tendermint (*cf.* Appendix C.1.1). To ensure that other PoS nodes can download v if needed, miners check its availability before accepting p's proposal message as valid. Similarly, miners check the availability of prevotes upon receiving a proposal that proposes a validValue held by the validator.

Let b_1 and b_2 denote the first blocks on p's Babylon chain that extend b by k_c and $2k_c$ blocks respectively. If p (or any honest PoS node) observes a checkpoint for a PoS block finalized at height h between b and b_1, it stops participating in the round recorded on-Babylon and moves to the next height, resuming its communication with the other validators through the network. Otherwise, if there are $\geq 2f+1$ non-censoring proposal messages signed by unique validators between b and b_1, it selects the non-censoring valid block proposed with the largest vr as the proposal of the 'Babylon round' emulated on-Babylon. If there are multiple proposals with the highest vr, p selects the one that appears earliest between b and b_1 on Babylon. Selecting the proposal with the largest vr ensures that the honest validators can later prevote and precommit for that block without violating Tendermint rules (*cf.* Appendix C.1.1).

Once p decides on a proposal B and observes b_1 in its Babylon chain, if it is locked on PoS block, it checks if B is the same block as its lockedValue or if the proposal's vr is larger than its lockedRound (*cf.* Appendix C.1.1). If so, it sends the following prevote and precommit messages for the selected proposal to Babylon: $\langle$PREVOTE, h, Babylon, $id(B)\rangle$ and $\langle$PRECOMMIT, h, Babylon, $id(B)\rangle$ (*cf.* Sections C.1.2 and C.1.3). If p is not locked on any PoS block, it directly sends the prevote and precommit messages. Unlike in Tendermint, p does not wait to observe $2f+1$ prevote messages for B before it sends its precommit message. This is because the purpose of the round emulated on Babylon is to catch unresponsive validators stalling the protocol, thus, does not need the two step voting. However, it still keeps the two step voting for the purpose of consistency with the Tendermint rounds that happened off-Babylon.

Finally, upon observing b_2 in its Babylon chain, p finalizes B if there are more than $2f+1$ prevotes and precommits for B, signed by unique validators, between b_1 and b_2. In this case, b_2 is designated as the Babylon block that has checkpointed the finalized block for height h. Upon finalizing B, p moves to the next height, resuming its communication with the other validators through the network.

If p observes no new checkpoints and less than $2f+1$ uniquely signed non-censoring proposals between b and b_1, it does not send a prevote or precommit, and instead attempts to restart the round on-Babylon by sending a new stalling evidence. Similarly, if p observes less than $2f+1$ uniquely signed prevotes or precommits for the selected proposal B between b_1 and b_2, it does not finalize B, again restarting the round on-Babylon. Finally, if p observes a fraud proof on Babylon that implies a safety violation on the PoS chains, it stops participating in the Tendermint round on-Babylon and temporarily halts finalizing new PoS blocks.

D PROOF OF THEOREM 1

We prove Theorem 1 below by showing properties **S1**-**S2** and **L1**-**L2** for the PoS chains.

D.1 Proof of the Safety Claims S1 & S2

Proposition 1. *If a transaction* tx *is sent to the miners at time* $t+\Delta$ *such that* $|\mathsf{PoWChain}_{\mathsf{c}}^{t}| \leq L$ *for all nodes* c, $\mathsf{tx} \in \mathsf{PoWChain}_{\mathsf{c}'}^{t'}$ *for any honest node* c′, *where* $|\mathsf{PoWChain}^{t'}| = L + k_w/2$.

Proof of Proposition 1 follows from the $k_w/2$-security of Babylon.

Proposition 2. *Consider a PoS block* $B \in \mathsf{PoSLOG}_i^t$, *checkpointed by a Babylon block* $b \in \mathsf{PoWChain}_i^t$. *Then, there cannot be any Babylon block in the prefix of* b *that checkpoints a PoS block conflicting with* B. *If* $B \in \mathsf{PoSLOG}_i^t$ *and is not checkpointed in* i*'s view by time* t, *then there cannot be any Babylon block* $b' \in \mathsf{PoWChain}_i^t$ *that checkpoints a PoS block conflicting with* B.

Proof. For the sake of contradiction, suppose there exists a Babylon block $b' \preceq b$ such that b' checkpoints a PoS block B' that conflicts with B. Then, via the fork-choice rule in Section 5.3, $B \notin \mathsf{PoSLOG}_i^t$, *i.e.* contradiction. Similarly, if B is not checkpointed in i's view by time t and there exists a Babylon block $b' \in \mathsf{PoWChain}_i^t$ such that b' checkpoints a PoS block B' that conflicts with B, again via the fork-choice rule in Section 5.3, $B \notin \mathsf{PoSLOG}_i^t$, *i.e.* contradiction. □

To show the safety claims **S1** and **S2**, we prove that if Babylon is secure with parameter $k_w/2$, then whenever there is a safety violation on the PoS chains, at least $1/3$ of the validator set becomes slashable in the view of all honest PoS nodes.

Proof. Suppose there is a safety violation on the PoS chains and PoSLOG_i^t observed by an honest node i at time t conflicts with $\mathsf{PoSLOG}_j^{t'}$ observed by an honest node j at time $t' \geq t$. Let B_1 and B_2 denote the first two conflicting PoS blocks on PoSLOG_i^t and $\mathsf{PoSLOG}_j^{t'}$ respectively. Via synchrony, by time $t'+\Delta$, every honest node observes B_1 and B_2, their prefixes and the protocol messages attesting to their PoS-finalization. Since the PoS protocol has an accountable safety resilience of $1/3$, upon inspecting the blocks, their prefixes and the associated messages, any node can irrefutably identify $1/3$ of the validator set for B_1 and B_2 as having violated the protocol, and submit a fraud proof to Babylon by time $t'+\Delta$. Let S denote the set of the adversarial validators witnessed by the fraud proof.

For the sake of contradiction, assume that there is a validator $\mathsf{v} \in S$ that has not become slashable in the view of an honest node c. Then, there exists a time t_0 and a PoS block B_2' containing v's withdrawal request such that B_2' is checkpointed by a Babylon block b_2' that is at least k_w-deep in $\mathsf{PoWChain}_{\mathsf{c}}^{t_0}$ and there is no fraud proof showing v's misbehavior on $\mathsf{PoWChain}_{\mathsf{c}}^{t_0}$ (*cf.* Section 5.4). Now, suppose b_2' has not become at least $k_w/2$ deep in the longest Babylon chain in the view of any node, including adversarial ones, by time $t'+\Delta$. In this case, since the fraud proof submitted to the Babylon chain by time $t'+\Delta$ will appear and stay in the canonical Babylon chain of all honest nodes within $k_w/2$ block-time of t' by Proposition 1, fraud proof will be on $\mathsf{PoWChain}_{\mathsf{c}}^{t_0}$ as well. However,

this is a contradiction, implying that there must be at least one, potentially adversarial, node j', which observes b_2' become $k_w/2$ deep in its longest Babylon chain at some time $s \leq t' + \Delta$.

Next, we analyze the following cases:

- **Case 1:** There exists a Babylon block $b_1 \in \text{PoWChain}_i^t$ such that b_1 checkpoints B_1 and $b_1 \preceq b_2' \in \text{PoWChain}_i^t$.
- **Case 2:** There exists a Babylon block $b_1 \in \text{PoWChain}_i^t$ such that b_1 checkpoints B_1 and $b_2' \prec b_1$.
- **Case 3:** $b_2' \notin \text{PoWChain}_i^t$.
- **Case 4:** There does not exist a Babylon block $b_1 \in \text{PoWChain}_i^t$ checkpointing B_1 at time t and $b_2' \in \text{PoWChain}_i^t$.

Case 1: $b_1 \preceq b_2'$. By Proposition 2, $b_1 \notin \text{PoWChain}_{j'}^{t'}$, which implies that the $k_w/2$ blocks building on b_2' in $\text{PoWChain}_{j'}^{s}$ are not in $\text{PoWChain}_j^{t'}$ at time $t' \geq s - \Delta$. However, this is a contradiction with the $k_w/2$-safety of Babylon.

Case 2: $b_2' \prec b_1$. If B_2' conflicts with B_1, as $B_1 \in \text{PoSLOG}_i^t$, via Proposition 2, $b_2' \prec b_1$ cannot be true, *i.e.* contradiction. On the other hand, if $b_2' \prec b_1$ and B_2' does not conflict with B_1, then $B_2' \prec B_1$, in which case v cannot be in the validator set S that voted for B_1, *i.e.* contradiction.

Case 3: $b_2' \notin \text{PoWChain}_i^t$. Suppose $t \geq s$. Since PoWChain_i^t does not contain the $k_w/2$ Babylon blocks following b_2' in j''s canonical Babylon chain at time s, in this case, Babylon cannot be safe with parameter $k_w/2$, *i.e.*, contradiction.

On the other hand, if $t < s$, we consider the following sub-cases:

- **Case 3-a:** There exists a Babylon block $b_2 \in \text{PoWChain}_j^{t'}$ such that b_2 checkpoints B_2 and $b_2 \preceq b_2'$.
- **Case 3-b:** There exists a Babylon block $b_2 \in \text{PoWChain}_j^{t'}$ such that b_2 checkpoints B_2 and $b_2' \prec b_2$.
- **Case 3-c:** $b_2' \notin \text{PoWChain}_j^{t'}$.
- **Case 3-d:** There does not exist a Babylon block $b_2 \in \text{PoWChain}_j^{t'}$ checkpointing B_2 at time t' and $b_2' \in \text{PoWChain}_j^{t'}$.

Case 3-a: $b_2 \preceq b_2'$. In this case, as $t < s$, by time $s + \Delta$, node i would have observed both PoS blocks B_1 and B_2 along with their prefixes and sent a fraud proof to the Babylon miners. Then, by Proposition 1, the fraud proof will appear in the prefix of the k_w-th Babylon block building on b_2' in c's canonical Babylon chain. However, this is a contradiction with the assumption that v has withdrawn its stake in c's view.

Case 3-b: $b_2' \prec b_2$. If B_2' conflicts with B_2, as $B_2 \in \text{PoSLOG}_j^{t'}$, via Proposition 2, $b_2' \prec b_2$ cannot be true, *i.e.* contradiction. On the other hand, if $b_2' \prec b_2$ and B_2' does not conflict with B_2, then $B_2' \prec B_2$, in which case v cannot be in the validator set S that voted for B_2, again a contradiction.

Case 3-c: $b_2' \notin \text{PoWChain}_j^{t'}$. In this case, $\text{PoWChain}_j^{t'}$ does not contain the $k_w/2$ Babylon blocks following b_2' in j''s canonical Babylon chain at time $s \leq t' + \Delta$. However, this contradicts with the $k_w/2$-safety of the Babylon chain.

Case 3-d: There does not exist a Babylon block $b_2 \in \text{PoWChain}_j^{t'}$ checkpointing B_2 at time t' and $b_2' \in \text{PoWChain}_j^{t'}$. In this case, if B_2' conflicts with B_2 and $B_2 \in \text{PoSLOG}_j^{t'}$, via Proposition 2, $b_2' \in \text{PoWChain}_j^{t'}$ cannot be true, *i.e.* contradiction. On the other hand, if $B_2' \prec B_2$, v cannot be in the validator set S that voted for B_2, again a contradiction. Finally, if $B_2 \preceq B_2'$, then b_2' also checkpoints B_2 by the monotonicity of checkpoints (*cf.* Section 5.3), which is a contradiction with the assumption that there does not exist a Babylon block $b_2 \in \text{PoWChain}_j^{t'}$ checkpointing B_2 at time t'.

Case 4: There does not exist a Babylon block $b_1 \in \text{PoWChain}_i^t$ checkpointing B_1 at time t and $b_2' \in \text{PoWChain}_i^t$. In this case, if B_2' conflicts with B_1, as $B_1 \in \text{PoSLOG}_i^t$, via Proposition 2, $b_2' \in \text{PoWChain}_i^t$ cannot be true, implying contradiction. On the other hand, if $B_2' \prec B_1$, v cannot be in the validator set S that voted for B_1, again a contradiction. Finally, if $B_1 \preceq B_2'$, then b_2' also checkpoints B_1 by the monotonicity of checkpoints (*cf.* Section 5.3), which is a contradiction with the assumption that there does not exist a Babylon block $b_1 \in \text{PoWChain}_i^t$ checkpointing B_1 at time t.

Thus, by contradiction, we have shown that if the Babylon chain satisfies $k_w/2$-security, none of the validators in the set S can withdraw their stake in the view of any honest node by time $t' + \Delta$. Moreover, since they have been irrefutably identified as protocol violators by every honest node by time $t' + \Delta$, they are slashable per Definition 3. Consequently, whenever there is a safety violation on the PoS chains, at least 1/3 of the validators become slashable if Babylon satisfies $k_w/2$-security. □

No honest validator becomes slashable in the view of any PoS node due to a safety violation since fraud proofs never identify an honest validator as a protocol violators via the accountability guarantee provided by Tendermint [15]. Thus, even if the adversary compromises the security of the Babylon chain, it cannot cause honest validators to get slashed for a safety violation on the PoS chain. However, when the security of the Babylon chain is violated, honest validators might be subjected to slashing for censorship and stalling as the arrow of time determined by Babylon is distorted. Since accountable liveness is impossible without external trust assumptions, slashing of honest nodes is unavoidable if the adversary can cause arbitrary reorganizations of blocks on Babylon. This point is addressed in the next section.

D.2 Proof of the Liveness Claims L1 & L2

In this section, we prove that if Babylon is secure with parameter $k_c/2 \leq k_w/2$, whenever there is a liveness violation exceeding $\Theta(k_c)$ block-time, at least 1/3 of the active validator set becomes slashable in the view of all honest nodes. Let f denote the safety and liveness resilience of Tendermint such that $n = 3f + 1$. In the rest of this section, we assume that there is no safety violation on the PoS chains and there is no fraud proof posted on Babylon that accuses $f + 1$ active validators of equivocating on prevote or precommit messages. We will relax this assumption and consider the interaction between safety and liveness violations at the end of this section.

In the rest of this section, we will assume that the Babylon chain is $k_c/2$-secure per Definition 2 unless stated otherwise. Moreover, in this section, Babylon chains in the view of honest nodes or validators will refer to the $k_c/2$-deep prefix of the longest chain in their view. Under the $k_c/2$-security assumption for Babylon, this reference ensures that (i) the Babylon chains observed by different honest nodes at any given time are prefixes of each other, and

(ii) the data behind every commitment appearing on the Babylon chains held by the honest nodes is available per Definition 2.

Let $\mathsf{PoWChain}^t$, without any node specified, denote the shortest Babylon chain in the view of the honest nodes at time t. Thus, $\mathsf{PoWChain}^t$ is a prefix of all Babylon chains held by the honest nodes at time t. Moreover, by the synchrony assumption, we deduce that all Babylon chains held by the honest nodes at time $t - \Delta$ are prefixes of $\mathsf{PoWChain}^t$. Using the notation $\mathsf{PoWChain}^t$, we can state the liveness property of the Babylon chain in the following way:

Proposition 3. *If a transaction* tx *is sent to the miners at time* $t+\Delta$ *such that* $|\mathsf{PoWChain}^t| = L, \mathsf{tx} \in \mathsf{PoWChain}^{t'}$, *where* $|\mathsf{PoWChain}^{t'}| = L + k_c$.

Proof of Proposition 3 follows from the $k_c/2$-security of Babylon.

As stated in Section 5.5, at least one honest node sends checkpoints for new finalized PoS blocks every time it observes the Babylon chain grow by k_c blocks. Moreover, if a PoS block is finalized for the first time in an honest node's view at time t, it is finalized in every honest node's view by time $t+\Delta$ via the synchrony assumption.

Proposition 4. *Suppose a PoS block is finalized in the view of an honest node at time* t *such that* $|\mathsf{PoWChain}^t| = L$. *Then, a checkpoint for the finalized block appears in* $\mathsf{PoWChain}^{t'}$, *where* $|\mathsf{PoWChain}^{t'}| = L + 2k_c$.

Proof of Proposition 4 follows from Proposition 3 and the assumption above.

Lemma 1. *Suppose* $|\mathsf{PoWChain}^t| = L$ *and* $|\mathsf{PoWChain}^{t'}| = L+5k_c$ *for times* t, t', *and the height of the last PoS block finalized in any honest node's view by time* t *is* $h-1$. *Then, either a new non-censoring Tendermint block for height* h *is checkpointed within the interval* $(t, t']$ *or* $f + 1$ *active validators must have violated the slashing rules for censorship and stalling in Sections 5.5.2, C.2 and 5.5.4 in the view of all honest nodes.*

Proof. Let t_i denote the first time such that $|\mathsf{PoWChain}^{t_i}| = L + ik_c$. If no new checkpoint for height h appears on the Babylon chain by time t_2, at least one honest node must have sent a stalling evidence to Babylon by that time. Thus, via Proposition 3, by time t_3, there exists a stalling evidence recorded in a block b in the Babylon chains held by all honest nodes. Similarly, by time t_4, block b_1 that extends b by k_c blocks appears in the Babylon chain of all honest nodes. At this point, there are three possibilities:

(1) There is a new checkpoint in the prefix of b_1 that commits to a new PoS block finalized for height h.
(2) There are less than $2f + 1$ proposal messages signed by unique validators, proposing non-censoring PoS blocks between b and b_1.
(3) There are $2f + 1$ or more proposal messages signed by unique validators, proposing non-censoring PoS blocks between b and b_1.

Case 1 implies that there is a new checkpoint in every honest node's view by time t' and case 2 implies that more than $f + 1$ active validators must have violated the slashing rules in Sections 5.5.2 and 5.5.4 in the view of all honest nodes.

If case 3 happens, validators are required to send prevote and precommit messages for the unique non-censoring PoS block B selected as described in Section C.2 by time t_4. Moreover, by time t_5, block b_2 that extends b_1 by k_c blocks appears in the Babylon chain of all honest nodes. Thus, if case 3 happens, there are two possibilities at time t_5:

(1) There are less than $2f + 1$ prevote or precommit messages for B, signed by unique validators, between b_1 and b_2.
(2) There are $2f + 1$ or more prevote and precommit messages for B, signed by unique validators, between b_1 and b_2.

In the former case, more than $f + 1$ active validators must have violated the slashing rules in Section 5.5.4 in the view of all honest nodes. In the latter case, block B is PoS-finalized and block b_2 acts as a new checkpoint for the PoS block at height h as stated in Section C.2. □

Lemma 2. *No honest validator violates the slashing rules for censorship and stalling in Sections 5.5.2, C.2 and 5.5.4 in the view of any honest node.*

Proof. **Censorship:** Suppose a Babylon block b containing a censorship complaint first appears in the Babylon chains of *all* honest nodes at time t_0, $|\mathsf{PoWChain}^{t_0}| = L$. Let t_i denote the first time such that $|\mathsf{PoWChain}^{t_i}| = L + ik_c$. For the sake of contradiction, suppose an honest validator proposed or voted (precommit or commit) for a censoring PoS block B. Let b' denote the Babylon block containing the first checkpoint after b that is *more than* $2k_c$ blocks apart from b. Given b and b', there are two cases for B:

- B is PoS-finalized and checkpointed by a Babylon block b'' such that $b' \prec b''$ in all of the Babylon chains held by the honest nodes.
- A proposal for B appears in response to a stalling evidence recorded by a Babylon block b'' such that $b' \prec b''$ (*cf.* Section C.2).

Let h denote the largest height for which a PoS block was finalized in the view of any honest node by time t_0. Then, by Proposition 4, a checkpoint for that block appears in the Babylon chains held by all honest nodes by time t_2 in the prefix of the $2k_c$-th block extending b. Thus, by the monotonicity of checkpoints, the checkpoint within block b' must cover the PoS block finalized for height $h + 1$. Moreover, since honest validators cannot propose or vote for PoS blocks at heights $\geq h + 2$ before a block for height $h + 1$ is finalized, if an honest validator proposed or voted for a PoS block excluding the censored transactions by time t_0, by definition, this block must have been proposed for height $h + 1$ or lower. Consequently, as honest validators would not propose or vote for new PoS blocks excluding the censored transactions after time t_0, such a block can only be finalized at heights $\leq h + 1$ and its checkpoint can only appear as a checkpoint either in block b' or within some other Babylon block in its prefix. Hence, given the first case above, it is not possible for an honest validator to have proposed or voted for B, *i.e.* contradiction.

In the second case, by Proposition 3, any stalling evidence recorded by a Babylon block b'' such that $b' \prec b''$, must have been sent after block b was observed by every honest node on their Babylon chains. From the explanation above, we know that a block that excludes the censored transactions and voted upon or proposed by honest

validators can only be finalized at heights $\leq h+1$ and its checkpoint can only appear as a checkpoint either in block b' or within some other Babylon block in its prefix. Thus, if a proposal for block B appears in response to a stalling evidence recorded by b' or one of its descendants, B must be for a height larger than $h+1$. Since no honest validator proposes or votes for PoS blocks from heights larger than $h+1$ that does not contain the censored transactions, it is not possible for an honest validator to have proposed B, *i.e.* contradiction. Consequently, no honest validator could have committed a slashable offense per the rules in Sections 5.5.2 and C.2, thus become slashable, in the view of any honest node.

Stalling: Consider a stalling evidence for height h recorded by a Babylon block b that is first observed by every honest node at time t_0 such that $|\mathsf{PoWChain}^{t_0}| = L$. Define b_1 and b_2 as the Babylon blocks that extend b by k_c and $2k_c$ blocks respectively in the Babylon chains of all honest nodes. For the sake of contradiction, suppose there is no checkpoint between b and b_1 for height h and a proposal message by an honest validator is missing in the same interval. In this case, the validator could not have seen a PoS block finalized for height h by time t_0 as it would have otherwise sent a checkpoint for it and the checkpoint would have been recorded between b and b_1 by Proposition 3. Hence, the validator must have sent a proposal message by time t_0, which would appear between b and b_1 by Proposition 3.

Next, assume that there are at least $2f+1$ proposal messages for non-censoring blocks between b and b_1, and no checkpoint for height h. For the sake of contradiction, suppose that prevote or precommit messages by an honest validator p are either missing between the blocks b_1 and b_2 or p sent these messages for a PoS block that is different from the proposal selected by another honest node p'. In either case, p becomes slashable for stalling in the view of p'. To rule out both cases, we first show that all honest validators choose the same block as their proposal, under our initial assumption that there is no fraud proof accusing $f+1$ active validators.

If all of the proposal messages between b and b_1 for non-censoring PoS blocks have $vr = -1$, then all honest validators choose the block proposed by the earliest message between b and b_1 as the proposal of the on-Babylon Tendermint round (*cf.* Section C.2). If there is only one proposal message between b and b_1 with $vr \neq -1$ for a non-censoring PoS block, then all honest validators choose this block as the proposal for the on-Babylon Tendermint round. Finally, suppose there are at least two proposal messages between b and b_1 with $vr \neq -1$ and for different non-censoring PoS blocks. Then, different PoS blocks must have acquired at least $2f+1$ prevote messages for the same round $vr \neq -1$ of the latest height h (*cf.* Section C.1). For different PoS blocks to acquire $2f+1$ prevote messages for the same round vr of height h, at least $f+1$ validators from the active validator set must have sent prevotes for conflicting blocks proposed for the same round vr. In this case, a fraud proof implicating these $f+1$ current validators will be created by an honest node and will eventually appear on the Babylon chain, which contradicts with our assumption on fraud proofs. Thus, under this assumption, either the maximum vr among all proposal messages is greater than -1, in which case the messages with the largest vr propose the same non-censoring PoS block, or all of them have $vr = -1$, in which case the non-censoring block within the earliest proposal on the Babylon chain between b and b_1 is selected by the nodes. Hence, in any of the cases, p could not have chosen, as its proposal for the on-Babylon round, a PoS block that is different from the proposal selected by another honest node p'.

Finally, if p's lockedRound $= -1$, then it directly sends prevote and precommit messages for the selected proposal upon observing block b_1 (*cf.* Section C.2). Otherwise, if its lockedRound $\neq -1$, its $-1 \neq$ validRound $\geq$ lockedRound, and, as an honest validator, p must have sent a proposal message with vr_{sent} equal to its validRound. Then, for the vr_{sel} of the selected proposal, which is the maximum among the vr values of all the proposals between b and b_1, thus $vr_{\text{sel}} \geq vr_{\text{sent}} =$ lockedRound $\geq$ lockedRound of p, there are two cases:

- vr_{sel} exceeds p's lockedRound, in which case p sends prevote and precommit messages for the proposal upon seeing block b_1.
- $vr_{\text{sel}} =$ lockedRound $=$ validRound $= vr_{\text{sent}} \neq -1$. In this case, p has sent a proposal message with the largest vr and must have proposed the same PoS block as the one suggested by the selected proposal message via the reasoning above, which is validValue by Tendermint rules (*cf.* Section C.1.1). As p's lockedRound $=$ validRound, it should be the case that p's lockedValue $=$ validValue unless again there are $f+1$ active validators that sent prevote messages for conflicting blocks at round lockedRound. Thus, p again sends prevote and precommit messages for the selected proposal upon seeing block b_1.

Thus, p sends prevote and precommit messages upon seeing b_1 for the selected proposal which is the same across all honest nodes, and these votes appear on Babylon in the view of all honest nodes between blocks b_1 and b_2 by Proposition 3. Consequently, p could not have committed a slashable offense per the rules in Section 5.5.4, and thus become slashable, in the view of any node p', *i.e.*, contradiction. □

Liveness part of Theorem 1 is proven below:

Proof. Let t_i denote the first time such that $|\mathsf{PoWChain}^{t_i}| = L + ik_c$. Suppose a censorship complaint is sent to Babylon at time t_0. Then, by Proposition 3, the complaint appears in the Babylon chain of every honest node within some block b by time t_1. Let $h-1$ be the height of the last PoS block finalized in any honest node's view by time t_3.

Suppose no more than f active validators violate the slashing rules in Sections 5.5.2, C.2 and C.2 in the view of all honest nodes. Then, via Lemma 1, a new non-censoring PoS block for height h is checkpointed by some Babylon block b' within the interval $(t_3, t_8]$. Similarly, again via Lemma 1, a new non-censoring PoS block B for a height larger than h is checkpointed by some Babylon block b'' by the time t_{13}. As B is non-censoring and appears within a Babylon block b'' such that $b' \prec b''$ (*cf.* Sections 5.5.1 and C.2), by definition of censoring blocks, it includes the censored transactions. Consequently, unless $f+1$ active validators violate the slashing rules for censorship and stalling in the view of all honest nodes, any valid PoS transaction becomes finalized and checkpointed within $13k_c$ blocktime of the time censorship is detected.

Finally, via Lemma 2, no honest validator violates the slashing rules for censorship and stalling in Sections 5.5.2, C.2 and 5.5.4 in the view of any honest node. Hence, if there is a liveness violation for a duration of more than $13k_c$ block-time, either of the following conditions must be true:

- **L1:** More than $f + 1 \geq n/3$ active validators, all of which are adversarial, must have violated the slashing rules for censorship and stalling in the view of all honest nodes. Thus, these protocol violators will be identified as irrefutably adversarial in the view of all honest nodes. As they are active validators and have not withdrawn their stake, they also become slashable in the view of all honest nodes.
- **L2:** The Babylon chain is not secure with parameter $k_c/2$.

This concludes the proof of the liveness claims in Theorem 1 under the assumption that no fraud proof appears on Babylon accusing $f + 1$ validators of a slashable offense for safety.

Finally, we relax the assumption on the fraud proof and safety. Suppose there is a fraud proof on Babylon implicating $f + 1$ active validators in a safety violation on the PoS chains. Then, the PoS protocol is temporarily halted and no validator can be slashed for any slashing rule other than for a safety violation (*cf.* Section 5.4). This precaution prevents adversarial validators from making honest ones slashable due to censorship or stalling in the event of a safety violation on the PoS chains; however, results in a liveness violation as the PoS protocol stops finalizing new blocks temporarily. Note that if the Babylon chain is secure with parameter $k_c/2$, by the assumption $k_c \leq k_w$, it is also secure with parameter k_w. Hence, if the protocol halts due to a safety violation on the PoS chains, at least one of the following conditions should be true:

- Babylon chain is not secure with parameter $k_c/2$, implying clause **L2**.
- Babylon chain is secure with parameters k_c and k_w, implying that at least $f+1 > n/3$ adversarial validators become slashable in the view of all honest nodes via the proof Section D.1, *i.e.*, clause **L1**.

Thus, even though the halting of the protocol due to a safety violation could cause a liveness violation, the liveness claims of Theorem 1 hold in this case as well. This concludes the liveness proof. □

Algorithm 1 Validation of the commitments sent to Babylon by the miners. Returns true if the commitment is valid given the data D.

```
function pow_validate_commitment(commitment, data, type)
    if type == checkpoint
        ▷ Parse the data for the checkpoint.
        block_headers, transaction_roots, block_bodies ← parse_block_data(data)
        ▷ Check if the transaction roots sent as part of the data commit to the PoS block bodies.
        for (block_header, transaction_root, block_body) ← block_headers, transaction_roots, block_bodies
            if transaction_root ≠ H(block_body)
                return False
            end if
        end for
        ▷ Check if the checkpoint was correctly calculated.
        if commitment == H(block_headers || transaction_roots)
            return True
        end if
        return False
    else if type == message
        ▷ Check if the message commitment was correctly calculated.
        if commitment == H(data)
            return True
        end if
        return False
    end if
end function
```

Algorithm 2 Generation of the commitments by the PoS nodes. Returns the calculated commitment.

```
function generate_commitment(data, type)
    if type == checkpoint
        block_headers, transaction_roots, block_bodies ← parse_block_data(data)
        return H(block_headers || transaction_roots)
    else if type == message
        return H(data)
    end if
end function
```

Algorithm 3 Validation of the commitments on Babylon by the full PoS nodes. Returns true if the commitment is valid given the data D.

```
function pos_validate_commitment(commitment, data, type)
    if type == checkpoint
        ▷ Parse the data for the checkpoint.
        block_headers, transaction_roots, block_bodies ← parse_block_data(data)
        for i ← 0, .., len(block_headers) − 1
            block_header ← block_headers[i]
            transaction_root ← transaction_roots[i]
            block_body ← block_bodies[i]
            ▷ Check if the checkpointed PoS blocks were finalized.
            if !is_finalized(block_header)
                return False
            end if
            ▷ Check if the checkpointed PoS blocks form a consecutive sequence on the PoS chain.
            if i ≠ len(block_headers) ∧ !block_headers[i + 1].is_ancestor(block_header || block_body)
                return False
            end if
            ▷ Check if the checkpoint was calculated with the correct transaction roots and if these roots commit to the PoS block body.
            if transaction_root ≠ block_header.transaction_root ∨ transaction_root ≠ H(block_body)
                return False
            end if
        end for
        ▷ Check if checkpoint was correctly calculated.
        if commitment == H(block_headers || transaction_roots)
            return True
        end if
        return False
    else if type == message
        ▷ Check if the message commitment was correctly calculated.
        if commitment == H(data)
            return True
        end if
        return False
    end if
end function
```

Algorithm 4 Identifying the canonical PoS chain when there is a safety violation on the PoS chain. Returns the canonical PoS chain.

```
function identify_PoS_chain(Babylon_chain, PoS_blocktree)
    PoS_canonical ← []
    ▷ Obtaining a sequence of valid checkpoints and the associated data from the PoW chain. Note that if there are two consecutive valid checkpoints on Babylon such that the second checkpoint commits to blocks conflicting with those of the first one, the second one is not returned.
    commitments, data ← Babylon_chain.get_valid_checkpoints()
    children ← [PoS_blocktree.genesis_block]
    next ← True
    ▷ i keeps track of which checkpoint in the sequence of returned valid checkpoints PoS node is currently considering.
    i ← 0
    if i ≥ len(commitments)
        block_headers, transaction_roots, block_bodies ← ⊥, ⊥, ⊥
    else
        cur_commitment ← commitments[0]
        cur_data ← data[0]
        block_headers, transaction_roots, block_bodies ← parse_block_data(cur_data)
    end if
    while children ≠ ∅
        for PoS_block ← children
            ▷ Check if the current PoS block is committed by a valid and early checkpoint, and should be part of the canonical PoS chain.
            if PoS_block.header ∈ block_headers
                children ← PoS_blocktree.get_children(PoS_block)
                next ← False
                PoS_canonical ← PoS_canonical + [PoS_block]
                ▷ All of the blocks in the current checkpoint are accounted for. PoS node now considers the PoS blocks attested by the next valid checkpoint.
                if PoS_block.header == block_headers[-1]
                    i ← i + 1
                    if i ≥ len(commitments)
                        block_headers, transaction_roots, block_bodies ← ⊥, ⊥, ⊥
                    else
                        cur_commitment ← commitments[i]
                        cur_data ← data[i]
                        block_headers, transaction_roots, block_bodies ← parse_block_data(cur_data)
                    end if
                end if
                Break
            end if
        end for
        ▷ If there is no checkpoint to help decide which children of a block on the PoS chain to follow as the next block on the canonical PoS chain, decision is made in favor of the first child returned by the get_children function.
        if next
            PoS_canonical ← PoS_canonical + [children[0]]
            children ← PoS_blocktree.get_children(children[0])
        end if
        next ← True
    end while
    return PoS_canonical
end function
```

Algorithm 5 Granting stake withdrawal request and slashing protocol violators in the event of a safety violation. Returns true if the withdrawal request for the specified validator is to be granted in the view of the PoS node running the function, slashes the stake of the validator if there is a fraud proof accusing it.

```
function grant_withdrawal_request(Babylon_chain, validator)
    commitments, data ← Babylon_chain.get_valid_checkpoints()
    ▷ get_all_checkpointed_blocks returns all checkpointed, valid and finalized PoS blocks given a sequence of commitments and data.
    pos_blocks ← get_all_checkpointed_blocks(commitments, data)
    PoS_block ← ⊥
    for block ← blocks
        if ∃ withdrawal_req by validator ∈ block
            PoS_block ← block
            Babylon_block_height ← Height of the Babylon block containing the checkpoint for PoS_block
        end if
    end for
    ▷ Without a checkpoint for the PoS block containing the withdrawal request, request cannot be granted.
    if PoS_block == ⊥
        return False
    end if
    ▷ If the checkpoint of the PoS block containing the request is not sufficiently deep in Babylon in the view of the PoS node, then the request is not granted.
    if len(Babylon_chain) ≤ Babylon_block_height + k_w
        return False
    end if
    ▷ If the checkpoint of the PoS block containing the request is indeed k_w deep in Babylon in the view of the PoS node, then the request is granted only after checking for the fraud proofs as specified by condition (3) in Section 5.4
    Babylon_fragment ← Babylon_chain[Babylon_block_height : Babylon_block_height + k_w]
    commitments, fraud_proofs ← Babylon_fragment.get_valid_fraud_proofs()
    for fraud_proof ← fraud_proofs
        if fraud_proof accuses validator
            ▷ Validator is slashed if there is a valid fraud proof on Babylon that irrefutably accuses the validator.
            slash_validator(validator, fraud_proof)
            return False
        end if
    end for
    return True
end function
```

Algorithm 6 Identifying all censoring PoS blocks with respect to a censorship complaint on Babylon. Takes as input the canonical Babylon chain, height of the Babylon block with the censorship complaint, and the canonical PoS chain. Slashes the validators that proposed or voted for the censoring blocks.

```
function is_block_censoring(Babylon_chain Babylon_block_height, PoS_block, PoS_canonical)
    commitments, censored_txs ← Babylon_chain[Babylon_block_height].get_valid_censorship_complaint()
    ▷ Get all checkpoints that extend the Babylon block with the censorship complaint by at least 2k_c blocks.
    commitments, data ← Babylon_chain[Babylon_block_height + 2k_c :].get_valid_checkpoints()
    ▷ Parse over these checkpoints starting from the second one as stated in Section 5.5.1
    for i ← 1, .., len(data) − 1
        block_headers, transaction_roots, block_bodies ← parse_block_data(data)
        ▷ Detect the censoring blocks on the PoS chain as attested by each checkpoint.
        for PoS_block ← block_headers, block_bodies
            if censored_txs ∉ PoS_canonical[: PoS_block] ∧ censored_txs valid w.r.t the state of PoS_canonical[: PoS_block]
                Slash validators that proposed and voted upon PoS_block
            end if
        end for
    end for
end function
```

Algorithm 7 Emulating a Tendermint round on Babylon when there is a stalling evidence. This function is evoked by validators if a stalling evidence is observed on the $k_w/2$-deep prefix of the longest Babylon chain. There are $n = 3f + 1$ active validators in total.

```
function emulate_round_on_Babylon
    upon stalling_evidence ∈ Babylon_block
        checkpoint_height ← Babylon_chain.get_last_checkpoints_height()
        commitment, data ← Babylon_chain.get_last_checkpoint()
        block_headers, transaction_roots, block_bodies ← parse_block_data(data)
        ▷ Stalling evidence is taken into consideration only if it comes to Babylon at least 2k_c blocks after the last checkpoint on Babylon.
        if Babylon_block.height ≥ checkpoint_height + 2k_c
            ▷ If the validator has observed new PoS blocks finalized at the heights not covered by the last checkpoint, it sends a checkpoint for the new PoS blocks instead of emulating the round on Babylon.
            if last_finalized_PoS_block.header ∉ block_headers
                Send checkpoint to Babylon
            else
                ▷ See Appendix C.2 for more information on how proposal messages are selected and structured.
                Send Tendermint proposal to Babylon
            end if
        end if
    end upon
    ▷ This is block b_1 on Figure 7.
    upon Babylon block with height checkpoint_height + 3k_c
        proposals ← Babylon_chain[−k_c :].get_non_censoring_proposals()
        if len(proposals) < 2f + 1
            Slash validators with missing or censoring proposals
            Send new stalling evidence
            return
        else
            ▷ Proposal with the largest validRound is selected, details are in Appendix C.2.
            round_proposal ← select_round_proposal(proposals)
            ▷ See Appendix C.2 for more information on how prevotes and precommits are structured.
            Send prevote and precommit to Babylon for round_proposal
        end if
    end upon
    ▷ This is block b_2 on Figure 7.
    upon Babylon block with height checkpoint_height + 4k_c
        prevotes ← Babylon_chain[−k_c :].get_prevotes(proposal)
        precommits ← Babylon_chain[−k_c :].get_precommits(proposal)
        if len(prevotes) < 2f + 1 ∨ len(precommits) < 2f + 1
            Slash validators whose prevotes and precommits are missing for round_proposal
            Send new stalling evidence
            return
        else
            Finalize round_proposal
        end if
    end upon
end function
```