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

Ertem Nusret Tas
Stanford University
nusret@stanford.edu

David Tse
Stanford University
dntse@stanford.edu

Fangyu Gai
BabylonChain
fangyu.gai@babylonchain.io

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

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

Fisher Yu
BabylonChain
fisher.yu@babylonchain.io

***Abstract*—Bitcoin is the most secure blockchain in the world, supported by the immense hash power of its Proof-of-Work miners. Proof-of-Stake chains are energy-efficient, have fast finality but face several security issues: susceptibility to non-slashable long-range safety attacks, low liveness resilience and difficulty to bootstrap from low token valuation. We show that these security issues are inherent in any PoS chain without an external trusted source, and propose a new protocol, Babylon, where an off-the-shelf PoS protocol checkpoints onto Bitcoin to resolve these issues. An impossibility result justifies the optimality of Babylon. A use case of Babylon is to reduce the stake withdrawal delay: our experimental results show that this delay can be reduced from weeks in existing PoS chains to less than 5 hours using Babylon, at a transaction cost of less than 10K USD per annum for posting the checkpoints onto Bitcoin.**

I. Introduction

A. From Proof-of-work to proof-of-stake

Bitcoin, the most valuable and arguably the most secure blockchain in the world, is supported by a proof-of-work (PoW) protocol that requires miners to compute many random hashes. Many newer blockchain projects eschew the proof-of-work paradigm in favor of proof-of-stake (PoS). A prominent example is Ethereum, which has migrated from PoW to PoS, a process that lasted 6 years. Other prominent PoS blockchains include single chain ecosystems such as Cardano, Algorand, Solana, Avalanche as well as multi-chain ecosystems such as Polkadot and Cosmos. The Cosmos ecosystem consists of many application-specific zones, all built on top of the Tendermint PoS consensus protocol [26], [25].

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

B. Proof-of-stake security issues

Security of PoS protocols has traditionally been shown under the honest majority assumption, which states that the honest parties hold the majority of the stake [33], [20], [32], [26]. Introduced by Buterin and Griffith [28], the concept of *accountable safety* enhances the notion of security under honest majority with the ability to provably identify protocol violators in the event of a safety violation. Thus, accountable safety not only implies security under an honest majority, but also the identification of protocol violators if a large quorum of the validators are adversarial and cause a safety violation. In lieu of making an unverifiable honest majority assumption, this approach aims to obtain a *cryptoeconomic* notion of security by holding protocol violators accountable and *slashing* their stake, thus enabling an exact quantification of the penalty for protocol violation. This *trust-minimizing* notion of security is central to the design of PoS protocols such as Gasper [29], the protocol supporting PoS Ethereum, and Tendermint [26], [25], supporting the Cosmos ecosystem. However, there are several fundamental limitations to the security of the PoS protocols:

Safety attacks are not slashable: While a PoS protocol with accountable safety can identify attackers, slashing of their stake is not always possible, implying a lack of *slashable safety*. For example, a posterior corruption attack can be mounted using old coins after the stake is withdrawn, and therefore cannot be slashed [27], [33], [20], [37]. 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. Posterior corruption is a long-known problem with PoS protocols, and several approaches have been proposed to deal with it under the honest majority assumption (Section II). Theorem 1 in Section IV says that no PoS protocol can provide slashable safety without *external* trust assumptions. A typical external trust assumption used in practice is *off-chain social consensus checkpointing*. As social consensus is a slow process, this type of checkpointing leads to a long *withdrawal delay* after a validator requests to withdraw its stake (*e.g.*, 21 days for Cosmos zones [14]), which reduces the liquidity of the system. Moreover, social consensus cannot be relied upon in smaller blockchains with an immature community.

Low liveness resilience: PoS protocols such as Snow

Contact author: DT

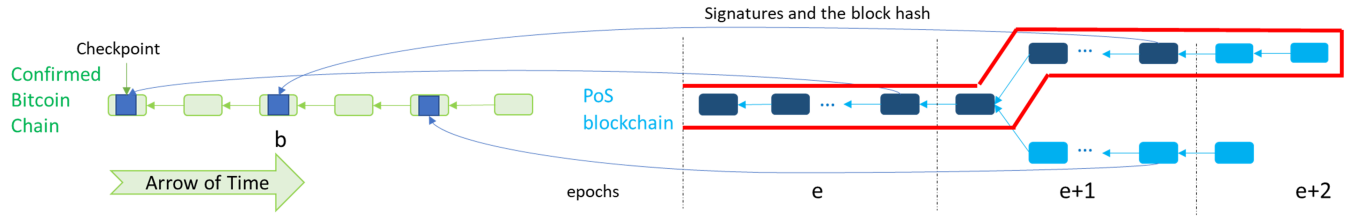

Fig. 1: Babylon posts PoS block hashes and the validator signatures on them to Bitcoin. Ordering of these hashes enable clients to break ties between conflicting PoS chains, and slash adversarial validators before they withdraw after a safety violation. The canonical PoS chain in a client c's view is shown by the red circle. Dark blue blocks represent the checkpointed chain of PoS blocks in c's view. The fast finality rule determines the PoS chain, while the slow finality rule determines the checkpointed chain, which is always a prefix of the PoS chain.

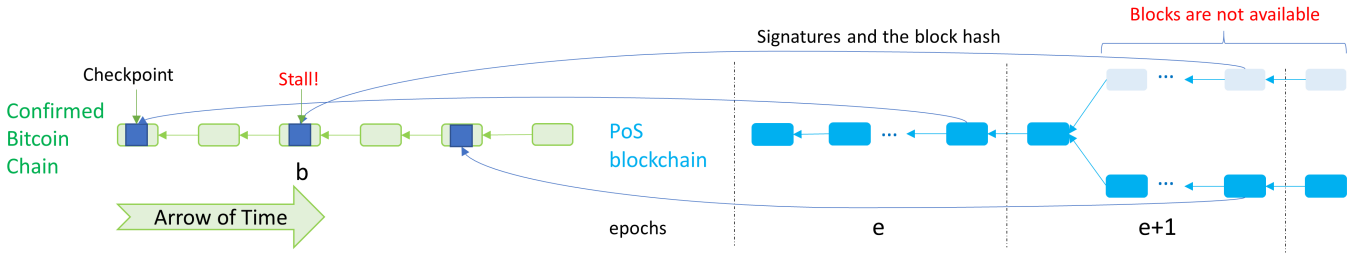

Fig. 2: An adversary that controls a supermajority of active validators finalizes PoS blocks on an attack chain (top). It keeps the attack chain private, yet posts the hashes of the private blocks and their signatures on Bitcoin. Once these checkpoints are deep in Bitcoin, adversary helps build a conflicting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

White [49], [33] and Ouroboros [42], [34], [20] guarantee liveness as long as the fraction of adversarial stake is below $1/2$. However, PoS protocols with accountable safety such as Tendermint and Gasper cannot ensure liveness beyond an adversarial fraction of $1/3$. This low liveness resilience of $1/3$ is a fundamental limitation of PoS protocols with accountable safety, even under synchrony [55, Appendix B].

The bootstrapping problem: Even if a PoS protocol provides slashable safety, the maximum penalty 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.*, Cosmos zones) start with a low token valuation. This makes it difficult for new PoS chains to support high-valued applications (*e.g.*, decentralized finance, NFTs). Similarly, a PoS chain that experiences a significant drop in token valuation becomes more susceptible to attacks.

C. Leveraging external trust

The main reason behind the security issues described above is the absence of a reliable *arrow of time*. For instance, posterior corruption attacks exploit the inability of the late-coming clients to distinguish between the canonical chain minted by the honest validators and the adversary's history-revision chain that is published much later [33], [37]. Hence, to guarantee a slashable notion of safety, PoS protocols need an external trust source that can periodically and publicly timestamp the canonical chain. Social consensus can be viewed as one such source of external trust, but because it is achieved off-chain, its level of security is hard to quantify. In this paper, we explore a more quantifiable approach, which is to use an *existing secure blockchain* as a source of external trust. Given such a trusted blockchain, we ask: *What is the limit to the security enhancement the trusted chain can provide to a PoS chain and what is the optimal protocol that achieves this limit?*

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

The idea of using a trusted parent chain to provide security to a PoS chain has been used in several industry projects and academic works. Most of these works focus on mitigating specific attack vectors. For example, a recently proposed protocol, BMS [56], uses Ethereum to keep track of the dynamic validator set of a PoS chain to withstand posterior corruption attacks. (That work was later extended

to a protocol using Bitcoin in a concurrent work [19] to ours.) In our paper, we broaden the investigation to find out the best security guarantees a trusted public blockchain such as Bitcoin can provide to a PoS chain, and construct an optimal protocol, Babylon, that achieves these guarantees. A detailed comparison of Babylon and other approaches is described in Table I and Section II.

D. Babylon

A PoS protocol is executed by *validators*, which lock up their stake in a contract on the PoS chain. The design of Babylon specifies the kind of information validators post on Bitcoin and how this information is used by the clients, *i.e.*, external observers of the protocol, to resolve attacks on the PoS chain (*cf.* Figure 1). Its highlights are presented below:

a) Checkpointing: PoS protocol proceeds in epochs during which the validator set is fixed. The honest validators sign the hash of the last PoS block of each epoch (*cf.* Figure 1). They subsequently post the hash and their signatures to Bitcoin as *checkpoints*. Ordering imposed on these checkpoints by Bitcoin enable the clients to resolve safety violations, and identify and slash adversarial validators engaged in posterior corruption attacks before they withdraw their stake.

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

Since Bitcoin helps resolve earlier forks and obtain a unique checkpointed chain, safety can only be violated for recent PoS blocks in c's view. Hence, adversarial validators cannot violate the safety of older PoS blocks through long range posterior corruption attacks after withdrawing their stake. On the other hand, if a safety attack is observed for the recent PoS blocks, c can detect the adversarial validators and enforce the slashing of their stake. Protocol thus ensures slashable safety.

c) Emergency break: If the adversary controls a supermajority of the validators, it can sign the hashes of PoS blocks privately, and keep the blocks hidden from the clients. A client stops adding new blocks to its PoS chain if it observes a signed checkpoint on Bitcoin, yet the corresponding block is unavailable. This *emergency break* is necessary to protect against data unavailability attacks (*cf.* Figure 2). If the checkpoints consisted only of the block hashes, then the adversary could stall the PoS chain by sending a *single* hash and pretending like it is the checkpoint of an unavailable PoS block. Thus, the signatures on the checkpoints ensure that the adversary cannot cause an emergency break, unless it corrupts the supermajority of the current validator set.

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

e) Bitcoin safety & slow finality fork-choice rule: Clients can achieve Bitcoin safety for their PoS chains if they adopt a slow finality rule, where they only output the checkpointed chain in their views. They wait until a PoS block or its descendants are checkpointed on Bitcoin before outputting the block as part of the PoS chain. In this case, the PoS chain is always safe (assuming Bitcoin is secure), however, its latency now becomes as large as Bitcoin latency.

E. Security guarantees

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

Safety: Under the fast finality rule, Babylon achieves slashable safety via checkpointing, and stalling whenever data is unavailable. Slashable safety is not possible without an external source of trust. Moreover, Babylon achieves this with a stake withdrawal delay equal to the Bitcoin confirmation latency, *i.e.*, the time for the timestamp of a withdrawal request to be confirmed on Bitcoin. To estimate this delay, we implemented a checkpointing protocol and measured the confirmation latency for the checkpoints. See Section VI.

Liveness: Babylon improves the liveness resilience from $1/3$ to $1/2$ by using Bitcoin as a fallback. However, when the adversarial fraction exceeds $1/2$, liveness cannot be guaranteed. As shown by Theorem 4, this is not Babylon's fault, but is inherent in any protocol that does not post the entire PoS transaction data to Bitcoin. Then, the protocol is susceptible to data unavailability attacks. In Section V, we show one such attack on *inactivity leak*, the method used by PoS Ethereum and Cosmos to slash inactive validators [23], [9].

Bootstrapping: Under the slow finality rule, Babylon is safe no matter how many adversarial validators there are, if Bitcoin is secure. Thus, Babylon achieves Bitcoin safety, albeit at the expense of Bitcoin confirmation latency. Thus is useful in a bootstrapping mode or for important transactions, where slashable safety is not sufficient.

F. Outline

The rest of the paper is organized as follows. Section II reviews related work. Section III presents the model and definitions of various security notions. In Section IV, we show that slashable safety is not possible for PoS chains without external trust. We then present Babylon 1.0, a Bitcoin-checkpointing protocol that provides slashable safety. In Section V, we show the impossibility of liveness beyond a $1/2$ adversarial

	Safety	Liveness			Withdrawal
		$f < n/3$	$n/3 \leq f < n/2$	$f \geq n/2$	
KES [32]	$n/3$-safe	PoS Latency	No guarantee	No guarantee	?
Pikachu [19]/BMS [56]	$n/3$-safe	PoS Latency	No guarantee	No guarantee	Bitcoin latency
Babylon: fast finality	$n/3$-slashably safe	PoS Latency	Bitcoin Latency	No guarantee	Bitcoin latency
Babylon: slow finality	Always safe	Bitcoin Latency	Bitcoin Latency	No guarantee	Bitcoin latency

TABLE I: The security guarantees of Babylon compared to other solutions, assuming the security of Bitcoin for Babylon and the security of Ethereum for BMS. Here, f is the number of adversarial validators and n is the total number of validators. m-safe means the protocol is safe whenever $f < m$, m-slashable-safe means that whenever safety is violated, m validators can be slashed (which is a stronger property than m-safe). Stake withdrawals happen with Bitcoin latency on Babylon as long as liveness is satisfied, whereas it happens with Ethereum latency on BMS. In theory, Algorand [32] can grant stake withdrawal requests in seconds as it uses key-evolving signatures (KES) to recycle keys after every signature, but since KES is highly incentive incompatible, Algorand still uses social consensus checkpointing.

fraction of validators, even when there is a data-limited source of external trust. We then improve Babylon 1.0 to the full Babylon protocol to provide the optimal liveness resilience. In Section VI, we provide measurements for the confirmation latency of Bitcoin transactions containing checkpoints and demonstrate the feasibility of the Babylon protocol. In Section VII, we describe the slow finality rule that provides Bitcoin safety to the PoS chains.

II. Related Works

A. Posterior corruption attacks

Among the PoS security issues discussed in Section I-B, posterior corruption attacks is the most well-known, [27], [33], [20], [37]. In a posterior corruption attack also known as founders' attack, long range attack, history revision attack or costless simulation, adversary acquires the old keys of the validators after they withdraw their stake. It then re-writes the protocol history by building a conflicting *attack* chain in private. The attack chain forks from the canonical one at a past block, where the old keys constituted a majority of the validator set. Subsequently, the adversary replaces the old validators with new ones under its control, and reveals the attack chain to the clients observing the system at a later time. The adversary thus causes clients to adopt conflicting chains at different times.

Several solutions have been proposed to mitigate the posterior corruption attacks: 1) checkpointing via social consensus (*e.g.*, [27], [33], [18], [21]); 2) use of key-evolving signatures (*e.g.*, [32], [42], [20]); 3) use of verifiable delay functions (*e.g.*, [59]); 4) timestamping on an existing PoW chain like Bitcoin [19].

1) Social consensus: Social consensus refers to a trusted committee, distinct from the PoS validators, which periodically checkpoints the finalized PoS blocks on the canonical chain. It attempts to prevent posterior corruption attacks by making the blocks on the attack chain distinguishable from the checkpointed ones on the canonical chain. For instance, in PoS Ethereum, clients identify the canonical chain with the help of checkpoints received from their peers. Since no honest peer provides a checkpoint on a private chain, posterior corruption attacks cannot confuse new validators [17].

As clients might receive their checkpoints from different peers, it is often difficult to quantify the trust assumption placed on social consensus. A small committee of peers shared by all clients would imply centralization of trust, and make security prone to attacks targeting few nodes. Conversely, a large committee would face the problem of reaching consensus on checkpoints in a timely manner, leading to long withdrawal delays. For instance, Avalanche, Cosmos zones and PoS Ethereum have withdrawal delays of 14, 21 and 13[1] days respectively [13], [14], [47].

2) Key-evolving signatures (KES): KES requires the validators to forget old keys so that a posterior corruption attack cannot be mounted. Security has been shown for various PoS protocols using key-evolving signatures under the honest majority assumption for the current, active validators [32], [20]. This assumption is necessary to ensure that the majority of the active validators willingly forget their old keys so that they cannot be given to an adversary at a later time. However, there might be a strong incentive for the validators to record their old keys in case they later become useful. Thus, KES render the honest majority assumption itself questionable by asking honest validators for a favor which they may be tempted to ignore for future gain. This observation is formalized in Section IV-B, which shows that KES cannot provide slashable safety for PoS protocols.

3) Verifiable Delay Functions (VDFs): VDFs can help the clients distinguish the canonical chain generated a long time ago from an attack chain created much later, thus providing an arrow of time for the clients and protecting the PoS protocol from posterior corruption attacks. However, like KES, VDFs standalone cannot provide slashable safety for PoS protocols (*cf.* Section IV-B). Another problem with VDFs is the possibility of finding faster functions [16], which can then be used to mount a posterior corruption attack.

4) Timestamping the validator set: Posterior corruption attacks can be thwarted by timestamping the PoS validator set on an external public blockchain such as Ethereum [56] and Bitcoin [19]. For instance, Blockchain/BFT Membership Service (BMS) [56] uses an Ethereum smart contract as a

[1]This is calculated for $130,000$ attesters with an average balance of 32 ETH to accurately model the targeted attester numbers on PoS Ethereum using [47, Table 1].

reconfiguration service that records the changes in the *current* validator set. When validators request to join or leave the current set, the existing validators send transactions containing the new validator set to the contract. Upon receiving transactions with the same new validator set from sufficiently many existing validators (*e.g.*, from over $1/3$ of the current validator set), the contract replaces the current set with the new one.

The goal of BMS is to protect the PoS protocol against posterior corruption attacks, where the adversary creates an attack chain using the credentials of old validators. These old validators are then replaced by new, adversarial ones that are distinct from those on the canonical chain. However, if the honest validators constitute over $2/3$ of the current validator set, the new adversarial validators on the attack chain cannot be validated and recorded by the contract before the new honest validators on the canonical chain. Hence, BMS enables the late-coming clients to identify and reject the attack chain, providing security to PoS chains as long as the fraction of active adversarial validators is bounded by $1/3$ at any given time (*cf.* Table I). By preventing posterior corruption attacks, BMS can also reduce the withdrawal delay of the PoS protocols from weeks to the order of minutes.

To prevent posterior corruption attacks, BMS requires an honest supermajority assumption on the current set of validators, and as such, does not provide slashable safety. If the adversary controls a supermajority of the validator set, it can create an attack chain in private, simultaneous with the public canonical chain, and post the validator set changes of the private chain to the contract before the changes on the canonical chain. Then, the late-coming clients could confuse the canonical chain for an attack chain, and identify its validators as protocol violators, which implies lack of slashable safety.

Unlike Babylon, BMS cannot ensure liveness if the fraction of adversarial active validators exceeds $1/3$ (*cf.* Table I). Similarly, whereas Babylon can provide Bitcoin safety for young PoS chains and important transactions, BMS cannot provide Ethereum safety to the constituent PoS protocols even by adopting a slow finalization rule. This is because the BMS contract only keeps track of the PoS validators, and does not receive any information about the PoS blocks themselves such as block headers.

B. Hybrid PoW-PoS protocols

A PoS protocol timestamped by Bitcoin is an example of a *hybrid PoW-PoS protocol*, where consensus is maintained by both the PoS validators and Bitcoin miners. One of the first such protocols is Casper FFG, a finality gadget used in conjunction with a longest chain PoW protocol [28]. The finality gadget is run by PoS validators as an overlay to checkpoint and finalize blocks in an underlay PoW chain, where blocks are proposed by the miners. The finality gadget architecture is also used in many other PoS blockchains, such as PoS Ethereum [29] and Polkadot [57]. Bitcoing timestamping can be viewed as a "reverse" finality gadget, where the miners run an overlay PoW chain to checkpoint the underlay PoS chains run by their validators. In this context, our design that combines Bitcoin with PoS protocols leverages off ideas from a recent line of work on secure compositions of protocols [45], [53], [46].

Combining a proof-of-work blockchain with classical BFT protocols was also explored in Hybrid consensus [48] and Thunderella [50]. Hybrid consensus selects the miners of a PoW-based blockchain as members of a rotating committee, which then runs a permissioned and responsive BFT protocol that is secure up to $1/3$ adversarial fraction. Unlike Babylon, Hybrid consensus cannot use Bitcoin as is, with no change to Bitcoin, and does not provide accountable or slashable safety. Thunderella combines a responsive BFT protocol with a slow, longest chain protocol to achieve responsiveness and security up to $1/4$ and $1/2$ adversarial fraction respectively. To improve the adversarial fraction tolerable for liveness from $1/3$ to $1/2$, Babylon uses insights from Thunderella [50]. However, unlike Babylon, Thunderella (in the PoW setting) cannot use Bitcoin as is and does not provide slashable safety.

C. Timestamping

Timestamping data on Bitcoin has been used for purposes other than resolving the limitations of PoS protocols. For instance, timestamping on Bitcoin was proposed as a method to protect Proof-of-Work (PoW) based ledgers against 51% attacks [41]. However, this requires the Bitcoin network to contain *observing* miners, which publish timestamps from the PoW ledger to be secured, only if the block data is available. This implies changing Bitcoin to incorporate data-availability checks for external ledgers, whereas in our work, we analyze the limitations of security that can be achieved by using Bitcoin as is.

Two projects that use Bitcoin to secure PoS and PoW child chains are Veriblock [52] and Komodo [7]. Both projects suggest checkpointing child chains on Bitcoin to help resolve forks. However, they lack proper security proofs, and do not analyze how attacks on PoS chains can be made slashable. Another use-case of timestamping is posting commitments of digital content to Bitcoin to ensure integrity of the data [40]. In this vein, [39] implements a web-based service to help content creators prove possession of a certain information in the past by posting its timestamps on Bitcoin.

Other projects planning to use Bitcoin checkpointing include Filecoin, which aims to prevent posterior corruption attacks using the checkpoints, and BabylonChain that serves as a checkpoint aggregator for Cosmos zones to decrease the checkpointing cost in the presence of many chains [11], [1].

III. Model

Notation. Given a positive integer m, $[m]$ denotes the set $\{1, 2, \ldots, m\}$. We denote PoS blocks by capital B and the Bitcoin blocks by lowercase b. An event is said to happen with overwhelming probability if it happens except with negligible probability in the security parameter λ.

Validators and clients. In the client-server setting of state machine replication (SMR), there are two sets of nodes: validators and clients. Validators receive transactions as input,

and execute a SMR protocol. Their goal is to ensure that the clients obtain the same sequence of transactions, thus, the same end state. We assume that the transactions are batched into *blocks*, and the sequence obtained by the clients is a blockchain, denoted by $\mathcal{L}$. Thus, we hereafter refer to the SMR protocols as *blockchain protocols*.

To output a chain, clients collect consensus messages from the validators and download blocks. Upon collecting messages from a sufficiently large quorum of validators, each client outputs a chain. Clients can choose to output a chain at arbitrary times, and might be offline in between. The set of clients include honest validators, as well as external nodes that observe the protocol infrequently.

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

Blocks and chains. Each block consists of a block header and transaction data. Block headers contain (i) a pointer (*e.g.*, hash) to the parent block, (ii) a vector commitment (*e.g.*, Merkle root) to the transactions data, and (iii) protocol related messages. The total order across the blocks in a chain together with the ordering of the transactions by each block gives a total order across all transactions within the chain.

For a block B, we say that $B \in \mathcal{L}$, if B is in the chain $\mathcal{L}$. Similarly, we say $\mathsf{tx} \in \mathcal{L}$, if the transaction tx is included in a block that is in $\mathcal{L}$. A block B is said to *extend* B', if B' can be reached from B by following the parent pointers. Conversely, the blocks B and B' are said to *conflict* with each other if B' does not extend B and vice versa. The notation $\mathcal{L}_1 \prec \mathcal{L}_2$ implies $\mathcal{L}_1$ is a strict prefix of $\mathcal{L}_2$, whereas $\mathcal{L}_1 \preceq \mathcal{L}_2$ implies $\mathcal{L}_1$ is either a prefix of or the same as $\mathcal{L}_2$. The chains $\mathcal{L}_1$ and $\mathcal{L}_2$ conflict with each other if they contain conflicting blocks.

Environment and adversary. Transactions are input to the validators by the environment $\mathcal{Z}$. Adversary $\mathcal{A}$ is a probabilistic polynomial time algorithm. Before the protocol execution starts, adversary can corrupt a subset of validators, which are subsequently called *adversarial*. These validators surrender their internal states to the adversary and can deviate from the protocol arbitrarily (Byzantine faults) under the adversary's control. The remaining validators are called *honest* and follow the blockchain protocol as specified. Time is slotted, and the validators are assumed to have synchronized clocks[2].

Networking. Validators can broadcast messages to each other and the clients. Messages are delivered by the adversary, which can observe a message before it is received. Network is synchronous, *i.e.*, the adversary delivers the messages sent by an honest validator to *all* other honest validators and clients within Δ slots. Here, Δ is a known parameter. Upon joining the network at slot t, new validators and late-coming clients receive all messages broadcast by the honest validators before slot $t - \Delta$.

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

- **Safety:** For any slots r, r' and clients c, c', either $\mathcal{L}_r^{\mathsf{c}}$ is a prefix of $\mathcal{L}_{r'}^{\mathsf{c}'}$ or vice versa. For any client c, $\mathcal{L}_r^{\mathsf{c}}$ is a prefix of $\mathcal{L}_{r'}^{\mathsf{c}}$ for all slots r and r', $r' \geq r$.
- $\mathbf{T}_{\text{fin}}$**-Liveness:** If $\mathcal{Z}$ inputs a transaction tx to an honest validator at some slot r, then, $\mathsf{tx} \in \mathcal{L}_{r'}^{\mathsf{c}}$ for any slot $r' \geq r + T_{\text{fin}}$ and any client c.

Let f denote the upper bound on the number of adversarial validators over the protocol execution. A PoS protocol provides f_{s}-safety if it satisfies safety whenever $f \leq f_{\text{s}}$. Similarly, a PoS protocol provides f_{l}-T_{fin}-liveness if it satisfies T_{fin}-liveness whenever $f \leq f_{\text{l}}$.

Accountable safety. To formalize accountable safety, we use the forensic analysis model by Sheng et al. [55]. During normal execution, validators exchange messages (*e.g.*, blocks or votes), and each validator records all protocol-specific messages received in an execution transcript. If a client observes a safety violation, it invokes a forensic protocol by sending the conflicting chains to the validators. Then, the honest validators send their transcripts to the client. The forensic protocol takes these transcripts, and outputs a proof that identifies f adversarial validators as protocol violators. This proof is subsequently sent to all other clients, and serves as evidence that the identified validators have irrefutably violated the protocol rules.

Definition 1. *A blockchain protocol is said to provide accountable safety with resilience f if when there is a safety violation, (i) at least f adversarial validators are identified by the forensic protocol as protocol violators, and (ii) no honest validator is identified, with overwhelming probability. Such a protocol provides* f-accountable-safety.

f-accountable-safety implies f-safety, and as such is a stronger property. Accountably-safe protocols considered in this paper have the same safety and accountable safety resilience, though this is not necessarily true for all protocols.

Proof-of-Stake (PoS) protocols. In a permissioned protocol, the set of validators stays the same over time. In contrast, PoS protocols allow changes in the validator set. In a PoS protocol, nodes lock up stake in a contract executed on the blockchain to become validators. To distinguish the validators that are currently executing the protocol at a given time from the old validators, we will refer to the current validators as *active*. We assume that each active validator has the same stake, and is equipped with a unique cryptographic identity. Once a validator becomes inactive, it immediately becomes adversarial if it has not been corrupted before, thus might engage in posterior corruption attacks. Protocol execution starts with an initial committee of n validators, and the contract allows at most n validators to be active at any slot. We assume that at all slots, there is a non-empty queue of nodes waiting to stake their coins, and each validator leaving the active set

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

can be immediately replaced by a new validator[3].

PoS protocols proceed in epochs starting at 1 and measured in the number of PoS blocks. For instance, if each epoch lasts m blocks and a client observes a chain of $5m + 3$ PoS blocks, then the first m blocks belong to the first epoch, the second m blocks to the second one, and so on, until the last 3 blocks, which are part of the on-going epoch 6. During an epoch, the active validator set is fixed and the execution of the PoS protocol mimicks that of a permissioned blockchain protocol. Across epochs, the PoS protocol supports changes in the active set through withdrawals. An active validator can send a *withdrawal* request to the protocol to leave the active set and retrieve its staked coin. At the end of each epoch, clients inspect their chains and identify the validators whose withdrawal requests have been included in the chain. Then, at the next epoch, these validators are replaced with new ones from the staking queue.

When a validator leaves the active set, its coin is *not* necessarily released by the contract immediately. Different PoS protocols can have different *withdrawal delays*. The withdrawal mechanism is central to security, and will be analyzed in later sections. If a withdrawing validator's stake is first released in the view of a client c at slot r, the validator is said to have *withdrawn its stake* in c's view at slot r.

Model for Bitcoin. We model Bitcoin using the backbone formalism [38] and treat it as a black-box blockchain protocol, which accepts transactions and outputs a totally ordered sequence of Bitcoin blocks containing these transactions. To output the Bitcoin chain *confirmed* with parameter k at slot r, a client c takes the longest chain of Bitcoin blocks in its view, removes the last k blocks, and adopts the k deep prefix as its Bitcoin chain at slot r. We denote the Bitcoin chain outputted by c at slot r by $\mathcal{C}_r^{\mathsf{c}}$. If a Bitcoin block b or transaction tx first appears in the confirmed Bitcoin chain (hereafter called the Bitcoin chain) of a client c at slot r, we say that tx or b has become *confirmed* in c's view at slot r. We say that Bitcoin is secure with parameter k if it satisfies safety and T_{fin}-liveness given the k-deep confirmation rule. Here, T_{fin} satisfies the following proposition:

Proposition 1 (Chain Growth)**.** *Suppose Bitcoin is secure with parameter k with overwhelming probability. Then, for any client c, if a transaction tx is sent to Bitcoin at slot r such that $|\mathcal{C}_{r-3\Delta}^{\mathsf{c}}| = \ell$, $\mathsf{tx} \in \mathcal{C}_{r'}^{\mathsf{c}}$ for any $r' \geq r + T_{\text{fin}}$, and $|\mathcal{C}_{r+T_{\text{fin}}}^{\mathsf{c}}| \leq \ell + k$ with overwhelming probability.*

If the adversarial fraction of the mining power is less than $1/2 - \epsilon$ for some $\epsilon > 0$, then there exists a parameter k polynomial in λ such that Bitcoin is secure with parameter k and satisfies the above proposition [38].

IV. Optimal safety

We next formalize the concept of slashable safety, and analyze a simplified version of Babylon, called Babylon 1.0, which achieves the optimal safety guarantees. The full protocol with the optimal liveness guarantees is presented in Section V.

A. Slashable safety

A useful feature of the PoS protocols is the ability to impose financial punishments on the protocol violators through the slashing of their locked stake. Slashable security extends the notion of accountability to PoS protocols. A validator v becomes *slashable* in the view of a client c at slot r if,

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

In practice, once the contract that locks v's stake receives a proof accusing v, it will attempt to slash v's stake if it is not withdrawn yet. However, if liveness is violated, the chain might not execute new transactions, and slash v's stake. Consequently, we opted to use the word 'slashable' to indicate the conditional nature of *slashing* on the resumption of chain activity after the security violation. We discuss the conditions under which stake can be slashed inSection V-C.

Definition 2. *A blockchain protocol is said to provide slashable safety with resilience f, if when there is a safety violation, at least f adversarial validators become slashable in the view of all clients, and (ii) no honest validator becomes slashable in any client's view, with overwhelming probability. Such a protocol provides f-slashable-safety.*

Slashable safety resilience of f implies that in the event of a safety violation, all clients identify f or more adversarial validators as protocol violators before they withdraw their stake, and no client identifies any honest validator as a protocol violator, with overwhelming probability. By definition, PoS protocols that provide f-slashable-safety also provide f-accountable safety and f-safety.

B. Slashable safety is not possible without external trust

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

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

[3]Many blockchains such as CosmosHub and Osmosis designate the top n participants in the queue with the largest stake as the validator set [12], [6].

A related but different theorem by Daian et al. states that without additional trust assumptions, it is impossible to design a secure PoS protocol, even under the honest majority assumption for the active validators, due to posterior corruption attacks [33, Theorem 2]. However, key-evolving signatures (KES) have been shown to prevent these attacks and help guarantee security under honest majority [34]. In contrast, Theorem 1 states that external trust assumptions are needed to build PoS protocols with slashable safety, which is a stronger property than safety under honest majority. To emphasize this point, the following paragraph presents an attack that illustrates how KES falls short of providing slashable safety despite mitigating posterior corruption attacks.

An attack on slashable safety given key-evolving signatures and VDFs. Suppose the adversary controls a supermajority of the active validators. In the case of KES, the adversarial active validators record their old keys, and use them to initiate a posterior corruption attack after withdrawing their stake. They can thus cause a safety violation, yet, cannot be slashed as the stake is withdrawn. In the case of VDFs, the adversarial active validators again construct a private attack chain while they work on the canonical one, and run multiple VDF instances simultaneously for both chains. After withdrawing their stake, they publish the attack chain with the correct VDF proofs, causing a safety violation without any slashing of their stake.

C. Babylon 1.0 protocol with fast finality

To provide PoS protocols with slashable safety, Bitcoin can be used as the additional source of trust. Babylon 1.0 is a checkpointing protocol which can be applied on any PoS blockchain protocol with accountable safety (*e.g.*, PBFT [30], Tendermint [26], HotStuff [60], Streamlet [31], PoS Ethereum finalized by Casper FFG [28], [29]) to upgrade the accountability guarantee to slashable safety. We describe Babylon 1.0 applied on Tendermint, which satisfies $n/3$-accountable safety.

Let c denote a client (*e.g.*, a late-coming client, an honest validator), whose goal is to output the *canonical* PoS chain $\mathcal{L}$ that is consistent with the chains of all other clients. We assume that c is a full node that downloads the PoS block headers and the corresponding transaction data. It also downloads the Bitcoin blocks and outputs a Bitcoin chain $\mathcal{C}$, confirmed with parameter k. In the following description, *finalized* blocks refer to the PoS blocks finalized by Tendermint, *i.e.*, the blocks with a $2n/3$ quorum of pre-commit signatures by the corresponding active validator set. A block finalized by Tendermint is not necessarily included in the canonical PoS chain $\mathcal{L}$ of a client. For instance, when the adversary engages in a posterior corruption attack, conflicting PoS blocks might seem finalized, forcing the late-coming clients to choose a subset of these blocks as part of their canonical PoS chain. Algorithm 1 describes how clients interpret the Bitcoin checkpoints to agree on the canonical chain.

a) Checkpointing the PoS chain: At the end of each epoch, the honest active validators sign the hash of the last finalized PoS block of the epoch (the hash is calculated using a collision-resistant hash function). Then, an honest active validator v sends a Bitcoin transaction called the *checkpoint transaction*. The transaction contains the hash of the block, its epoch and a quorum of signatures on the hash from over $2n/3$ active validators of the epoch. These signatures can be a subset of the (pre-commit) signatures that have finalized the block within the PoS protocol. We hereafter refer to the block hash, epoch number and the accompanying signatures within a checkpoint transaction collectively as a *checkpoint*.

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

b) Fork-choice rule: (Alg. 1, Figure 1) To identify the canonical PoS chain $\mathcal{L}^{\mathsf{c}}_r$ at some slot r, c first downloads the finalized PoS blocks and constructs a blocktree denoted by $\mathcal{T}$. The blocktree contains the quorums of pre-commit signatures

Algorithm 1 The function used by a bootstrapping client c to find the canonical PoS chain $\mathcal{L}^{\mathsf{c}}_r$ at some slot r. It takes the blocktree $\mathcal{T}$ of finalized PoS blocks and the confirmed Bitcoin chain $\mathcal{C}$ in c's view at slot r as input, and outputs $\mathcal{L}^{\mathsf{c}}_r$. The function GETCKPTS outputs the sequence of checkpoints on the given Bitcoin chain $\mathcal{C}$ in the order of their appearance. The function GETACTVALS takes a PoS chain L, an epoch number ep, and outputs the active validators of epoch ep as determined by the given chain. The function ISVALID checks if the given checkpoint contains signatures by $2/3$ of the given active validators on its block hash, and if its epoch number matches the given number ep. The function GETCHAIN returns the finalized chain within the given blocktree that ends at the preimage of the given block hash. It returns $\perp$ if a block at the preimage of the hash or its prefix chain is unavailable or not finalized. The function $\mathrm{ep}(.)$ returns the epoch of the last block within the input PoS chain. The function ISLAST returns true iff the given PoS chain ends at a block that is the last block of its epoch. The function GETCHILDREN returns the children of the given block within the blocktree. L_0 denotes the genesis PoS block and the letter L is reserved for PoS chains.

```
1:  function OUTPUTPOSCHAIN(𝒯, 𝒞)
2:      h_1, ..., h_m ← GETCKPTS(𝒞)
3:      𝒞𝒫, ep, actval ← L_0, 1, GETACTVALS(L_0, 1)
4:      for i = 1 to m                          ▷ Obtain the checkpointed chain
5:          if ISVALID(h_i, actval, ep)         ▷ Check validity.
6:              L_i ← GETCHAIN(𝒯, h_i)
7:              if L_i = ⊥
8:                  return 𝒞𝒫                   ▷ Stall: Data Unavailable
9:              else if 𝒞𝒫 ⪯ L_i ∧ ep(L_i) = ep
10:                 𝒞𝒫 ← L_i                    ▷ Update 𝒞𝒫.
11:                 if ISLAST(L_i)
12:                     ep ← ep + 1
13:                     actval ← GETACTVALS(L_i, ep)
14:                 end if
15:             end if
16:         end if
17:     end for
18:     ℒ, chs ← 𝒞𝒫, GETCHILDREN(𝒯, 𝒞𝒫[−1])
19:     while |chs| = 1
20:         ℒ ← ℒ ‖ chs                         ▷ Add the new child to ℒ
21:         chs ← GETCHILDREN(𝒯, ch)
22:     end while
23:     return ℒ
24: end function
```

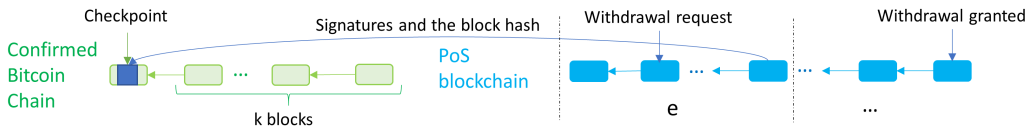

Fig. 3: There is an epoch e block containing a withdrawal request, and the hash of the last epoch e block and the signatures from the corresponding active validator set appear in a confirmed Bitcoin block in a client's view. The validator is granted permission to withdraw its stake once the Bitcoin block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

on its finalized blocks. Let h_i, $i \in [m]$, denote the sequence of checkpoints on Bitcoin, listed from the genesis to the tip of $\mathcal{C}^{\mathsf{c}}_r$, the confirmed Bitcoin chain in c's view at slot r. Starting at the genesis PoS block $B_0 = L_0$, c constructs a *checkpointed* chain $\mathcal{CP}^{\mathsf{c}}_r$ of finalized PoS blocks by sequentially going through the checkpoints. For $i = 1, \ldots, m$, let L_i denote the chain of PoS blocks ending at the block (denoted by B_i) at the preimage of the hash within h_i, if B_i and its prefix chain are available in c's view at slot r.

If a client observes a checkpoint h_i with epoch e on its Bitcoin chain at slot r, yet does not see by slot $r + \Delta$ all the block data of L_i and the respective validator signatures finalizing these blocks, then blocks on chain L_i are deemed to be *unavailable* and *not finalized* in the client's view.

Suppose c has gone through the sequence of checkpoints until h_j for some $j \in [m]$, and obtained the chain L ending at some block B from epoch e as the latest checkpointed chain based on the blocktree $\mathcal{T}$ and $h_1 \ldots, h_j$. Define $\tilde{e} = e + 1$ if B is the last block of its epoch; and $\tilde{e} = e$ otherwise. The checkpoint h_{j+1} is said to be *valid* (with respect to its prefix) if h_{j+1} contains $\tilde{e}$ as its epoch, and over $2n/3$ signatures on its block hash by the active validators of $\tilde{e}$ (Alg. 1, Line 5).

1) (Alg. 1, Line 9) If (i) h_{j+1} is valid, (ii) every block in L_{j+1} is available and finalized in c's view by the active validators of their respective epochs, (iii) the block B_{j+1} is from epoch $\tilde{e}$, and (iv) $L \preceq L_{j+1}$, then c sets L_{j+1} as the checkpointed chain.
2) **Emergency Break:** (Alg. 1, Line 7, Figure 2) If (i) h_{j+1} is valid, and (ii) a block in L_{j+1} is either unavailable or not finalized by its respective active validators in c's view, then c stops going through the sequence h_j, $j \in [m]$, and outputs L as its final checkpointed chain[4]. This premature stalling of the fork-choice rule is necessary to prevent the data availability attack described by Figure 2.
3) If both of the cases above fail, c skips h_{j+1} and moves to h_{j+2} and its preimage block as the next candidate.

Unless case (2) happens, c sifts through h_j, $j \in [m]$, and subsequently outputs the checkpointed chain $\mathcal{CP}^{\mathsf{c}}_r$. If (2) happens, c outputs L as both the checkpointed chain $\mathcal{CP}^{\mathsf{c}}_r$ and the canonical PoS chain $\mathcal{L}^{\mathsf{c}}_r$. Moreover, after waiting for a period of 2Δ, c sends a checkpoint transaction to Bitcoin for the block at the tip of $\mathcal{L}^{\mathsf{c}}_r$ to enforce slashable safety (*cf.* proof of Theorem 2). However, if c had previously outputted finalized blocks extending $\mathcal{CP}^{\mathsf{c}}_r$ in its canonical PoS chain, it does not roll back these blocks and freezes its old chain.

Finally, suppose c outputs a checkpointed chain with some block B at its tip. Then, starting at B, c traverses a path to the leaves of the blocktree (Alg. 1, Line 18 onward). If there is a single chain from B to a leaf, c outputs the leaf and its prefix chain as the PoS chain $\mathcal{L}^{\mathsf{c}}_r$. Otherwise, c identifies the highest PoS block B' (potentially the same as B) in the subtree of B, which has two or more siblings, and outputs B' and its prefix chain as $\mathcal{L}^{\mathsf{c}}_r$. Since c attaches the latest finalized PoS blocks to the tip of its PoS chain $\mathcal{L}^{\mathsf{c}}_r$, as long as there are no forks among finalized PoS blocks, the latency for transactions to enter $\mathcal{L}^{\mathsf{c}}_r$ matches the latency of the PoS protocol, hence rendering *fast finality* to the protocol.

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

1) The withdrawal request appears in a PoS block B in c's checkpointed chain $\mathcal{CP}^{\mathsf{c}}_r$.
2) Checkpoint of B or one of its descendants appears in a Bitcoin block that is at least k deep in $\mathcal{C}^{\mathsf{c}}_r$.
3) The client has not observed any fraud proof in $\mathcal{C}^{\mathsf{c}}_r$ that identifies v as a protocol violator. Similarly, the client has not observed checkpoints for conflicting PoS blocks in $\mathcal{C}^{\mathsf{c}}_r$, that implicates v in a protocol violation.

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

d) Slashing: Suppose a validator v has provably violated the protocol rules. Then, the contract on the PoS chain slashes v's locked coins upon receiving a fraud proof incriminating v if the PoS chain is live and v has not withdrawn its stake.

[4]The client c knows the active validator set for all epochs $e \leq \tilde{e}$. Every block in its checkpointed chain L is available in c's view. If B is the last block of epoch e, c can infer the active validator set of epoch $e + 1$ from L.

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

D. Security analysis

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

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

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

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

Proof sketches for Theorems 2 and 3 are given in Appendix B, and the full proofs are in Appendix E.

V. Optimal liveness

Babylon 1.0 provides slashable safety to Tendermint. However, the protocol guarantees liveness only when over $2/3$ of the active validators is honest. We next explore how Babylon 1.0 can be improved to achieve optimal liveness.

A. No liveness beyond $1/2$ adversarial fraction

Our first result is that no PoS protocol can achieve a liveness resilience of $f_{\text{l}} \geq n/2$ even with the help of Bitcoin, or for that matter any timestamping service, unless the entirety of the PoS blocks are uploaded to the service.

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

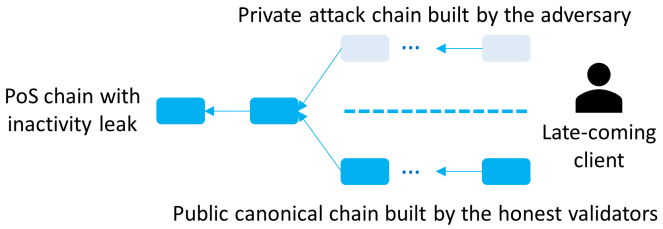

Fig. 4: Inactivity leak attack. Due to inactivity leak, honest and adversarial validators lose their stake on the attack and canonical chains respectively. A late-coming client cannot differentiate the canonical and attack chains.

Theorem 4. *Consider a PoS or permissioned protocol with n validators in a Δ synchronous network such that the protocol provides f_{s}-accountable-safety for some $f_{\text{s}} > 0$, and has access to a timestamping service. Suppose each validator is given an externally valid input of m bits by the environment $\mathcal{Z}$, but the number of bits written to the timestamping service is less than $m\lfloor n/2 \rfloor - 1$. Then, the protocol cannot provide f_{l}-T_{fin}-liveness for any $f_{\text{l}} \geq n/2$ and $T_{\text{fin}} < \infty$.*

To give intuition for the theorem, we analyze *inactivity leak*, used by Cosmos zones [9] and proposed for PoS Ethereum [23] to slash inactive validators and recover from liveness attacks. We show that when the adversary controls half of the validators, it can do a safety attack, where inactivity leak results in a gradual slashing of the *honest* validators' stake in the view of *late-coming* clients.

Consider PoS Ethereum with the setup on Figure 4. Half of the validators are adversarial, and build an attack chain that is initially kept private. They do not communicate with the honest validators or vote for the blocks on the public, canonical chain. As the honest validators are not privy to the adversary's actions, they also cannot vote for the blocks on the attack chain. Since only half of the validators are voting for the public blocks, liveness of the finality gadget, Casper FFG, is temporarily violated for the canonical chain. At this point, inactivity leak kicks in, and gradually slashes the stake of the adversarial validators on the canonical chain to recover liveness. Similarly, the honest validators lose their stake on the private attack chain due to inactivity leak. Finally, the adversary publishes the attack chain, which is subsequently adopted by a late-coming client. As there are conflicting chains in different clients' views, this is a safety violation, and by accountable safety, the late-coming client must identify at least one validator as a protocol violator. Since the client could not have observed the attack in progress, it cannot distinguish the attack chain from the canonical one. As a result, with non-negligible probability, it identifies the honest validators on the canonical chain as protocol violators. This contradicts with the accountability guarantee.

In the attack above, the data-limited timestamping service cannot help the late-coming client distinguish between the canonical and attack chains. The honest validators cannot timestamp the entirety of public blocks on a data-limited service. Thus, they cannot prove to a late-coming client that

the canonical chain was indeed public before the attack chain was published. This enables the adversary to plausibly claim to a late-coming client that the canonical chain was initially private, and its private attack chain was the public one.

The proof of Theorem 4 is given in Appendix D, and generalizes the attack on inactivity leak to any PoS or permissioned protocol. It exploits the indistinguishability of two worlds with different honest and adversarial validator sets, when the adversary controls over half of the validators and the timestamping service is data-limited.

B. Optimal liveness: full Babylon protocol with fast finality

We now analyze whether Bitcoin, despite its data limitation, can help increase the liveness resilience of Babylon 1.0 presented in Section IV-C. Babylon 1.0 provides $n/3$-slashable-safety and $n/3$-liveness; whereas Theorem 4 states that the liveness resilence of an accountably safe protocol cannot exceed $n/2$. We next close this gap and achieve the optimal liveness resilience of $n/2$ for the full Babylon protocol. Note that by [55, Appendix B], liveness resilience of a PoS protocol that provides $n/3$-accountable-safety cannot exceed $n/3$ *in the absence of external trust*, so the improvement of Babylon's resilience from $n/3$ to $n/2$ depends crucially on the use of the data-limited timestamping service. Indeed, if the adversary controls $f \in [n/3, n/2)$ of the active validators and violates liveness, the improved protocol uses Bitcoin as a fallback mechanism to guarantee eventual liveness.

The full Babylon protocol proceeds in two modes: the normal mode and the rollup mode, where Bitcoin plays a more direct role in the ordering of the PoS blocks. Execution starts and continues in the normal mode as long as no PoS transaction is censored. If a transaction is observed to be censored, clients can force the execution to switch to the rollup mode. During the normal mode, checkpointing of the PoS chain, fork-choice rule, stake withdrawals and slashing work in the same way as described in Section IV-C (*cf.* Alg. 2 for the complete fork-choice rule of the full Babylon protocol). Thus, we next focus on the rollup mode.

a) Checkpointing: If a transaction tx, input to an honest validator by the environment $\mathcal{Z}$ at slot r, has not appeared in an honest validator v's PoS chain within T_{tm} slots, *i.e.*, is not in $\mathcal{L}^{\mathsf{v}}_{r+T_{\mathrm{tm}}}$, v sends a *liveness transaction* to Bitcoin. Here, T_{tm} denotes the latency of Tendermint under synchrony. The liveness transaction contains the hash of the censored tx, and signals a liveness violation in v's view.

Suppose there is a block b within an honest validator v's Bitcoin chain, containing a liveness transaction for some censored tx. Upon observing b become k deep, v sends a checkpoint transaction for its PoS chain, even if it has not reached the last block of its epoch. If tx is not in its checkpointed chain, v also stops executing Tendermint. When b becomes $2k$ deep in its Bitcoin chain, if tx is still not in v's checkpointed chain, v enters the *rollup mode*. If tx appears in v's checkpointed chain before b becomes $2k$ deep in its Bitcoin chain, then v resumes its execution of Tendermint. Honest validators agree on when

Algorithm 2 The function used by a bootstrapping client c to find the canonical PoS chain $\mathcal{L}^{\mathsf{c}}_r$ at a given slot r. Here, t_i denotes the type of the transaction on Bitcoin, which can either be a checkpoint transaction, liveness transaction or a bundle. If t_i is a checkpoint transaction or bundle, h_i denotes the checkpoint, whereas if t_i is a liveness transaction, h_i denotes the hash of the censored transaction itself. The functions are defined in the caption of Algorithm 1 except for GETHEIGHT, which returns the height of the Bitcoin block containing the given checkpoint, and ISVALB, which checks the validity of the bundle checkpoint. The Boolean variables rmode and censor become True if the checkpoints are associated with a rollup mode or the grace period respectively. The variable censortx denotes the censored transaction. The notation $a \in c$ is a Boolean, which returns True iff the chain c contains a transaction whose hash is a.

function OUTPUTPOSCHAIN($\mathcal{T}$, $\mathcal{C}$)
  $(t_1, h_1), \ldots, (t_m, h_m) \leftarrow$ GETCKPTS($\mathcal{C}$)
  $\mathcal{CP}$, ep, actval $\leftarrow L_0, 1,$ GETACTVALS$(L_0, 1)$
  rmode, censor, censortx, ht $\leftarrow$ False, False, $\perp, -1$
  for $i = 1, \ldots, m$
    if censor $\wedge$ GETHEIGHT$(h_i) \geq$ ht $+ 2k$
      ▷ *Enter rollup mode*
      rmode, censor, censortx $\leftarrow$ True, False, $\perp$
    else if rmode $\wedge$ GETHEIGHT$(h_i) \geq$ ht $+ 2k + T_{\mathrm{btc}}$
      ▷ *Exit rollup mode*
      rmode, ht $\leftarrow$ False, -1
    end if
    ▷ *Obtain the checkpointed chain*
    if $t_i =$ checkpoint $\wedge \neg$rmode $\wedge$ ISVALID$(h_i,$ actval, ep)
      ▷ *Normal mode*
      $L_i \leftarrow$ GETCHAIN$(\mathcal{T}, h_i)$
      if $L_i = \perp$
        return $\mathcal{CP}$ ▷ *Stall: Data Unavailable*
      else if $\mathcal{CP} \preceq L_i \wedge \mathrm{ep}(L_i) =$ ep
        $\mathcal{CP} \leftarrow L_i$
        if ISLAST(L_i)
          ep, actval $\leftarrow$ ep $+ 1,$ GETACTVALS$(\mathrm{L_i},$ ep$)$
        end if
        if censor $\wedge$ (censortx $\in \mathcal{CP}$)
          censor, censortx, ht $\leftarrow$ False, $\perp, -1$
        end if
      end if
    else if $t_i =$ liveness $\wedge \neg$rmode ▷ *Liveness transaction*
      if ($\mathsf{tx} \notin \mathcal{CP}$) $\wedge \neg$censor
        censor, censortx $\leftarrow$ True, $\{\mathrm{h_i}\}$
        ht $\leftarrow$ GETHEIGHT(h_i)
      else if ($\mathsf{tx} \notin \mathcal{CP}$) $\wedge$ censor
        censortx $\leftarrow$ censortx $\cup \{h_i\}$
      end if
    else if $t_i =$ bundle $\wedge$ rmode $\wedge$ ISVALB$(h_i,$ actval, ep$)$
      ▷ *Rollup mode*
      $B_i \leftarrow$ GETCHAIN$(\mathcal{T}, h_i)$
      if $B_i = \perp$
        return $\mathcal{CP}$ ▷ *Stall: Data Unavailable*
      else
        $\mathcal{CP} \leftarrow \mathcal{CP} \parallel B_i$ ▷ *Attach B_i to $\mathcal{CP}$*
      end if
    end if
  end for
  $\mathcal{L}$, chs $\leftarrow \mathcal{CP},$ GETCHILDREN$(\mathcal{T}, \mathcal{CP}[-1])$
  while $|\mathrm{chs}| = 1$
    if rmode $\vee$ (censor $\wedge\ |\mathcal{C}| \geq$ ht $+ k$)
      Break
    end if
    $\mathcal{L}$, chs $\leftarrow \mathcal{L} \parallel$ chs, GETCHILDREN$(\mathcal{T},$ chs$)$
  end while
  return $\mathcal{L}$
end function

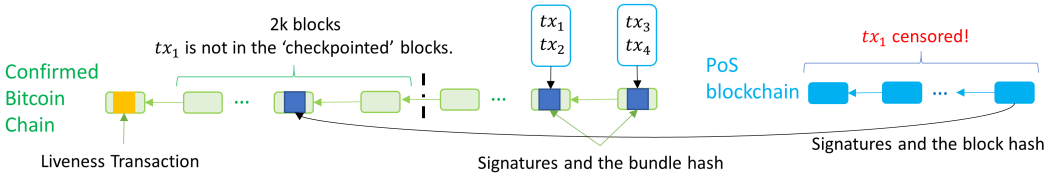

Fig. 5: If tx_1 is observed to be censored by an honest validator v, it sends a *liveness transaction* to Bitcoin. Once the liveness transaction becomes $2k$ deep in v's and the clients' views, they enter the rollup mode. In the rollup mode, validators group transactions into bundles and post signed hashes of these bundles to Bitcoin.

to halt or resume the protocol execution as these decisions are based on the checkpoints on Bitcoin.

Once in the rollup mode, each honest validator collects transactions into *bundles* that are broadcast to all other validators. Upon observing a bundle of externally valid transactions, honest validators sign the hash of the bundle. If an honest validator observes a bundle whose hash has been signed by over $n/2$ validators, it sends a checkpoint transaction containing the hash of the bundle, the last epoch number, and the signatures to Bitcoin. We refer to the hash, epoch and the quorum of signatures collectively as a bundle checkpoint.

b) Fork-choice rule (Figure 5, Alg. 2): Consider a client c that observes the protocol at some slot $r' \geq r + T_{\text{tm}}$. Suppose there is a block b within $\mathcal{C}^{\mathsf{c}}_{r'}$, containing a liveness transaction for some censored tx. Once b becomes k deep in c's Bitcoin chain, if tx is still not in c's checkpointed chain, c freezes its PoS chain. At this point, c also sends a checkpoint transaction for its PoS chain, even if it has not reached the last block of its epoch. Afterwards, c outputs new PoS blocks as part of its PoS chain, only if these new blocks are also part of its checkpointed chain (Alg. 2, Line 47). Note that if c was previously awake and has already outputted PoS blocks extending its checkpointed chain when the PoS chain was frozen, it does not roll back these blocks.

If c observes tx within its checkpointed chain by the time b becomes $2k$ deep, it resumes outputting new blocks as part of its PoS chain, and continues the protocol execution in the normal mode(Alg. 2, Line 24). Otherwise, once b becomes $2k$ deep in c's Bitcoin chain, if tx is not yet in c's checkpointed chain, it enters the *rollup mode* (Figure 5, Alg. 2, Line 6).

Once in the rollup mode, c first constructs the checkpointed chain by observing the prefix of its Bitcoin chain that ends at the $2k^{\text{th}}$ block extending b. Suppose B from epoch e is the last PoS block appended to the checkpointed chain, and it is followed by a sequence h_j, $j \in [m]$,of bundle checkpoints in c's Bitcoin chain. Let $\tilde{e} = e + 1$ if B is the last block of epoch e, and $\tilde{e} = e$ otherwise. The bundle checkpoint h_j is said to be *valid* (with respect to its prefix) if h_j contains $\tilde{e}$ as its epoch, and over $n/2$ signatures on its block hash by the active validators of $\tilde{e}$ (Alg. 2, Line 35). After constructing the checkpointed chain, c sifts through h_j, $j \in [m]$, iteratively:

1) (Alg. 2, Line 40) If (i) h_j is valid, and (ii) its preimage bundle is available in c's view, then c attaches the bundle to its checkpointed chain. During the rollup mode, clients output the same chain as the checkpointed and PoS chains.
2) (Alg. 2, Line 38) If (i) h_j is valid, and (ii) its preimage bundle is not available in c's view, then c stops going through the sequence h_j, $j \in [m]$, and returns its current checkpointed and PoS chains.
3) If neither of the conditions above are satisfied, c skips h_j, and moves to h_{j+1} as the next candidate.

The client c leaves the rollup mode when it sees the $T^{\text{th}}_{\text{btc}}$ Bitcoin block extending b (*cf.* Alg. 2 Line 9). Here, T_{btc} is a protocol parameter for the duration of the rollup mode. After exiting the rollup mode, validators treat the hash of the last valid bundle checkpoint as the parent hash for the new PoS blocks, and execute the protocol in the normal mode. It is possible to support stake withdrawals during the rollup mode, once the bundles containing withdrawal requests are timestamped by k deep valid bundle checkpoints in Bitcoin.

C. Slashing and liveness after a safety violation

Theorem 2 and its analogue below states that if Bitcoin is secure, at least $n/3$ adversarial validators become slashable in the view of all clients when there is a safety violation. However, the theorems do not specify whether these validators can be *slashed* at all. Indeed, slashing can only be done if the PoS chain is live, a condition that might not be true after a safety violation.

When blocks on conflicting chains are finalized in Tendermint, the chain with the earlier checkpoint in Bitcoin is chosen as the canonical one. However, the honest active validators might not be able to unlock from the conflicting chains they have previously signed and start extending the canonical one, as that would require them to sign conflicting blocks. Due to the absence of signatures from these stuck validators, the PoS chain might stall after a safety violation. Then, even though the validators that have caused the safety violation become slashable, the on-chain contract cannot slash them since the chain itself is not live.

This issue of liveness recovery after a safety violation is present in many BFT protocols, which strive to support accountability. For example, Cosmos chains enter into a panic state when safety is violated, and need a manual reboot based on social consensus with slashing done off chain [10]. With Bitcoin, however, this problem can be solved to an extent. As in the case of stalling or censorship, after which the protocol switches to the rollup mode, the honest validators can use Bitcoin to unstuck from their respective forks, bootstrap the

PoS chain, and slash the adversarial validators, if the honest validators constitute over *half* of the active validator set. They can use the same process described in Section V-B for entering the rollup mode: Once the PoS chain loses liveness, an honest validator posts a liveness transaction to Bitcoin for the censored PoS transactions. Soon afterwards, the honest PoS validators enter the rollup mode. This is because new checkpoints appearing on Bitcoin and signed by the slashable validators will not be considered valid by the honest validators, and cannot prevent the protocol from switching to the rollup mode. Once in the rollup mode, with their majority, the honest PoS validators can sign new bundles, and send their checkpoints to Bitcoin. Through these new bundles, they can finalize the censored transactions, and slash the adversarial validators that have previously become slashable.

After $n/3$ adversarial active validators are slashed, the remaining $n/2$ honest validators constitute a supermajority of the active validator set. By treating the last valid bundle checkpoint as the new genesis block, they can switch back to the normal mode, and continue finalizing new PoS blocks through Tendermint (Alg. 2, Line 9). Thus, the PoS chain can bootstrap liveness, and eventually return to the normal mode with fast finality after the safety violation.

D. Analysis

Theorem 5 (Slashable Safety)**.** *Suppose Bitcoin is secure with parameter k with overwhelming probability, and there is one honest active validator at all times. Then, the Babylon protocol (Section V-B) with fast finality satisfies $n/3$-slashable safety with overwhelming probability.*

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

Proof sketches for Theorems 5 and 6 are given in Appendix B, and the full proofs are in Appendix E.

VI. Experiments

Babylon allows a validator to withdraw its stake once the checkpoint of a PoS chain containing the withdrawal request becomes k deep in the confirmed Bitcoin chain. This implies a withdrawal delay on the order of Bitcoin's confirmation latency for the checkpoints, as opposed to the week long periods imposed by many PoS chains (*cf.* Section II). To measure the reduction in the withdrawal delay and to demonstrate the feasibility of using Bitcoin for checkpointing, we implemented a prototype *checkpointer* for Tendermint, the consensus protocol of Cosmos zones. We sent periodic checkpoints to the Bitcoin Mainnet via the checkpointer to emulate the operation of Babylon 1.0 securing a Cosmos zone and measured the confirmation times of the checkpoints.

A. Checkpoints

Our checkpointer utilizes Bitcoin's OP_RETURN opcode, which allows 80 bytes of arbitrary data to be recorded in an unspendable transaction [8] (*cf.* Bartoletti and Pompianu for an empirical study of the data within OP_RETURN transactions [22]). Checkpoints consist of an epoch number, the hash of a PoS block, and signatures by $2/3$ of the active validators of the epoch. In Tendermint, each pre-commit signature is of 64 bytes [5], whereas prominent Cosmos zones such as Osmosis and Cosmos Hub feature 100 to 175 validators [15], [12]. However, if the checkpoints include signatures by 67 validators, a single checkpoint would be approximately 4.3 kBytes, requiring at least 54 separate Bitcoin transactions to record the information within *one* checkpoint.

To reduce the number of transactions needed per checkpoint, Cosmos zones can adopt Schnorr [54] or BLS [24] signatures that allow signature aggregation. In our experiments, we consider a Cosmos zone that uses BLS signatures. To construct a checkpoint, we first generate 32 bytes representing the block hash as specified by Tendermint's LastCommitHash [5], and 67 BLS signatures on this value, each of 48 bytes. They emulate the pre-commit signatures by a $2/3$ quorum among 100 validators. We then aggregate these 67 BLS signatures into a single one. Thus, the checkpoints contain (i) an epoch number of 8 bytes, (ii) a block hash of 32 bytes, (iii) an aggregate BLS signature of 48 bytes, and (iv) a bit map of 13 bytes to help clients identify which validators' signatures were aggregated. This implies a total size of 111 bytes per checkpoint, reducing the number of transactions needed per checkpoint from 54 to 2. In each of the two transactions, the OP_RETURN also includes a fixed tag of 4 bytes for the letters 'BBNT' to signal the Babylon checkpoints. This makes the total size of the two Bitcoin transactions 195 and 183 bytes respectively including other parts of the transactions.

B. Measurements

The withdrawal delay consists of two components: (i) the time to send the first checkpoint for a PoS chain including the withdrawal request, and (ii) the time for the checkpoint to become confirmed, *i.e.*, k deep in Bitcoin, after it is sent. The first component equals the epoch length as a checkpoint is sent at the end of every epoch by an honest validator, and the time to prepare the checkpoint transactions is negligible compared to the epoch length. This is because it takes 0.143 ms to generate one BLS signature, and 0.042 ms to aggregate 67 BLS signatures on an Apple Macbook Pro M1 machine using the codebase [4].

To estimate the withdrawal delay of a Tendermint based PoS protocol using Babylon 1.0, we sent hourly checkpoints over the course of one day[5], for an epoch length of one hour. At every hour, *two* pairs of OP_RETURN transactions for two identical checkpoints were broadcast to the Bitcoin network: one pair sent with a miner fee of 12 Satoshis/byte, and the

[5]Experiment was started at 1 am August 18, 2022, and run until 1 am August 19 (PT).

other sent with a fee of 3 Satoshis/byte. Given the transaction sizes, and Bitcoin's token price[6], this implies a total cost of 1.07 USD *per checkpoint* at the fee level of 12, and 0.27 USD per checkpoint at the level of 3 Satoshis/byte. The fee levels were determined before the experiments with the help of a Bitcoin transaction fee estimator [3], which displayed 12 and 3 Satoshis as the fee levels to have the transaction included within the next block, and the next 6 blocks, respectively.

For each checkpoint associated with a different fee, we measured (i) the time $T_{k=6}$ for both transactions to be at least $k = 6$ deep, and (ii) the time $T_{k=20}$ for both to be at least $k = 20$ deep. The confirmation depth $k = 6$ was chosen following the conventional rule for Bitcoin. The depth $k = 20$ implies a low probability (10^{-7}) of safety violation for the blocks containing the checkpoints [43, Figure 2][7]. We report the mean and standard deviation for these parameters across the 24 checkpoints in Table II[8].

Checkpointing cost per annum	$T_{k=6}$ (mins)	$T_{k=20}$ (mins)
9373 USD	69.7 ± 20.7	192.7 ± 26.1
2365 USD	77.0 ± 21.2	204.1 ± 31.2

TABLE II: Mean and standard deviation for $T_{k=6}$ and $T_{k=20}$ for the checkpoints sent over a day at two different fee levels.

Table II shows that at an annual cost of less than $10,000$ USD, a PoS chain using Babylon can reduce its withdrawal delay from weeks to below 4 hours. The small cost of checkpointing is achieved via aggregate signatures, in the absence of which the cost would be on the order of millions.

C. Cost of the full Babylon protocol

To estimate the additional cost of checkpointing during the rollup mode, we consider a liveness transaction for some tx. The transaction uses the OP_RETURN opcode to record the hash of tx. Upon observing the liveness transaction on Bitcoin, clients send checkpoints to Bitcoin for their canonical PoS chains. To avoid duplicate transactions for the same tx and duplicate checkpoints for the same PoS chain, a special validator can be tasked with posting liveness transactions, collecting PoS blocks from the clients and posting the checkpoint of the longest canonical PoS chain. Acting as watch towers, clients observe the posted checkpoints and send checkpoints of higher or conflicting PoS blocks if the validator fails. The number of checkpoints on Bitcoin before the rollup mode can thus be bounded while the special validator remains functioning.

Once in the rollup mode, honest validators periodically send bundle checkpoints to Bitcoin. These checkpoints have the same structure as the checkpoints of PoS blocks. The period of bundle checkpoints determine the cost of checkpointing during the rollup mode. Frequent checkpoints reduce latency down to Bitcoin latency, whereas infrequent checkpoints result in a smaller cost. The honest validators can designate a leader in round-robin fashion to create a bundle every hour, thus send one checkpoint per hour. This makes the cost of checkpointing in the rollup mode equal to the normal mode.

D. Checkpoint verification by light nodes

In Sections IV-C and V-B, clients download all block data from Bitcoin to obtain the complete sequence of checkpoints. To detect checkpoints, they check for the 'BBNT' tag at the beginning of the OP_RETURN data. Although any client can send a Bitcoin transaction with the 'BBNT' tag, transaction fees can deter spamming attacks, and miners can reject bursts of OP_RETURN transactions. An alternative solution is to leverage Bitcoin's Taproot upgrade to prevent a minority adversary from posting invalid checkpoints [19].

Clients also run a Bitcoin light node to avoid downloading invalid checkpoints. Light nodes download block headers to find the confirmed Bitcoin chain and rely on full nodes to receive the checkpoints along with their (Merkle) inclusion proofs that are verified against the transaction roots in the headers [44]. However, as the validity of checkpoints depends on the preceding ones and Bitcoin does not verify their validity, clients cannot validate a checkpoint based solely on its inclusion in a Bitcoin block. Noticing that Bitcoin acts as a *lazy blockchain* towards checkpoints, we can use insights from light nodes of lazy blockchains to help clients identify the canonical PoS chain without downloading all checkpoints and validating them sequentially [58].

VII. Babylon with slow finality: Bitcoin safety

So far in the paper, we have focused on the scenario where the clients of the PoS chain use the native *fast finality rule*, where blocks are finalized upon gathering signatures from the PoS validators. Since Bitcoin confirmation operates at a slower time scale, Bitcoin cannot protect the PoS chain against safety attacks under the fast finality rule. Bitcoin instead makes these attacks *slashable* by not allowing the attackers to withdraw stake after signing conflicting blocks. However, clients might want Bitcoin level safety for important transactions or during the bootstrapping phase of a PoS chain, when slashability is not sufficient. For this purpose, clients can choose to use a *slow finality rule*, which requires a PoS block to be checkpointed by Bitcoin in addition to its finalization on the PoS chain. More specifically, a client c using the slow finality rule sets its PoS chain to be the same as its checkpointed chain at any time slot: $\mathcal{L}^{\mathsf{c}}_r = \mathcal{CP}^{\mathsf{c}}_r$. A major drawback of this scheme is its lateny: client now waits until the checkpoints are k deep in Bitcoin, before it can output their blocks as its PoS chain.

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

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

[6] 1 BTC is taken as $23,563.83$ USD, the maximum Bitcoin price observed on August 18, 2022 [2].

[7] This is for an adversary controlling 10% of the hash rate, given a network delay of 10 seconds for the Bitcoin network

[8] The data and the exact OP_RETURN bytes can be accessed at https://github.com/gitferry/Babylon-checkpoints.

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

Proof sketch for Corollary 2 is given in Appendix B, and the full proof is in Appendix E.

ACKNOWLEDGEMENTS

We thank Shresth Agrawal, Kamilla Nazirkhanova, Joachim Neu, Lei Yang and Dionysis Zindros for several insightful discussions on this project. We thank Dionysis Zindros also for his help with the experiments.

Ertem Nusret Tas is supported by the Stanford Center for Blockchain Research, and did part of this work as part of an internship at BabylonChain Inc. David Tse is a co-founder of BabylonChain Inc.

REFERENCES

[1] BabylonChain. https://babylonchain.io/.
[2] Bitcoin historical data. https://coinmarketcap.com/currencies/bitcoin/historical-data/. Accessed: 2022-08-18.
[3] Bitcoin Transaction Fee Estimator & Calculator. https://coinmarketcap.com/.
[4] blst. https://github.com/supranational/blst.
[5] Data structures. https://docs.tendermint.com/master/spec/core/data_structures.html.
[6] Glossary. https://docs.osmosis.zone/overview/terminology/.
[7] Komodo. Advanced blockchain technology, focused on freedom. https://docs.komodoplatform.com/whitepaper/introduction.html.
[8] Op_return. https://en.bitcoin.it/wiki/OP_RETURN.
[9] Running a validator. https://hub.cosmos.network/main/validators/validator-setup.html.
[10] Running in production. https://docs.tendermint.com/v0.34/tendermint-core/running-in-production.html.
[11] The Pikachu RFP: Checkpointing Filecoin onto Bitcoin. https://research.protocol.ai/blog/2022/the-pikachu-rfp-checkpointing-filecoin-onto-bitcoin/.
[12] Validators overview. https://hub.cosmos.network/main/validators/overview.html.
[13] What is staking? https://docs.avax.network/nodes/validate/staking.
[14] Launch communications — june community update. https://blog.cosmos.network/launch-communications-june-community-update-e1b29d66338, 2018.
[15] Guide to osmosis. https://www.coinbase.com/cloud/discover/protocol-guides/guide-to-osmosis, 2022.
[16] VDF Alliance. VDF Alliance FPGA Competition. https://supranational.atlassian.net/wiki/spaces/VA/pages/36569208/FPGA+Competition, 2019.
[17] Aditya Asgaonkar. Weak Subjectivity in Eth2.0. https://notes.ethereum.org/@adiasg/weak-subjectvity-eth2, 2019.
[18] Sarah Azouvi, George Danezis, and Valeria Nikolaenko. Winkle: Foiling long-range attacks in proof-of-stake systems. In *AFT*, pages 189–201. ACM, 2020.
[19] Sarah Azouvi and Marko Vukolic. Pikachu: Securing pos blockchains from long-range attacks by checkpointing into bitcoin pow using taproot. *arXiv:2208.05408*, 2022.
[20] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros Genesis: Composable proof-of-stake blockchains with dynamic availability. In *CCS*, pages 913–930. ACM, 2018.
[21] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better - how to make bitcoin a better currency. In *Financial Cryptography*, volume 7397 of *Lecture Notes in Computer Science*, pages 399–414. Springer, 2012.
[22] Massimo Bartoletti and Livio Pompianu. An analysis of bitcoin op_return metadata. In *Financial Cryptography Workshops*, volume 10323 of *Lecture Notes in Computer Science*, pages 218–230. Springer, 2017.
[23] Carl Beekhuizen. Validated, staking on eth2: #1 - Incentives. https://blog.ethereum.org/2020/01/13/validated-staking-on-eth2-1-incentives/, 2020.
[24] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.
[25] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains, 2016.
[26] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *arXiv:1807.04938*, 2018.
[27] Vitalik Buterin. Proof of stake: How i learned to love weak subjectivity. https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/, 2014.
[28] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv:1710.09437*, 2019.
[29] 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.
[30] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186. USENIX Association, 1999.
[31] Benjamin Y. Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains. In *AFT*, pages 1–11. ACM, 2020.
[32] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.
[33] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography*, volume 11598 of *Lecture Notes in Computer Science*, pages 23–41. Springer, 2019.
[34] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT 2018*, pages 66–98. Springer, 2018.
[35] Christian Decker and Rusty Russell. eltoo : A simple layer 2 protocol for bitcoin. 2018.
[36] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *SSS*, volume 9212 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015.
[37] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725, 2019.
[38] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.
[39] Bela Gipp, Norman Meuschke, and Andre Gernandt. Decentralized trusted timestamping using the crypto currency bitcoin. In *Proceedings of the iConference 2015*, 2015.
[40] Thomas Hepp, Patrick Wortner, Alexander Schönhals, and Bela Gipp. Securing physical assets on the blockchain: Linking a novel object identification concept with distributed ledgers. In *CRYBLOCK@MobiSys*, pages 60–65. ACM, 2018.
[41] Dimitris Karakostas and Aggelos Kiayias. Securing proof-of-work ledgers via checkpointing. In *IEEE ICBC*, pages 1–5. IEEE, 2021.
[42] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388. Springer, 2017.
[43] Jing Li, Dongning Guo, and Ling Ren. Close latency-security trade-off for the nakamoto consensus. In *AFT*, pages 100–113. ACM, 2021.
[44] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.
[45] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *IEEE Symposium on Security and Privacy*, pages 446–465. IEEE, 2021.
[46] 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.
[47] Daejun Park and Aditya Asgaonkar. Analysis on weak subjectivity in ethereum 2.0. https://github.com/runtimeverification/beacon-chain-verification/blob/master/weak-subjectivity/weak-subjectivity-analysis.pdf, 2021.

[48] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, volume 91 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
[49] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *ASIACRYPT 2017*, pages 380–409. Springer, 2017.
[50] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2018.
[51] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2016.
[52] Maxwell Sanchez and Justin Fisher. Proof-of-proof: A decentralized, trustless, transparent, and scalable means of inheriting proof-of-work security. https://veriblock.org/wp-content/uploads/2018/03/PoP-White-Paper.pdf, 2018.
[53] Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. Blockchain CAP theorem allows user-dependent adaptivity and finality. In *Financial Cryptography (2)*, volume 12675 of *Lecture Notes in Computer Science*, pages 84–103. Springer, 2021.
[54] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.
[55] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. BFT protocol forensics. In *CCS*, pages 1722–1743. ACM, 2021.
[56] Selma Steinhoff, Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolic. BMS: Secure Decentralized Reconfiguration for Blockchain and BFT Systems. *arXiv:2109.03913*, 2021.
[57] Alistair Stewart and Eleftherios Kokoris-Kogia. GRANDPA: A Byzantine finality gadget. *arXiv:2007.01560*, 2020.
[58] Ertem Nusret Tas, Dionysis Zindros, Lei Yang, and David Tse. Light clients for lazy blockchains. *IACR Cryptol. ePrint Arch.*, page 384, 2022.
[59] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0.8.13. https://solana.com/solana-whitepaper.pdf, 2019.
[60] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356. ACM, 2019.

APPENDIX A
COMPARISON OF BABYLON WITH BITCOIN IMPROVEMENT PROPOSALS AND PAYMENT CHANNELS

Payment channels is a layer-2 scalability solution on Bitcoin that enables processing payments among multiple parties without using Bitcoin except in the case of adversarial participants [36], [51], [35]. Unlike payment channels, Babylon uses Bitcoin to order checkpoints from PoS chains, while the transaction logic lives solely on the PoS chain. Any existing PoS chain can opt in to use Babylon, with no change to Bitcoin. Thus, Babylon preserves the responsiveness (fast finality) and high throughput of PoS chains without any upgrade to Bitcoin. In contrast, BIPs for offline transactions require changes to Bitcoin and cannot directly leverage off the properties of existing protocols.

APPENDIX B
PROOF SKETCHES FOR THE SECURITY THEOREMS

A. Proof Sketch for Theorem 2

By the $n/3$-accountable safety of the PoS protocol, at least $n/3$ adversarial validators are identified after a safety violation. These violations can only happen due to short-range forks at the tip of the clients' PoS chains, since the checkpointed chains do not conflict with each other by Proposition 2. As the conflicting blocks within the short-range forks are not checkpointed by sufficiently deep Bitcoin blocks, the identified adversarial validators could not have withdrawn their stake in any clients' view. Thus, at least $n/3$ validators become slashable in all clients' views after the safety violation. The full proof of Theorem 2 is presented in Appendix E.

B. Proof Sketch for Theorem 3

When the number of adversarial active validators is less than $n/3$, the PoS protocol satisfies safety and T_{fin}-liveness. Then, no unavailable checkpoint gathers enough signatures to be valid, preventing the adversary from triggering the stalling condition (Line 7, Alg. 1). The full proof of Theorem 3 is presented in Appendix E.

C. Proof Sketch for Theorem 5

Since Bitcoin is responsible for ordering the bundles in the rollup mode and is assumed to be secure, there cannot be safety violations due to conflicting bundles. Thus, safety violations can only happen due to short-range forks with PoS blocks, in which case slashability of the adversarial validators follows from the proof of Theorem 2. The full proof of Theorem 5 is presented in Appendix E.

D. Proof Sketch for Theorem 6

When the number of adversarial active validators is less than $n/2$, the protocol satisfies liveness during the rollup mode, since the bundle checkpoints require signatures from half of the validators. Moreover, no unavailable checkpoint gathers enough signatures to be valid, preventing the adversary from triggering the stalling condition (Lines 17 and 38, Alg. 1). The full proof of Theorem 6 is presented in Appendix E.

E. Proof Sketch for Corollary 2

When the number of adversarial active validators is less than $n/2$, the PoS chains satisfy liveness by Theorem 6. In this case, after entering the PoS chains held by the clients, transactions soon appear in the checkpointed chains as the PoS chains are checkpointed regularly, implying the liveness of the protocol with slow finality. Proof is given in Appendix E.

APPENDIX C
PROOF OF THEOREM 1

Proof. Proof is by contradiction based on an indistinguishability argument between two worlds. Towards contradiction, suppose there exists a PoS protocol Π that provides f_{l}-T_{fin}-liveness and f_{a}-slashable-safety for some integers $f_{\text{l}}, f_{\text{a}} > 0$ and $T_{\text{fin}} < \infty$.

Let n be the number of active validators at any given slot. Let P_0, P, Q' and Q'' denote disjoint sets of validators: $P_0 = \{\mathsf{v}_i^0, i = 1, .., n\}$, $P = \{\mathsf{v}_i, i = 1, .., n\}$, $Q' = \{\mathsf{v}_i', i = 1, .., n\}$ and $Q'' = \{\mathsf{v}_i'', i = 1, .., n\}$. Let $T < \infty$ denote the time it takes for a validator to withdraw its stake after its withdrawal transaction is finalized by the PoS protocol.

The initial set of active validators is P_0. The environment inputs transactions to the active validators so that the active validator set changes over time. Suppose the active validator set becomes P at some slot t (*e.g.*, $P = P_0$ at $t = 0$). We then consider the following four worlds:

World 1: Validators in P and Q' are honest. At slot t, $\mathcal{Z}$ inputs transactions tx'_i, $i = 1, .., n$, to the validators in P. Here, tx'_i is the withdrawal transaction for v_i. Validators in P execute the PoS protocol, and record the consensus messages in their transcripts. The environment $\mathcal{Z}$ replaces v_i with $\mathsf{v}'_i \in Q'$ as the new active validator.

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

World 2: Validators in P and Q'' are honest. At slot t, $\mathcal{Z}$ inputs transactions tx''_i, $i = 1, .., n$, to the validators in P. Here, tx''_i is the withdrawal transaction for v_i. Validators in P execute the PoS protocol, and record the consensus messages in their transcripts. The environment $\mathcal{Z}$ replaces each v_i with $\mathsf{v}''_i \in Q''$ as the new active validator.

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

World 3: Validators in Q' are honest. Validators in P and Q'' are adversarial. At slot t, $\mathcal{Z}$ inputs transactions tx'_i, $i = 1, .., n$, to the validators in P. Validators in P execute the PoS protocol, and record the consensus messages they observe in their transcripts.

Simultaneous with the execution above, $(\mathcal{A}, \mathcal{Z})$ creates a simulated execution, where a different set of transactions, tx''_i, $i \in [n]$, are input to the validators in P at slot t. In the simulated execution, $\mathcal{Z}$ replaces each v_i with $\mathsf{v}''_i \in Q''$ as the new active validator. As in the real execution, validators in P record the consensus messages of the simulated execution in their transcripts.

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

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

World 4: World 4 is the same as world 3, except that the validators in Q' are adversarial, those in Q'' are honest, and the real and simulated executions are run with the transactions tx''_i and tx'_i respectively. Given the messages received by the clients c_1 and c_2 from the validators, the worlds 3 and 4 are indistinguishable in their views except with negligible probability. Thus, again a validator from Q'' becomes slashable in the clients' views in world 4 with non-negligible probability. However, the validators in Q'' are honest in world 4, which is a contradiction with the definition of the forensic protocol. □

APPENDIX D
PROOF OF THEOREM 4

Proof. Proof is by contradiction based on an indistinguishability argument between two worlds. Towards contradiction, suppose there exists a permissioned (or PoS) protocol Π with access to a data-limited timestamping service, that provides f_{s}-accountable-safety, and f_{l}-T_{fin}-liveness for some integers $f_{\text{s}} > 0$, $f_{\text{l}} \geq n/2$, and $T_{\text{fin}} < \infty$. Let P and Q denote two sets that partition the validators into two groups of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ respectively. Consider the following worlds, where $\mathcal{Z}$ inputs externally valid bit strings, tx^P_i, $i \in [\lceil n/2 \rceil]$, and tx^Q_j, $j \in [\lfloor n/2 \rfloor]$, to the validators in P and Q respectively at the beginning of the execution. Here, each validator i in P receives the unique string tx^P_i, and each validator j in Q receives the unique string tx^Q_j. Each string consists of m bits, where m is a polynomial in the security parameter λ.

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

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

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

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

As $|P| = \lceil n/2 \rceil \leq f_{\text{l}}$, by the f_{l}-liveness of Π, clients c_1 and c_2 both output tx^Q_j, $j \in [\lfloor n/2 \rfloor]$, as part of their chains by slot T_{fin}. Since there can be at most $m\lfloor n/2 \rfloor - 1$ bits of data on I, and tx^P_i, $i \in [\lceil n/2 \rceil]$, consist of $m\lceil n/2 \rceil$ bits in total, c_1

does not learn and cannot output all of tx_i^P, $i \in [\lceil n/2 \rceil]$, as part of its chain by slot T_{fin}, with overwhelming probability

The worlds 1 and 2 are indistinguishable by c_2 in terms of the messages received from the validators and observed on the timestamping service, except with negligible probability. Thus, it outputs the same chain containing tx_i^P, $i \in [\lceil n/2 \rceil]$, and tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, in both worlds with overwhelming probability. However, c_1's chain contains tx_i^P, $i \in [\lceil n/2 \rceil]$, but not tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, in world 1, and tx_j^Q, $j \in [\lfloor n/2 \rfloor]$, but not tx_i^P, $i \in [\lceil n/2 \rceil]$, in world 2. This implies that there is a safety violation in either world 1 or world 2 or both worlds with non-negligible probability.

Without loss of generality, suppose there is a safety violation in world 2. In this case, an honest validator v asks the other validators for their transcripts, upon which the adversarial validators in P reply with transcripts that omit the messages received from the set Q. As $f_{\text{s}} > 0$, by invoking the forensic protocol with the transcripts received, v identifies a non-empty subset $S \subseteq P$ of the adversarial validators, and outputs a proof that the validators in S have violated the protocol Π. However, in this case, an adversarial validator in world 1 can emulate the behavior of v in world 2, and ask the validators for their transcripts. It can then invoke the forensic protocol with the transcripts, and output a proof that identifies the same subset $S \subseteq P$ of validators as protocol violators. Since the two worlds are indistinguishable by c_2 except with negligible probability, upon receiving this proof, it identifies the honest validators in $S \subseteq P$ as protocol violators in world 1 as well with non-negligible probability, which is a contradiction. By the same reasoning, if the safety violation happened in world 1, an adversarial validator in world 2 can construct a proof accusing an honest validator in world 2 in c_2's view with non-negligible probability, again a contradiction. □

Appendix E
Security Proofs

Proof of Proposition 2. Since Bitcoin is safe with parameter k, without loss of generality, suppose $\mathcal{C}_{r_1}^{\mathsf{c}_1} \preceq \mathcal{C}_{r_2}^{\mathsf{c}_2}$ for two clients c_1 and c_2 and time slots r_1 and r_2. Let h_i, $i \in [m_1]$, and h_j, $j \in [m_2]$, $m_1 \leq m_2$, denote the sequence of checkpoints in c_1's and c_2's views at slots r_1 and r_2 respectively. Note that the sequence observed by c_1 is a prefix of c_2's sequence.

Starting from the genesis block, let B_1 denote the first PoS block in $\mathcal{CP}_{r_1}^{\mathsf{c}_1}$ that is not available or finalized in c_2's view at slot r_2, and define i_1 as the index of the first valid checkpoint on $\mathcal{C}_{r_1}^{\mathsf{c}_1}$, whose block extends or is the same as B_1 on $\mathcal{CP}_{r_1}^{\mathsf{c}_1}$. (If there is no such block B_1, $i_1 = \infty$.) Similarly, let B_2 denote the first PoS block in $\mathcal{CP}_{r_2}^{\mathsf{c}_2}$ that is not available or finalized in c_1's view at slot r_1, and define i_2 as the index of the first valid checkpoint, whose block extends or is the same as B_2 on $\mathcal{CP}_{r_2}^{\mathsf{c}_2}$. (If there is no such block B_2, $i_2 = \infty$.)

By the collision-resistance of the hash function, for any valid checkpoint h_i, $i \in [m_1]$, with index $i < \min(r_1, r_2)$, the condition at Line 9 of Alg. 1 is true for c_1 if and only if it is true for c_2. Similarly, the clients must have updated their checkpointed chains with the same L_i at Line 10 of Alg. 1. This is because no checkpoint with index $i < \min(r_1, r_2)$, $i \in [m_1]$ triggers the stalling condition (Line 7, Alg. 1). Thus, if $i_1 = \infty$, $\mathcal{CP}_{r_1}^{\mathsf{c}_1} \preceq \mathcal{CP}_{r_2}^{\mathsf{c}_2}$. However, if $i_1 < \infty$, then $i_2 = \infty$, due to Line 7 of Alg. 1 being triggered for c_2. Thus, if $i_1 < i_2$, then $\mathcal{CP}_{r_2}^{\mathsf{c}_2} \prec \mathcal{CP}_{r_1}^{\mathsf{c}_1}$, and if $i_2 \leq i_1$, then $\mathcal{CP}_{r_1}^{\mathsf{c}_1} \preceq \mathcal{CP}_{r_2}^{\mathsf{c}_2}$. Finally, by the safety of Bitcoin with parameter k, $\mathcal{C}_{r_1}^{\mathsf{c}} \preceq \mathcal{C}_{r_2}^{\mathsf{c}}$ for any $r_2 \geq r_1$ and client c. Thus, $\mathcal{CP}_{r_1}^{\mathsf{c}} \preceq \mathcal{CP}_{r_2}^{\mathsf{c}}$. □

For the PoS protocol underlying Babylon to enforce accountability when there is a safety violation, it must ensure that the PoS blocks outputted by a client and the corresponding signatures are seen by at least one honest validator before the client goes offline; so that the honest validators can detect safety violations and create proofs accusing the adversarial validators. Hence, in the proof below, we assume that the consensus messages and blocks in a clients' views are eventually observed by the validators before the client goes offline.

Proof of Theorem 2. Suppose there are two clients c_1, c_2, and slots r_1, r_2 such that $\mathcal{L}_{r_1}^{\mathsf{c}_1}$ conflicts with $\mathcal{L}_{r_2}^{\mathsf{c}_2}$. Starting from the genesis, let B_1 and B_2 denote the earliest conflicting blocks in $\mathcal{L}_{r_1}^{\mathsf{c}_1}$ and $\mathcal{L}_{r_2}^{\mathsf{c}_2}$ respectively. As B_1 and B_2 share a common parent, they also share the same active validator set.

Let r denote the first slot such that a valid checkpoint for a chain L extending B appears in the (confirmed) Bitcoin chain held by a client c. If there is no such slot, let $r = \infty$. If $r < \infty$, define b as the Bitcoin block containing the checkpoint on $\mathcal{C}_r^{\mathsf{c}}$. By synchrony and the safety of Bitcoin, b appears in the Bitcoin chains of all awake clients (including honest validators) by slot $r + \Delta$ with the same prefix and contains the first valid checkpoint on those chains, whose PoS block extends B. Since B_1 and B_2 conflict with each other, a block in L also conflicts with at least one of the blocks B_1 and B_2 (By the collision-resistance of the hash function, the adversary cannot open the checkpoint within b to conflicting chains). Without loss of generality, suppose a block in L conflicts with B_1. Let B' denote the first block in L conflicting with B_1 and $r' \in [r, r + \Delta)$ denote the slot, in which b appears in c_1's Bitcoin chain for the first time (if c_1 is online). We next consider the following cases:

a) $r = \infty$: The active validators for the blocks B_1 and B_2 cannot withdraw their stake in any client's view before a valid checkpoint for a chain extending B appears in the (confirmed) Bitcoin chain held by the client, *i.e.*, before slot r. Moreover, the blocks B_1 and B_2 and the corresponding pre-commit signatures are eventually observed by an honest validator. Since B_1 and B_2 are finalized and conflicting, and the PoS protocol satisfies $n/3$-accountable safety, the honest validator creates and sends a fraud proof (to Bitcoin) that identifies at least $n/3$ adversarial validators in the common active validator set of B_1 and B_2 as protocol violators. This proof eventually appears in the Bitcoin chains held by all clients, implying that at least $n/3$ adversarial active validators become slashable their views. We hereafter assume that $r < \infty$.

b) $r_1 \geq r' + \Delta$: If L is available and finalized in c_1's view, then c_1 would not have outputted B_1 as part of its PoS chain since B_1 conflicts with B' (Line 9, Alg. 1). If L contains

a block that is unavailable or not finalized in c_1's view, then it would have triggered the stalling condition for c_1, implying that c_1 would not have outputted B_1 as part of its PoS chain (Line 9, Alg. 1). Thus, by contradiction, it must be that $r_1 < r' + \Delta$, and we hereafter assume $r_1 < r' + \Delta$.

c) The client c_1 is online, and L is available and finalized in c_1's view: In this case, since B_1 and B' are conflicting and finalized, and the PoS protocol satisfies $n/3$-accountable safety, c_1 creates and sends a fraud proof (to Bitcoin) that identifies at least $n/3$ adversarial validators in the common active validator set of B_1 and B' as protocol violators, by slot $r' + \Delta \leq r + 2\Delta$. By Proposition 1, this proof appears in the Bitcoin chains held by all clients c by slot r'' such that $|\mathcal{C}^{\mathsf{c}}_{r''}| = |\mathcal{C}^{\mathsf{c}}_{r}| + k$. Thus, at least $n/3$ adversarial active validators become slashable in all clients' views.

d) The client c_1 is online, and L is unavailable or not finalized in c_1's view: In this case, the chain L triggers the stalling condition for c_1 (Line 7, Alg. 1), in which case it sends a checkpoint to Bitcoin for its PoS chain at slot $r' + \Delta \leq r + 2\Delta$. By Proposition 1, this checkpoint appears in the Bitcoin chains held by all clients c by slot r'' such that $|\mathcal{C}^{\mathsf{c}}_{r''}| = |\mathcal{C}^{\mathsf{c}}_{r}| + k$. Since there are now valid checkpoints for conflicting PoS blocks (B_1 and B') with the same active validator set within k Bitcoin blocks of each other, and the PoS protocol satisfies $n/3$-accountable safety, at least $n/3$ adversarial active validators become slashable in all clients' views (*cf.* condition 3 for withdrawals in Section IV-C).

e) The client c_1 has gone offline by slot $r + 2\Delta$, and L is available and finalized in an honest validator v's view: In this case, c_1 must have sent its PoS chain including block B_1 and its pre-commit signatures to the validators by slot $r + 2\Delta$, which are received by all honest validators by slot $r + 3\Delta$. Thus, both B_1 and B' must have become available and finalized in v's view by slot $r+3\Delta$. Since these are conflicting blocks, and the PoS protocol satisfies $n/3$-accountable safety, v creates and sends a fraud proof (to Bitcoin) that identifies at least $n/3$ adversarial validators in the common active validator set of B_1 and B' as protocol violators, by slot $r + 3\Delta$. By Proposition 1, this proof appears in the Bitcoin chains held by all clients c by slot r'' such that $|\mathcal{C}^{\mathsf{c}}_{r''}| = |\mathcal{C}^{\mathsf{c}}_{r}| + k$, implying that at least $n/3$ adversarial active validators become slashable in all clients' views.

f) The client c_1 has gone offline by slot $r+2\Delta$, and L is unavailable or not finalized in all honest validators' views: In this case, c_1 must have sent its PoS chain including block B_1 and its pre-commit signatures to the validators by slot $r+2\Delta$, which are received by all honest validators by slot $r + 3\Delta$. If there is an honest validator v whose PoS chain at slot $r+3\Delta$ has a block B' conflicting with B_1, then, v creates and sends a fraud proof (to Bitcoin) that identifies at least $n/3$ adversarial validators in the common active validator set of B_1 and B' as protocol violators, by slot $r + 3\Delta$. By Proposition 1, this proof appears in the Bitcoin chains held by all clients c by slot r'' such that $|\mathcal{C}^{\mathsf{c}}_{r''}| = |\mathcal{C}^{\mathsf{c}}_{r}| + k$, implying that at least $n/3$ adversarial active validators become slashable in all clients' views.

On the other hand, suppose the PoS chains of all honest validators are consistent with that of c_1. Since L contains a block that is unavailable or not finalized in the honest validators' views, it triggers the stalling condition (Line 7, Alg. 1), in which case each honest validator sends a checkpoint to Bitcoin for its PoS chain at some slot in $[r+\Delta, r+3\Delta]$. By Proposition 1, these checkpoints appear in the Bitcoin chains held by all clients c by slot r'' such that $|\mathcal{C}^{\mathsf{c}}_{r''}| = |\mathcal{C}^{\mathsf{c}}_{r}| + k$. Moreover, these checkpoints are for PoS chains extending that of c_1. Since there are now valid checkpoints for conflicting PoS blocks (B_1 and B') with the same active validator set within k Bitcoin blocks of each other, and the PoS protocol satisfies $n/3$-accountable safety, at least $n/3$ adversarial active validators become slashable in all clients' views (*cf.* condition 3 for withdrawals in Section IV-C). □

Proof of Theorem 3. By Theorem 2, Babylon 1.0 protocol satisfies $n/3$-slashable safety. Hence, when the number of active adversarial validators is less than $n/3$ at all slots, it satisfies safety. In this case, no valid checkpoint for unavailable or non-finalized blocks appears on Bitcoin. Thus, clients never stop outputting new PoS blocks as part of their checkpointed and PoS chains.

Suppose a transaction tx is first input to an honest validator at some slot r. Then, from slot r on, each honest validator v includes tx in its proposal block until it outputs a PoS block containing tx. Let c' be the client with the longest PoS chain among all clients at slot r. By synchrony and the safety of the PoS protocol, for every client c, $\mathcal{L}^{\mathsf{c}'}_{r} \preceq \mathcal{L}^{\mathsf{c}}_{r+\Delta}$. Then, either $\mathcal{L}^{\mathsf{c}'}_{r} = \mathcal{L}^{\mathsf{c}}_{r+\Delta}$ for all clients c, or there exists a client c such that $\mathcal{L}^{\mathsf{c}'}_{r} \prec \mathcal{L}^{\mathsf{c}}_{r+2\Delta}$.

In the former case, every client agrees on the validator set at slot $r + \Delta$. By [26, Lemma 7], there exists a finite $T_{\mathrm{tm}} = \mathrm{poly}(\lambda)$ such that if every client agrees on the validator set, a new block that extends $\mathcal{L}^{\mathsf{c}'}_{r}$ is finalized and becomes part of all clients' PoS chains by slot $r + T_{\mathrm{tm}}$, with overwhelming probability. In the latter case, a new block that extends the longest PoS chain, thus all PoS chains held by the clients at slot r, is finalized in the view of c' by slot $r + 2\Delta$, and by synchrony, becomes part of the PoS chains in all clients' views by slot $r + 3\Delta$.

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

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

of generality, let r_1 and r_2 be the first slots B_1 and B_2 appear in c_1's and c_2's PoS chains respectively.

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

We first consider the case, where at least one of the blocks is a bundle. Without loss of generality, let B_1 be a bundle and B denote the common parent of B_1 and B_2. Let b denote the Bitcoin block with the liveness transaction that triggered the rollup mode, during which the valid bundle checkpoint h_1 for B_1 appeared in $\mathcal{C}^{\mathsf{c}_1}_{r_1}$. At slot r_1, the prefix of c_1's PoS chain ending at B_1 consists of two parts: (i) a checkpointed chain outputted using the prefix of $\mathcal{C}^{\mathsf{c}_1}_{r_1}$ that ends at the $2k^{\text{th}}$ block extending b, (ii) bundles extending the checkpointed chain until B_1.

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

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

Proof of Theorem 6. As the number of honest active validators is over $n/2$ at all times, no valid PoS or bundle checkpoint for unavailable or non-finalized blocks or bundles appears on Bitcoin. Hence, the clients do not stop outputting new PoS blocks or bundles as part of their PoS chains due to unavailable checkpoints (Lines 17 and 38 of Alg 2 are never triggered).

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

Once a client c observes b become k deep in its Bitcoin chain, which happens by some slot less than $r_1 + R$, it sends a checkpoint transaction for its PoS chain. Subsequently, b becomes at least $2k$ deep in c's Bitcoin chain by some slot $r_2 \leq r_1 + 2R$. In this case, there are two possibilities: If $\mathsf{tx} \in \mathcal{CP}^{\mathsf{c}}_{r_2}$, then for all clients c', it holds that $\mathsf{tx} \in \mathcal{L}^{\mathsf{c}'}_{r_2+\Delta}$. If $\mathsf{tx} \notin \mathcal{CP}^{\mathsf{c}}_{r_2}$, then all clients enter the rollup mode by slot $r_2 + \Delta$.

Upon entering the rollup mode, an honest validator v prepares a bundle containing tx, which is viewed by all clients and signed by all honest validators by slot $r_2+2\Delta$. Upon gathering these signatures, *i.e.*, by slot $r_2+3\Delta$, v sends a checkpoint for that bundle. By the security of Bitcoin, the checkpoint appears in the Bitcoin chain of all clients at the same position by slot $r_3 = r_2 + 3\Delta + R$, with overwhelming probability. Since the PoS protocol is accountable, an honest validator cannot be identified as a protocol violator and never becomes slashable in the view of any client. Thus, the signatures within the bundle checkpoint from the honest validators are counted and suffice to pass the $n/2$ threshold, implying the validity of the bundle. Consequently, $\mathsf{tx} \in \mathcal{L}^{\mathsf{c}}_{r_3}$ for all clients c.

Setting $T_{\text{fin}} = r_3 - r = 3\Delta + 4R + T_{\text{tm}} = \text{poly}(\lambda)$, we conclude the proof of T_{fin}-liveness for the Babylon protocol. □

Proof of Corollary 2. By Theorem 6, if the number of adversarial active validators is less than $n/2$ at all times, the Babylon protocol of Section V-B with fast finality satisfies T_{fin}-liveness, where $T_{\text{fin}} = \text{poly}(\lambda)$. Thus, if a transaction tx is input to an honest validator at some slot r, then for all clients c that follow the fast finality rule, tx will be in $\mathcal{L}^{\mathsf{c}}_{r+T_{\text{fin}}}$. Let $R = \text{poly}(\lambda)$, denote the confirmation latency of Bitcoin with parameter k. Let $T = \text{poly}(\lambda)$, denote an upper bound on the duration of epochs when there is no safety or liveness violation on Tendermint.

If $\mathsf{tx} \in \mathcal{CP}^{\mathsf{c}'}_{r+T_{\text{fin}}}$ for a client c', then, by synchrony and the safety of the checkpointed chains, it appears within the checkpointed chains of all clients by slot $r + T_{\text{fin}} + \Delta$. On the other hand, tx might be in $\mathcal{L}^{\mathsf{c}}_{r+T_{\text{fin}}}$ for all clients c, but not in $\mathcal{CP}^{\mathsf{c}}_{r+T_{\text{fin}}}$ for any client c. In this case, let e denote the epoch of the PoS block containing tx.

Suppose all clients eventually observe the same canonical PoS chain ending at the last block of epoch e, and no liveness transaction appears on the Bitcoin chain of any client by that time. At the end of the epoch, an honest active validator v sends a checkpoint transaction for its PoS chain, which happens at some slot $r' < r + T_{\text{fin}} + T$. Then, for any client c, v's checkpoint is in $\mathcal{C}^{\mathsf{c}}_{r'+R}$, with overwhelming probability. Then, either $\mathsf{tx} \in \mathcal{CP}^{\mathsf{c}}_{r'+R}$ for all clients c, or there is a checkpoint or fraud proof in the prefix of v's checkpoint for conflicting PoS blocks. In the latter case, the protocol enters the rollup mode as described in Section V-C, in which case tx is eventually included within the checkpointed chains of all clients by slot $r' + R + T_{\text{fin}}$.

Finally, if the above condition is violated, then there must have been a safety or liveness violation, or a liveness transaction must have appeared in the Bitcoin chain of a client, by slot $r + T_{\text{fin}} + T$. In this case, the process to enter the rollup mode is triggered by slot $r + T_{\text{fin}} + T$, and tx is eventually included within the checkpointed chains of all clients by slot $r' + T + 2T_{\text{fin}}$. Consequently, Babylon with slow finality satisfies T'_{fin}-liveness, where $T'_{\text{fin}} = T + R + 2T_{\text{fin}} = \text{poly}(\lambda)$. □