PoTS—A Secure Proof of TEE-Stake for Permissionless Blockchains

Sébastien Andreina, Jens-Matthias Bohli, Wenting Li, Ghassan O. Karame, Giorgia Azzurra Marson
NEC Laboratories Europe, Germany
Email: firstname.surname@neclab.eu

Abstract — Proof-of-Stake (PoS) protocols have been actively researched for the past few years. PoS finds direct applicability in permissionless blockchain platforms and emerges as one of the strongest candidates to replace the largely inefficient Proof of Work mechanism that is currently plugged in the majority of existing permissionless blockchain systems. Although a number of PoS variants have been proposed, these protocols suffer from a number of security shortcomings. Namely, most existing PoS variants are either subject to the *nothing at stake*, the *long range*, or the *stake grinding* attacks which considerably degrade security in the blockchain. These shortcomings do not result from a lack of foresight when designing these protocols, but are inherently due to the ease of manipulating "stake" when compared to other more established variants, such as "work". In this paper, we address these problems and propose a secure Proof of Stake protocol, PoTS, that leverages Trusted Execution Environments (TEEs), such as Intel SGX, to ensure that each miner can generate at most one block per "height" for strictly increasing heights—thus thwarting the problem of nothing at stake and a large class of long-range attacks. In combination with TEEs, PoTS additionally uses cryptographic techniques to also prevent grinding attacks and protect against posterior corruption. We show that our protocol is secure, in the sense of well-established cryptographic notions for blockchain protocols, down to realistic hardware assumptions on TEE and well-established cryptographic assumptions. Finally, we evaluate the performance of our proposal by means of implementation. Our evaluation results show that PoTS offers a strong tradeoff between security of performance of the underlying PoS protocol.

I. Introduction

Blockchain technologies are gaining increasing attention nowadays motivated by the wide success of the Bitcoin cryptocurrency. To reach distributed agreement, blockchains rely on consensus protocols which ensure that all nodes in the network share a consistent view on a common distributed ledger. Most existing open blockchain systems rely on Bitcoin's Proof of Work (PoW) to reach network consensus in permissionless systems that do not require the knowledge of nodes' identities. However, PoW has been often criticized for its huge waste of energy. It is estimated that Bitcoin miners consume, by the end of 2018, as much electricity as Austria, or equivalently 0.5% of the world's total consumption [1].

To remedy the limitations of PoW, the community has turned to Proof of Stake (PoS) protocols in the hope of offering a more efficient and environment-friendly alternative. Unlike PoW, PoS leverages virtual and energy-friendly resources such as the "stake" of a node in the system in order to perform leader election and maintain consensus in the network.

Nevertheless, although many PoS variants have been proposed [2], [3], [4], [5], [6], [7], [8], PoS is still not widely deployed in existing blockchains. Namely, in spite of their efficiency, PoS-powered blockchains still account for less than 5% of the market capitalization of existing digital currencies. This is mostly due to the fact that most existing PoS protocols are vulnerable to a number of security threats, such as *nothing at stake* [9], *long range* [10], and *stake grinding* [11] attacks. The former attack allows the nodes to mine conflicting blocks without risking their stake, which increases the number of forks in the system as well as the time to reach consensus in the network. Long range attacks (commonly referred to as history attacks) aim at altering the entire history of the blockchain starting from early blocks (even from the genesis block). This can be achieved when, e.g., the attacker acquires the private keys of old accounts which no longer have any stake at the moment, but that have accrued majority stake at previous block height h; the attacker can then construct a fork starting from block h leveraging these accounts. Finally, in stake grinding attacks, adversaries use their computational resources to increase their probabilities of being elected as leaders/miners of the upcoming blocks.

To remedy these attacks, several proposals suggest the reliance on deposit-based PoS [12], [13] and on checkpoints [2], [5], [7], [8]. Deposit-based PoS essentially requires each validator to make a deposit in the system; should a validator generate conflicting blocks their deposit will be withdrawn, thus alleviating nothing at stake attacks by introducing accountability in the system. Checkpoints, on the other hand, enforce that any fork generated prior to a sufficiently deep block (i.e., prior to a "checkpoint") is discarded. Although this solution limits the impact of the long-range attack as the earliest attack point has to be after the last checkpoint, it does not completely prevent long range attacks, especially when the victim is a node that freshly joined the network or that has been offline for a long time. To combat stake grinding attacks, a number of PoS proposals utilize a deterministic algorithm to elect block leaders. This, however, gives rise to another important problem: the leader of each valid block is predictable in the network, thus making it vulnerable to planned denial-of-service attack strategies. Moreover, an adversary can still perform stake grinding by skipping an opportunity to create

a block if they are able to increase their advantage over future blocks. These factors explain the reason why none of today's large permissionless blockchain platforms rely solely on PoS protocols in their offerings—in spite of the plethora of PoS proposals that populate the literature [2], [5], [7], [8], [12], [13].

In this paper, we address the aforementioned problems, and propose a novel solution—*Proof of TEE-Stake* (PoTS)—which addresses the shortcomings of existing PoS protocols by leveraging functionality from Trusted Execution Environments (TEEs), such as Intel SGX. PoTS emerges as the first workable PoS protocol that protects against all the aforementioned attacks on PoS protocols without hampering the performance of the underlying consensus protocol.

The security of PoTS relies on TEEs to enforce validators to sign blocks on strictly increasing heights, fully preventing nothing at stake attacks and a large class of long-range attacks. Since the relevant operation of the leader election process are computed within the TEE, PoTS prevents an adaptive adversary from corrupting future leaders, as well as malicious validators from mounting grinding attacks. In doing so, we point out that PoTS tolerates possible compromise of TEEs (e.g., Spectre [14] or Foreshadow attacks [15]) as long as the adversary does not control a significant number of TEE devices. Our solution additionally deals with stake-bleeding attacks [16] and posterior-corruption attacks by adapting a cryptographic technique to achieve forward security—updating signing keys and deleting old ones—to the blockchain setting. In contrast to purely cryptographic solutions, our protocol leverages the ledger functionality of the blockchain itself to allow for an unlimited number of key updates. Finally, we point out that our protocol can tolerate compromised TEEs as long as the adversary does not control a significant number of TEE devices.

We argue that PoTS is technically and economically viable. Indeed, TEEs are nowadays pervasive on mobile platforms, PCs, and servers [17]. While the main barriers of wide adoption of PoS lie in the lack of security given their high efficiency, PoTS bridges these gaps and establishes a strong trade-off between the efficiency and the security of PoS. More importantly, although our solution relies on TEEs, it does not tax ordinary users (that are not miners) from transacting in the system with such a requirement. Namely, PoTS does not require the presence of TEEs for issuing transactions, and validating the correctness of exchanged transactions/blocks.

This paper extends our previous work in [18]. Here, we additionally provide a security model, specifying formal properties for proof-of-stake protocols in the style of recent work from the cryptographic literature [19], [20], and considerably revisit the TEE-based protocol from [18] to meet these properties. Finally, we evaluate an improved implementation of the proposed scheme, and compare its performance with state-of-the-art PoS protocols.

We summarize our additional contributions as follows:

Security Model. We present a security model for blockchain protocols and specify the relevant properties that these protocols are expected to meet. Our model builds on well-established results from the cryptographic literature and addresses several security risks for PoS simultaneously.

Concrete instantiation. We present a secure Proof of Stake protocol, PoTS, which is resistant against nothing at stake, long range, and grinding attacks, and analyse its security in the presented model. Our solution relies on realistic hardware assumptions for TEEs and on standard network and cryptographic assumptions.

Prototype Implementation. We implement and evaluate a prototype based on our instantiation using Intel SGX, and we show that our solution does not incur any meaningful overhead compared to Nxt's protocol, while offering comparable security to the (slower) Ouroboros Praos protocol. Our implementation results also show that PoTS only moderately increments the TCB by approximately 250 Lines of Code (LoC).

The remainder of this paper is organized as follows. In Section II, we overview existing PoS protocols and discuss relevant attacks against these protocols. In Section III, we present a novel security model for (Proof of Stake) blockchain protocols. We then introduce PoTS in Section IV, and in Section V we analyse its security based on the model outlined in Section III. In Section VI, we describe a prototype implementation and evaluation of our proposal, and we compare its performance to the PoS schemes Nxt [7] and Ouroboros Praos [21]. In Section VII, we overview related work in the area, and we conclude the paper in Section VIII.

II. Notation & Background

In this section, we introduce the notation that we will use throughout the paper, recall some background concepts, and discuss relevant attacks on proof of stake protocols.

A. General notation

For $a,b \in \mathbb{N}$, we use the shortcut $[a..b]$ to denote the set $\{x \in \mathbb{N} : a \leq x \leq b\}$. We write $X \leftarrow x$ for assigning value x to variable X. We use $=$ instead for the equality operator. An expression $X = x$ evaluates `TRUE` if the values of variables X and x coincide, otherwise `FALSE`. For a tuple $X = (x_1,...,x_n)$, we denote the number of its components (i.e., its length) by $|X| = n$. We denote by $\|$ the concatenation operator, and write $X \xleftarrow{\|} x$ for the operation of appending a x to a string/list X. If X is a list, we also denote its length by $|X|$, and we refer to its last element by $\mathsf{Last}(X)$. If $n = |X|$, for $1 \leq i \leq j \leq n$ we denote by $X[i..j]$ the sublist of X containing from the i-th element up to the j-th element. We write $X \preceq Y$ to indicate that X is a prefix of Y, meaning that $X = Y[1..n]$ for some $1 \leq n \leq |Y|$ using the above notation. For $0 \leq k \leq n = |X|$ we write $X[1..-k]$ to denote the prefix of X obtained by chopping the last k elements (conventionally $X[1..-0] = X$ and $X[1..-n] = \emptyset$). Within an algorithm specification: when writing $\mathsf{Parse}(X)$ we mean the operation that labels the components $x_1,...,x_n$ of a tuple X, implicitly assuming the format of X be known to the algorithm; the instruction 'Enforce C', should the condition C be be unfulfilled, causes the algorithm to halt with output `FALSE`.

B. Blockchain protocols

At a high level, the goal of a blockchain protocol is to allow mutually untrusted parties to agree on a growing number of transactions, i.e., to instantiate a *distributed ledger*. In a *permissionless* (a.k.a. open) blockchain, anybody can join the blockchain network, at any time, without the need of authenticating to a trusted entity. We correspondingly assume a dynamic set of $N \in \mathbb{N}$ mutually untrusted protocol partys, where 'dynamic' means that the set of parties may change over time (in particular: N is variable) and that each party may be online to contribute to the protocol execution, or be offline as it pleases them. Throughout the paper, we refer to each party, or blockchain node, with a (unique) handle $\mathcal{P}_i$ for $i \in [1..N]$. Parties interact with each other by exchanging messages over a peer-to-peer network in a broadcast fashion. We assume two abstract subroutines, BCAST and FETCH, providing application interfaces to send messages over, respectively, to retrieve incoming messages from the network.

Each node may simply use the ledger functionality by issuing transactions, or they may actively contribute in the system by verifying the validity of these transactions—the latter parties we name *validators*. To confirm transactions and assign them an order in the ledger, validators regularly broadcast *blocks*, containing a list of verified transactions, which build on top of each others—hence the term "blockchain". Apart from the initial block (dubbed "genesis block"), which is fixed upfront and emulates a core root of trust, all other blocks are proposed and agreed upon by the various validators. More concretely, validators also have to verify blocks, where a block is deemed as valid if (recursively) it builds on a valid blockchain and it contains valid transactions. Since all validators may propose blocks, a selection mechanism is put in place to indicate who will have the right to generate the next block.

Initialization. To join the system, a new party $\mathcal{P}_i$ runs an initialization procedure to create an account and retrieve the current blockchain. Upon initialization, the party's local state, that we denote by $state_i$, encodes all information necessary to run the protocol. We assume a minimal setup shared by all blockchain protocols: the state records the account key pair (pk_i, sk_i) and a copy $\mathbf{BC}$ of (the party's view on) the current blockchain.

Chain extension, Validation, and Fork Resolution. For the sake of abstraction, we describe the execution of the protocol through asynchronous rounds, where each round r corresponds to the creation of block B_r. Throughout the paper, we conventionally refer to the length of a given blockchain (i.e., the number of blocks it contains) also with the term *height*; thus, we may use the expressions "block of round r" and "block at height r" interchangeably. In each round, all (online) parties compete to produce the next block, and a leader-election procedure resolves this competition by indicating who has the "right" to generate the block for that round. Depending on the protocol type, this prerogative may be assigned based on various resources (such as computing power in proof-of-work.) More precisely, the protocol Π that parties execute to maintain the blockchain consists of three phases: firstly, after retrieving new blocks and transactions from the network, parties (are expected to) verify the validity of transactions as well as of possible alternative chains they received. In the sequel, we denote the chain-validation procedure by Validate. Should a party receive multiple valid chains, they run a fork-resolution algorithm Resolve to establish which one among these chains should be selected as 'the current' one to built upon. Notice that the possibility of having more than one eligible party per round is inherent in *eventual consensus* protocols, which favors liveness over safety (cf. Section III-A). Finally, the party attempts to extend the updated chain by generating the next block; this procedure we name Extend.

For completeness, we provide an abstract specification of a generic (eventual-consensus) blockchain protocol in Figure 1.

Algorithm $\mathsf{Init}_\Pi(\mathcal{P}_i)$:
1. Generate account $\langle (sk_i, pk_i), stake_i \rangle$
2. Retrieve blockchain $\mathbf{BC}$
3. $\mathcal{T}^* \leftarrow \emptyset$ // Unconfirmed transactions
4. $state_i \leftarrow \langle \mathbf{BC}, (sk_i, pk_i), stake_i, \mathcal{T}^* \rangle$
5. Return $state_i$

Algorithm $\Pi(state_i)$:
6. $(\mathcal{C}, \mathcal{T}) \leftarrow$ FETCH
7. $\mathcal{C}^* \leftarrow \emptyset$
8. For all $\mathbf{C} \in \mathcal{C}$:
9. If $\mathsf{Validate}(\mathbf{C}) = \mathtt{TRUE}$: $\mathcal{C}^* \xleftarrow{\cup} \mathbf{C}$
10. $\mathbf{BC} \leftarrow \mathsf{Resolve}(\mathcal{C}^* \cup \mathbf{BC})$
11. If $\mathbf{BC}$ changed: verify $\mathcal{T}^*$ w.r.t. $\mathbf{BC}$
12. For all $tx \in \mathcal{T}$:
13. If tx is valid w.r.t. $\mathbf{BC}$: $\mathcal{T}^* \xleftarrow{\cup} tx$
14. Select TX from $\mathcal{T}^*$
15. $B \leftarrow \mathsf{Extend}(pk_i, sk_i, stake_i, \mathbf{BC}, TX)$
16. If $B \neq \perp$: $\mathbf{BC} \xleftarrow{\|} B$; BCAST$(\mathbf{BC})$
17. Return

Fig. 1: Abstract specification of a blockchain protocol for eventual consensus. We denote by $\mathcal{C}$ and $\mathcal{T}$ (resp. $\mathcal{C}^*$ and $\mathcal{T}^*$) the sets of received chains and transactions (resp. valid ones). We are deliberately vague about how transactions are verified (because it is a protocol-specific detail that does not play a role in our treatment). State variables (within $\langle\rangle$) are understood as global ones, i.e., any modification is permanent.

C. Proof-of-stake protocols

Proof-of-Stake (PoS) is an emerging blockchain consensus protocol that appoints block leaders depending on some properties of the stake they own. Here, "stake" is a virtual resource that is tied to a validator in a publicly verifiable manner. The idea behind PoS security is that as long as the majority of stake is held by honest parties, the protocol works correctly, i.e., honest parties do agree on a common ledger.

Formally, each party $\mathcal{P}_i$ is associated with the stake $stake_i$ they own in the system—thus parties are also referred to

as stakeholders. We assume an initial stake distribution $\mathcal{S}_0 = (stake_1,...,stake_N)$, where $N = n_0$ is the initial number of participants, to be encoded in the genesis block B_0; we also assume the latter to be distributed in a trustworthy manner.

A blockchain $\mathbf{BC} = (B_0,...,B_R)$ induces a stake distribution $\mathcal{S}_{\mathbf{BC}} = (stake_1,...,stake_N)$ where $stake_i$ is the stake balance of $\mathcal{P}_i$ w.r.t. blockchain $\mathbf{BC}$ (in this notation, $\mathcal{S}_0$ is a special case of $\mathcal{S}_{\mathbf{BC}}$ for $\mathbf{BC} = (B_0)$). Throughout the paper, we denote by $stake_i(\mathbf{BC})$ the stake of party $\mathcal{P}_i$ w.r.t. blockchain $\mathbf{BC}$.

The community currently features a number of PoS variants. Peercoin [2], Cloakcoin [3], and Novacoin [6] use "coin age", i.e., the accumulated time a node holds their stake before generating a new block, as mining resource. Since hoarding the coins increases one's chance to generate a block, relying on coin age discourages nodes from actively participating in the consensus process. VeriCoin [4] uses a similar mining resource, so-called "stake-time", which also takes into account coin age but it also incentivizes participation: if a validator does not generate any block for a long time, their stake-time starts to degrade. The consensus protocols of Blackcoin [5] and Nxt [7] rely on the amount of stake owned by validators. BitShares [8] proposes a so-called "delegated PoS", where validators vote for a group of delegates who are in charge of generating blocks in round-robin fashion, and each validator has voting power proportional to their stake. Slasher [12] and Casper the Friendly Finality Gadget (Casper FFG) [13], both proposed by Ethereum, are instances of deposit-based PoS, meaning that they require nodes to deposit their stake in order to become validators.

All of the above protocols are not supported by a thorough security analysis in a precisely specified model. The following protocols, instead, come with a cryptographic (reductionist) proof. Ouroboros [22] employs a multi-party computation protocol to generate unbiased randomness for the leader election process. This approach requires coordination among the validators in the network. Algorand [23], [24] is a PoS variant based on a Byzantine Agreement protocol [25]. In contrast to the blockchain protocols discussed earlier, which achieve *eventual consensus*, Algorand favors safety over liveness and makes *final* decisions. Thus, it prevents forks (with high probability) by design. Snow White [26] is specifically designed to offer protection in the presence of *posterior corruption*, meaning that security holds as long as the majority of the active stake is not controlled by the adversary. Here, "active" stake refers to the stake owned at the present by validators who are online. Neither Algorand nor Ouroboros are secure against posterior corruption, while Ouroboros Praos [21] improves Ouroboros by additionally offering this kind of protection. In Section VII, we discuss in greater detail the security guarantees provided by these protocol and the assumptions made to prove their security.

D. Attacks against PoS protocols

In what follows, we describe in greater details the nothing at stake, grinding, and long range attacks, and discuss existing countermeasures.

Nothing at stake attack. In most existing PoS protocols, validators have an incentive to work on multiple forks since generating a block in PoS is no more than generating one signature. In other words, in order to maximize their benefits, validators could generate conflicting blocks on all possible forks with nothing at stake. This behavior, referred to as the *nothing at stake* attack [9], slows down the consensus process and, thus degrades performance. Moreover, it leads to frequent forks, making the system more vulnerable to double spending attacks.

In terms of countermeasures, Ouroboros [22] introduces a reward mechanism that specifically incentivizes validators not to maintain incompatible chains. However, it only discourages opportunistic adversaries and cannot prevent targeted attacks, such as double-spending, that would benefit from (temporarily) forked blockchain. Slasher [12] proposes a punishment mechanism to discourage dishonest behavior. Specifically, it requires validators to provide a deposit which is then locked for a given period; should a validator generate conflicting blocks at the same height, they will lose the deposit. BitShares [8] adopts a similar approach to Slasher: any misbehaving validator loses the ability to generate blocks in the future.

A major drawback of deposit-based countermeasures is that they freeze a considerable amount of stake in the network. In addition, they do not prevent targeted double-spending attacks involving transactions carrying more than the deposit.

Long range attack. Long range attacks [10] refer to the ability of an attacker to rewrite the blockchain history starting from a block generated far in the past. Since this attack requires the majority of voting power, it is inherently resolved in a proof of work setting under the honest majority assumption of computing power. In a proof-of-stake setting, however, long range attacks can be mounted by controlling accounts that have no stake at the moment, but used to have a large amount of stake at some point in the past. That is, an attacker can create forks from past blocks and overtake the current chain by leveraging "past majority of stake". It could do so, for instance, by compromising the private keys of nodes which no longer have stake at the moment, but were wealthy and jointly accrued majority of stake in the past.

The most common countermeasure is to use *checkpoints* to limit the impact of such attacks. A checkpoint refers to a round r such that the blockchain prefix $\mathbf{BC}[0..r]$ is regarded as immutable. A number of PoS instantiations [2], [5], [8] rely on a centralized checkpoint server which periodically defines the correct chain. Following the checkpoints principle, correct Nxt nodes do not accept a fork that differs from the local chain except for the last 720 blocks [7]. Similarly, Snow White [26] and Ouroboros [22] instruct nodes to reject alternative chains which differ from the local chain far in the past (how "far" depending on the specific choice of security parameters). Ouroboros Praos [21] proposes a mitigation based on key-evolving cryptography; however, this only prevents long-range attacks that exploit key compromise, and still assumes checkpoints to achieve full security. We emphasize that checkpoints require nodes to be synchronized, as anybody who recently

joined the network (or re-joins after a long offline period) has no way to distinguish honest chains from malicious ones.

Stake grinding attack. Blockchain consensus protocols select the leader for each round proportionally to some mining resources, depending on the protocol type (e.g., computing power in the case of proof of work). In the ideal case—if parties do follow the protocol faithfully—the probability of being elected leader is proportional to the resource defining the protocol type, and independent of any other resource.

In some proof of stake protocols, an attacker can leverage computational resources to increase their chance to generate a block, violating the above-mentioned principle. This is commonly referred to as a *stake grinding* attack [11]. For instance, consider an eligibility function which depends only on a validator's stake, the previous block, and some input variable that can be (partially) chosen by the validator. Then, an attacker could, once elected leader, compute multiple candidate blocks (one for each value of the input variable) until they find one that lets them become leader again in the subsequent round. In this way, the attacker could in principle monopolize the leader election process. While, strictly speaking, this is not an attack on the consensus protocol, it may give an unfair advantage to greedy validators.

Nxt [7] attempts to mitigate stake grinding using a deterministic algorithm for the leader election process. This measure leads to a different issue, referred to as "transparent forging": here, anybody can predict which validators will become leaders in the future. Transparent forging opens an additional attack vector by allowing adversaries to selectively nit-pick the next leader to compromise. Even worse, an adversary can still perform stake grinding by skipping opportunities to create a block in order to increase their advantage over future blocks [27]. Blackcoin [5] also relies on a deterministic election process, but aims at mitigating the transparent forging problem using a "stake modifier", which periodically introduces entropy to the leader selection process. This technique, however, only limits the period in which leaders are predictable, namely between any two subsequent updates of the stake modifier.

Ouroboros [22] uses multi-party computation to generate unbiased randomness for the leader election process, and provably prevents the attack—however, at the expense of higher communication/synchrony requirements. Stake grinding attacks do not apply to Bitshares [8] and Casper FFG [13] PoS protocols. In Bitshares, leaders are not chosen randomly but selected in a round-robin fashion from a set of delegates. Casper FFG instead relies on proof-of-work for the leader election, and uses a PoS layer only to finalize the blocks generated within a certain time interval.

Table I summarizes the security of existing PoS protocols against the nothing at stake, long range, and stake grinding attacks. As one can see, none of these protocols offers full protection against all of the attacks.

Protocol	Secure against:		
	NaS	LR	Gr
Cloakcoin	✗	✗	✗
Novacoin	✗	✗	✗
Blackcoin	✗	○	○
Peercoin	✗	○	✗
Nxt	✗	○	○
Slasher	○	✗	✗
Casper FFG	○	✗	–
Vericoin	○	✗	✗
BitShares	○	○	–
Snow White	✗	○	✗
Ouroboros	✗	○	✓
Ouroboros Praos	✗	○	○
Algorand	✗	○	✓

TABLE I: Resilience of existing PoS protocols against nothing at stake (NaS), long range (LR), and grinding attacks (Gr). The symbol '○' indicates that security is only partially achieved, while '–' means that the attack does not apply to the considered protocol.

III. Security model

In this section, we present a formal model for the security of blockchain protocols, describing the major properties that a blockchain consensus protocol should target according to established works in the domain of provable security and pinpointing how these properties capture the aforementioned attacks (among others).

A. Blockchain security

Functionality and security properties targeted by blockchain protocols are typically specified in terms of *safety* (a.k.a. *consistency*) and *liveness*. Conforming with analogous notions for consensus, safety means that honest parties—who faithfully follow the protocol—agree on the same sequence of transactions, perhaps with the exception of a few "most recent" ones, while liveness guarantees that transactions are confirmed relatively quickly (i.e., progress happens). Both properties are expected to hold despite limited Byzantine failures and the network being asynchronous for potentially long periods[1], as we expand next.

Following the modeling approach of David *et al.* [21], we assume that time is divided in discrete units called *slots*, and that the protocol participants have loosely synchronized clocks that indicate the current slot; in what follows we use the terms slot and time interchangeably. Our model assumes a *partially synchronous* network [29]: messages may be delayed, but for at most Δ slots (otherwise, a participant who does not receive messages within Δ slots should be considered offline). We assume that protocol participants have *loosely synchronized clocks* and, for the sake of abstraction, we also assume any potential drift to be insignificant compared to the length of a

[1]While it is impossible to provably achieve safety and liveness in fully asynchronous networks [28], blockchain protocols target *eventual* consistency, resp liveness, meaning that the chain of some honest party may be inconsistent, resp. stuck, during asynchronous periods but then it will catch up as soon as the network obeys a synchronous regime.

time slot. Thus, without loss of generality, all clocks agree on the current slot. We consider an adversary $\mathcal{A}$ who controls the network and may adaptively corrupt any party of their choosing, meaning that $\mathcal{A}$ obtains the entire state of that party and hence may fully control them. The number of corruptions is limited, however, such that the majority of (online) stake remains in honest hands, otherwise the adversary can take over the blockchain and generate all the blocks, and the corruption model is "mildly adaptive" (as in [22], [26]), meaning that it takes a short while for $\mathcal{A}$'s corruption requests to become effective.

We assume that the adversary may obtain all network messages, delay and modify them, subject to the assumed cryptographic constraints, i.e., $\mathcal{A}$ may not modify integrity-protected fields generated by uncorrupted participants, find hash collisions etc, and delivery guarantees, i.e., $\mathcal{A}$ may not delay messages forever.

In the cryptographic literature, consistency/safety and liveness for blockchain-based distributed ledgers are formalized in terms of the three properties described below (that we borrow from [19], [30], [31] and adapt to our syntax). For a secure blockchain protocol, we require the properties below to hold in the adversarial model just described, particularly under the assumption that $\mathcal{A}$ does not control the majority of active/online stake. For ease of presentation, we use the following notation: if $\mathbf{BC}$ denotes the local blockchain maintained by party $\mathcal{P}_i$, we write $\mathbf{BC}^t$ for the view of $\mathcal{P}_i$ on the blockchain at time-slot t.

The first property, *common prefix*, is closely related to consistency and states that the blockchains of honest parties share a large common prefix, i.e., they may differ only in (few) trailing blocks.

Definition 1 (Common prefix)**.** *Let $\mathbf{BC}_i$ and $\mathbf{BC}_j$ be the blockchains of honest parties $\mathcal{P}_i$ and $\mathcal{P}_j$, for $i,j \in [1..N]$. Then, for all slots t and t', with $t \leq t'$, it holds $\mathbf{BC}_i^t[0..-s] \preceq \mathbf{BC}_j^{t'}$, where s is called the stability parameter.*

The second property, *chain growth*, guarantees that the blockchains maintained by (honest) parties are extended relatively quickly, depending on the synchronicity of the network.

Definition 2 (Chain growth)**.** *Let $\mathbf{BC}$ be the blockchain of some honest party and $\ell \in \mathbb{N}$ a security parameter. Then, for every slot t and every $T > \ell$ it holds $|\mathbf{BC}^{t+T} - \mathbf{BC}^t| \geq g \cdot T$, where g is called the growth parameter.*

The third property, called *chain quality*, establishes that if one observes the evolution of a honest blockchain for sufficiently many rounds, among all the newly added blocks a relatively large fraction has been contributed by honest parties.

Definition 3 (Chain quality & fairness)**.** *Let $\mathbf{BC}$ be the blockchain of some honest party and $\ell \in \mathbb{N}$ a security parameter. Then for every round r and every $T > \ell$, the number of honestly produced blocks in the subchain $\mathbf{BC}[r..r+T]$ is at least $q \cdot T$, where q is called the quality parameter. In the ideal case, every party (or coalition of parties) holding a fraction ρ of the overall stake should generate roughly a fraction ρ of the blocks (*ideal chain quality*, or* fairness*).*

The common prefix, chain growth, and chain quality properties of a blockchain protocol imply safety and liveness of the associated distributed ledger [19], [30]. Specifically, common prefix implies safety, while chain growth and chain quality together imply liveness. The chain quality property also loosely expresses the fairness of a blockchain protocol, in the sense of guaranteeing that each validator becomes block leader (and hence obtains the reward associated to the block) with probability proportional to the owned stake. Given this, a fair PoS protocol is one achieving an *ideal* chain quality, meaning that the fraction of generated blocks is the same as the fraction of the held relative stake.

We finally note that the above properties capture (the absence of) nothing at stake, grinding, and long range attacks in the aforementioned adversarial model. Namely, a long range attack causes honest parties to revert their blockchains starting from a block far in the past, hence violating the common prefix property and, more generally, the safety of the underlying ledger. A grinding attack allows a dishonest party to generate more blocks than they should, thus violating the chain quality property; this attack also has an impact on the liveness of the associated ledger, because having dishonest parties generating most blocks may slow down the process of including transactions into the blockchain. Perhaps surprisingly, the nothing at stake attack is not captured by the above properties as it does not violate them directly. However, a large number of parties mounting the attack may have a negative impact on the liveness as it significantly slows down the consensus process.

IV. PoTS: A secure PoS protocol

In this section, we present PoTS—Proof of TEE–Stake—a novel proof-of-stake protocol that binds voting power to stake *and* Trusted Execution Environment (TEE) devices, and meets security in the strong adversarial model described in the previous section.

A. PoTS in a nutshell

The core component of PoTS is the use of trusted execution environments (TEEs) to enforce "sufficiently honest" behavior of validators, as proposed in [18]. Following this principle, a strawman solution would be to execute the protocol as a whole within the trusted application, which is sufficient for disallowing validators to deviate from the protocol instructions. This strawman approach, however, suffers from a number of shortcomings:

Need for trusted hardware: Ordinary users (who do not wish to become validators) might not be necessarily equipped with TEE platforms. Embedding the entire PoS protocol within a TEE would prevent a proper deployment of the protocol in realistic settings.

Need for registration: To guarantee that TEE platforms run the correct applications, users typically need to register with the hardware manufacturer to enable attestation services. For instance, Intel SGX attestation requires registration with Intel Attestation Service (IAS) [32]. Clearly, such a process undermines the system's ease of deployment.

TCB size: Special care should be taken when deploying complex applications within TEEs, as exploits and vulnerabilities of an application itself cannot be prevented by the trusted hardware. Therefore, one should specifically cater for a minimal TCB size where only the critical functions of an application are implemented within TEEs (i.e., so that the trusted application can be formally verified).

Security in the presence of compromised TEEs: Recent news reveal that TEEs might as well be compromised [33], [34], [35], [36], [37], [38]. Should even a single TEE be compromised, the aforementioned straw-man solution can no longer offer security. Namely, the strawman solution would no longer offer protection against posterior-corruption attacks. This is the case since an adversary can easily acquire stake majority in the past by transferring the stake of posterior-TEE-corrupt users.

Security against stake-bleeding attacks: Finally, we note that using TEEs does not suffice to prevent a minority coalition of dishonest validators to maintain a private blockchain, which may eventually take over the honest one. In a PoS setting, this problem can lead to stake-bleeding vulnerabilities.

In this paper, we designed our PoS protocol to specifically address these challenges. In contrast to the strawman solution mentioned above, PoTS employs TEEs only for the critical operations of the mining protocol, namely eligibility and block signing. In doing so, PoTS ensures a small TCB size, resulting in about 250 LoC. This would be already sufficient to achieve a decent level of security, as we expand in Section V.

Our solution complements the trusted hardware component with a cryptographic technique that requires users to frequently update their account keys in order to prohibit the transfer of large amounts of stake (in the past) to compromised accounts, and ultimately prevents the adversary from rewriting the blockchain history. In particular, PoTS is robust against posterior-corruption attacks. We emphasize that the latter technique is also effective in the realistic case an adversary manages to compromise some TEE platforms and extract the corresponding cryptographic keys, provided that the stake associated to the compromised TEEs does not account for the majority of stake in the network. Furthermore, PoTS employs a technique that prevents dishonest validators from incorporating transactions to a privately maintained blockchain (i.e., to perform stake-bleeding attacks) by letting users add to each transaction a pointer linking to the most recent blockchain [39].

As a by-product, we stress that the operations computed within the TEE exclusively affect the mining protocol. Therefore, only validators need to be equipped with TEEs, which makes PoTS viable for ordinary users who do not wish to mine. Moreover, among the validators, only a few of them need to be registered with the TEE manufacturer. Namely, PoTS leverages a proxied remote-attestation protocol to distribute trust among all validators, letting the few designated validators (proxies) bootstrapping trust among the manufacturer and the other validators without compromising the security of the registration process.

B. Building blocks

In the following, we discuss the building blocks and corresponding assumptions that our protocol uses.

Cryptographic primitives. Similarly to other blockchain proposals, our protocol uses cryptographic primitives such as signature schemes and hash functions to protect the integrity of transactions and blocks and to link blocks to the intended chain. We assume a cryptographic hash function $H\colon \{0,1\}^* \to \{0,1\}^\ell$ and model it as a random oracle [40] in the sequel. For the signature scheme, our protocol uses the (Elliptic Curve) Digital Signature Algorithm (EC)DSA [41], however, its security relies generically on the unforgeability of the scheme [42]. For ease of presentation in the protocol specification as well as its analysis, we assume a generic signature scheme $(\mathsf{KeyGen},\mathsf{Sign},\mathsf{Verify})$ that offers unforgeability under a chosen-message attack.

Trusted Execution Environments (TEEs). The mining protocol of PoTS relies on TEEs, which are pervasive nowadays and supported by many commodity platforms. For instance, Intel's SGX is being deployed on PCs and servers, while mobile platforms are mostly supported by ARM's Trustzone. TEEs provide an isolated environment running in parallel with the rich operating system. They provide standard cryptographic functionalities and restrict the memory access from the hosting OS—thus ensuring secure execution for the code running inside TEE. As a result, the data integrity and confidentiality of trusted applications deployed inside TEE are ensured. In the following, we describe the core properties of TEEs that we leverage for building a secure PoS protocol.

Remote attestation: A TEE is able to provide publicly verifiable proofs of the integrity of the initial code and data of the trusted applications. Should a secure-hardware manufacturer be involved in the remote attestation process, we assume the manufacturer offers a trusted remote attestation service (RAS) such as the Intel Attestation Service. Throughout the paper, we refer to the attestation evidence that the TEE platform prepares for the RAS as a quote, denoted by `RA_quote`. A quote usually contains a measurement, performed by the TEE and signed with a dedicated attestation key provisioned by the manufacturer. Upon receiving `RA_quote`, the RAS validates the measurement and any other relevant data included into the quote, and returns a signed attestation result `RA_result`, which can be then publicly verified. The RAS, however, should not be able to link any of the quotes and the TEE platforms that generated them, therefore preserving the users'anonymity.[2]

Trusted Monotonic Counters: To prevent replay attacks, the TEE platform provides tamper-resistant counters whose values cannot be reverted once incremented. Monotonic counters often rely on the non-volatile memory in the

[2]The attestation proof preserves the privacy of the platform's user using cryptographic primitives such as Direct Anonymous Attestation [43] (DAA) or Enhanced Privacy ID [44], [45] (EPID) of Intel.

TEE platform, which is usually implemented as flash memory and only supports a limited number of writes—resulting in a wear-out of those counters. Therefore, TEE implementations often throttle the increment operation of the monotonic counters [46], [47].

Isolation: Data within a TEE application resides in the protected memory which has guaranteed confidentiality and integrity; data and code used by the TEE application cannot be modified by any external processes.

Sealing: In case of lack of secure persistent storage, application data is encrypted and authenticated using platform-specific keys before being passed to the persistent storage of the untrusted system. The sealed data can only be decrypted if the integrity of the application and the data is assured.

The above properties can be assumed to hold as long as the TEE platform is not compromised (and its cryptographic keys cannot be accessed). Such assumption is not always realistic, as demonstrated by a series of attacks on TEEs [33], [34], [35], [36], [37], [38]. Thus, any security argument that relies on these properties can be in principle challenged in the presence of compromised TEEs which no longer offer the relevant guarantees. We anticipate, and later expand in the security analysis (cf. Section V), that PoTS retains its security even in such cases, provided that the adversary cannot compromise a large number of TEEs, namely not as many to control stake majority at any point in time.

C. PoTS: protocol specification

We proceed with presenting our PoS proposal in details.

Setup. We require all validators in the network to be equipped with secure hardware to run trusted code within the TEE. Since block mining employs a trusted application within the TEE, validators need to prove to the network that they are hosting legitimate hardware for block generation, and that corresponding private keys are hardware-protected and not accessible from the outside. This can be achieved by distributing the trusted application as part of the client wallet application when nodes first join the network. To this end, a new node initializes the trusted application for block generation and registers it to the blockchain network.

For ease of presentation, we consider each transaction as a pair $tx=(m,\sigma)$, where m is the payload of the transaction and σ is a signature of m under the account key of the transaction issuer. We later discuss the detailed format of transactions in PoTS. The initialization and registration procedures are illustrated in Figure 2, and described below. As previously mentioned, the registration procedure assumes a scenario in which remote attestation requires the assistance of a trusted RAS.

In the rest of this section, we denote the trusted application by $\mathcal{A}_T$, and correspondingly write $\mathcal{A}_T$.Proc to indicate any procedure Proc executed by $\mathcal{A}_T$ within the TEE. More generally, P.Proc means that procedure Proc is executed by a party P.

Algorithm $\mathcal{A}_T$.Init():
18 $(sk_T,pk_T) \leftarrow_{\$}$ KeyGen // generate signing key pair
19 $r^{\texttt{last}} \leftarrow 0$ // track round of latest EP request
20 $request^{\texttt{bs}} \leftarrow$ FALSE // record if BS was requested
21 $ctr_T \leftarrow \langle r^{\texttt{last}}, request^{\texttt{bs}} \rangle$
22 Generate quote RA_quote for $\mathcal{A}_T$
23 Return $(pk_T$,RA_quote$)$

Algorithm $\mathcal{P}_i$.Register$(state_i$,RA_result$)$:
24 $\sigma^{\texttt{reg}} \leftarrow$ Sign$(sk_i$,RA_result$)$
25 $tx^{\texttt{reg}} \leftarrow (RA_result,\sigma^{\texttt{reg}})$
26 BCAST$(tx^{\texttt{reg}})$

Algorithm $\mathcal{P}_j$.Validate$(tx^{\texttt{reg}})$:
27 (RA_result$,\sigma^{\texttt{reg}}) \leftarrow$ Parse$(tx^{\texttt{reg}})$
28 Extract key pair (pk_i,pk_T) from RA_result
29 Verify $\sigma^{\texttt{reg}}$ under key pk_i
30 Verify RA_result under RAS public key
31 $pk_{T,i} \leftarrow pk_T$
32 Save $\langle pk_i,pk_{T,i} \rangle$ for account of $\mathcal{P}_i$
33 Return

Fig. 2: Initialization and registration of the TEE trusted application for block generation.

TEE initialization and registration. Upon receiving an *initialization* request, $\mathcal{A}_T$ generates a signing key pair (sk_T,pk_T) to be used for block generation. The trusted application also initializes a variable ctr_T which record relevant information about block signing requests, as we will see shortly. Hence, $\mathcal{A}_T$ produces an attestation quote RA_quote, which includes: a hash value $H(pk_i,pk_T)$ that binds the validator's account key pk_i and the block verification key pk_T generated by $\mathcal{A}_T$, a description/identifier of the trusted application itself, a measurement produced by TEE for the trusted application, and a signature of all these values under the attestation key provisioned on the TEE device. Finally, $\mathcal{A}_T$ returns the public key pk_T along with the quote. The node can now send the attestation quote to the RAS; the latter, upon verifying the quote, provides an attestation result RA_result stating that the application $\mathcal{A}_T$ is running on a trusted TEE platform. Finally, the attestation result is posted to the blockchain network by the node through a registration transaction $tx^{\texttt{reg}}$. Therefore, all peers can verify the RAS-signed attestation result and record the relevant data for the newly joint validator.

Typically, a remote attestation service requires all users to be registered with that service, which may be not always desirable in an open network such as a blockchain network. Our solution, however, relaxes the latter requirement by relying on a few nodes registered with the RAS who function as *proxies* and forward the quotes of all other nodes to the RAS (the latter can be realized via mechanisms similar to [48]). The proxied attestation protocol works as follows: A new validator $\mathcal{P}_v$ initializes the TEE trusted application included in the blockchain client

implementation, and sends an attestation request to the trusted application. The attestation request in turns includes the service provider ID (SPID)[3] of a selected proxy node $\mathcal{P}_p$. Hence, the trusted application generates a quote based on the given SPID, and validator $\mathcal{P}_v$ sends this quote to the proxy node $\mathcal{P}_p$, who further forwards the quote to the RAS through an authenticated and confidential channel. After verifying the quote, i.e., whether the attested trusted application is running on a legitimate TEE platform, the RAS signs the attestation result and sends it back to $\mathcal{P}_p$. The latter extracts the attestation result from the reply of RAS and sends it to $\mathcal{P}_v$. The validator can now prepare the registration transaction and broadcast it to the network.

Once the registration process is completed, the validator can participate the mining process.

As we will see in Section V, the security of our protocol relies on each TEE device supporting at most one enclave at a time. The remote-attestation procedure of our solution guarantees this. More concretely, if the TEE is instantiated with Intel SGX or with TPM, the attestation quote includes an EPID [44], [45] or a DAA [43] pseudoname, respectively, which can be configured as linkable by specifically choosing the basename. Namely, using the linkable mode, all quotes produced by the same platform must use the same basename, hence multiple enclaves running on the same platform result in the same EPID/DAA signature authenticating the quotes.

Chain Extension. Akin to the concept of *cryptographic sortition* introduced by Micali [24], cf. Section VII), our protocol lets each individual validator check privately whether they are eligible to extend the blockchain in a given round and, if this is the case, produce a publicly verifiable proof to provide evidence of this.

The chain extension algorithm Extend consists of three main steps: *(i)* it checks whether the validator is eligible to become leader in the next round, and if so it proceeds with *(ii)* preparing the block header, and *(iii)* signing both the header and the proof. A block B_r consists of validator's identifier $\mathcal{P}_{i_r}$, block header hd_r, proof of eligibility π_r, block signature σ_r, and a list of transactions TX. Both the eligibility proof and the block signature are computed with support of the TEE. We provide a full specification of the chain extension routine in Figure 3, and explain it in detail below.

To compute proof of eligibility and block signature, a validator must issue corresponding requests to the trusted application. The eligibility proof consists of an "eligibility signature" $\sigma^{\mathtt{ep}}$, which ensure public verifiability, and a timestamp to throttle the number of eligible validators. To generate the eligibility proof for round r, the validator submits a request to $\mathcal{A}_T$ containing the eligibility signature $\sigma^{\mathtt{ep}}_{r-1}$ for the previous block along with the current round value r (cf. lines 49–54 in Figure 3).

Each validator can only issue requests for strictly increasing block heights. To ensure this, the trusted application tracks the round information of each request using a pair of counters $ctr_T = (r^{\mathtt{last}}, request^{\mathtt{bs}})$, where $r^{\mathtt{last}}$ records

[3]Registered parties by RAS is assigned a service provider ID and a correspondent certificate.

Algorithm $\mathsf{Extend}(pk_i, sk_i, stake_i, \mathbf{BC}, TX)$:
34 $B_{r-1} \leftarrow \mathsf{Last}(\mathbf{BC})$
35 $(*, hd_{r-1}, \pi_{r-1}, *) \leftarrow \mathsf{Parse}(B_{r-1})$
36 $(*, T^{\mathtt{base}}_{r-1}, CD_{r-1}) \leftarrow \mathsf{Parse}(hd_{r-1})$
37 $(\sigma^{\mathtt{ep}}_{r-1}, t_{r-1}, *) \leftarrow \mathsf{Parse}(\pi_{r-1})$
38 $\sigma^{\mathtt{ep}}_r \leftarrow \mathcal{A}_T.\mathsf{SignEP}(r, \sigma^{\mathtt{ep}}_{r-1})$
39 $hit_r \leftarrow H(\sigma^{\mathtt{ep}}_r)$
40 $\Delta t \leftarrow hit_r / (T^{\mathtt{base}}_{r-1} \cdot stake_i)$
41 Repeat:
42 $t_r \leftarrow \mathsf{Clock}()$ // local time
43 Until $t_r \geq t_{r-1} + \Delta t$ // Eligible now!
44 Update $T^{\mathtt{base}}_r$ and CD_r
45 $hd_r \leftarrow (r, H(B_{r-1}), H(TX), T^{\mathtt{base}}_r, CD_r)$
46 $\sigma^{\mathtt{bs}}_r \leftarrow \mathcal{A}_T.\mathsf{SignBS}(r, hd_r, \sigma^{\mathtt{ep}}_r, t_r)$
47 $B_r \leftarrow (\mathcal{P}_i, hd_r, (\sigma^{\mathtt{ep}}_r, t_r), \sigma^{\mathtt{bs}}_r, TX)$
48 Return B_r

Algorithm $\mathcal{A}_T.\mathsf{SignEP}(r, \sigma^{\mathtt{ep}}_{r-1})$:
49 $\langle r^{\mathtt{last}}, request^{\mathtt{bs}} \rangle \leftarrow ctr_T$
50 If $r^{\mathtt{last}} < r$:
51 $ctr_T \leftarrow \langle r, \mathtt{FALSE} \rangle$
52 $\sigma^{\mathtt{ep}}_r \leftarrow \mathsf{Sign}(sk_T, \mathtt{"ep"} \,\|\, r \,\|\, \sigma^{\mathtt{ep}}_{r-1})$
53 Else: $\sigma^{\mathtt{ep}}_r \leftarrow \perp$
54 Return $\sigma^{\mathtt{ep}}_r$

Algorithm $\mathcal{A}_T.\mathsf{SignBS}(r, hd_r, \sigma^{\mathtt{ep}}_r, t_r)$:
55 $\langle r^{\mathtt{last}}, request^{\mathtt{bs}} \rangle \leftarrow ctr_T$
56 If $r = r^{\mathtt{last}}$ and $request^{\mathtt{bs}} = \mathtt{FALSE}$:
57 $\sigma^{\mathtt{bs}}_r \leftarrow \mathsf{Sign}(sk_T, \mathtt{"bs"} \,\|\, r \,\|\, hd_r \,\|\, \sigma^{\mathtt{ep}}_r \,\|\, t_r)$
58 $request^{\mathtt{bs}} \leftarrow \mathtt{TRUE}$
59 Else: $\sigma^{\mathtt{bs}}_r \leftarrow \perp$
60 Return $\sigma^{\mathtt{bs}}_r$

Fig. 3: Chain extension routine of PoTS. It is executed by validators and requires a registered TEE.

the latest round for which an eligibility proof request has been submitted, and $request^{\mathtt{bs}}$ is a boolean flag indicating whether a corresponding block signature in that round has been generated. Upon receiving an eligibility proof request, the trusted application checks whether the submitted round value r is strictly larger than the counter $r^{\mathtt{last}}$ and, if this is the case, updates $r^{\mathtt{last}} \leftarrow r$ and $request^{\mathtt{bs}} \leftarrow \mathtt{FALSE}$, computes the new eligibility signature $\sigma^{\mathtt{ep}}_r$ by signing $\mathtt{"ep"} \,\|\, r \,\|\, \sigma^{\mathtt{ep}}_{r-1}$, and returns the resulting $\sigma^{\mathtt{ep}}_r$; otherwise, it rejects the request.

The eligibility signature effectively provides a "cryptographic ticket" for the sortition-based leader election step: the validator can now use $\sigma^{\mathtt{ep}}$ to derive their so-called *hit* for round r, namely $hit_r = H(\sigma^{\mathtt{ep}}_r)$. The hit provides a value in $[0, 2^{\ell-1}]$, which is unpredictable unless $\sigma^{\mathtt{ep}}$ is known. The validator $\mathcal{P}_i$ is eligible to generate a block for round r if their hit is smaller than a given *target*, which in turns depends on the validator's stake $stake_i$, a target-base $T^{\mathtt{base}}_{r-1}$ value recorded in the latest block (which can be flexibly adapted to reach a desired block-generation

rate), and the elapsed time since the previous block has been generated. Note that any validator who waits long enough will eventually be eligible, and the time validator $\mathcal{P}_i$ needs to wait is given by $\Delta t = hit_r/(T_{r-1}^{\texttt{base}} \cdot stake_i)$ (cf. line 43).

Thus, once the local clock indicates time $t_{r-1} + \Delta t$, where t_{r-1} is the timestamp recorded in the latest block, the validator can submit the block signature request to the TEE (cf. lines 55–60).[4] For the block signature request, the validator provides the prepared block header hd_r, the eligibility signature $\sigma_r^{\texttt{ep}}$, and its (local) timestamp t_r (cf. line 46). The trusted application extracts the round r from the block header hd_r and checks it against the counters in ctr_T: if $r^{\texttt{last}} = r$ (i.e., the rounds match) and $request^{\texttt{bs}} = \texttt{FALSE}$ (i.e., no block signature has been requested for that round), $\mathcal{A}_T$ computes the block signature $\sigma_r^{\texttt{bs}}$ over $\texttt{"bs"} \,\|\, r \,\|\, hd_r \,\|\, \sigma_r^{\texttt{ep}} \,\|\, t_r$.

Upon successful completion of the above operations, the validator broadcasts the candidate block $B_r = (\mathcal{P}_i, hd_r, \pi_r, \sigma_r, TX)$, where $\pi_r = (\sigma_r^{\texttt{ep}}, t_r)$ and $\sigma_r = \sigma_r^{\texttt{bs}}$.

Chain validation and fork resolution. The validation algorithm Validate verifies that all blocks of the chain were generated "properly", according to the protocol specification. Notice that proofs of eligibility are publicly verifiable since account public keys and stake information can be retrieved from the blockchain itself. In Figure 4, we illustrate the details of the chain validation routine that we explain below.

Algorithm Validate($\mathbf{BC}$):
61 $t_{now} \leftarrow \mathsf{Clock}()$
62 $R \leftarrow |\mathbf{BC}|$
63 $(*, (*, t_R), *) \leftarrow \mathsf{Parse}(B_R)$
64 Enforce $t_R \leq t_{now}$ // no timestamp in the future
65 For $r \leftarrow R$ down to 1:
66 $(\mathcal{P}_{i_r}, hd_r, \pi_r, \sigma_r, TX_r) \leftarrow \mathsf{Parse}(B_r)$
67 $(r, h_r^{-1}, h_r, T_r^{\texttt{base}}, CD_r) \leftarrow \mathsf{Parse}(hd_r)$
68 $stake_{i_r} \leftarrow stake_{i_r}(\mathbf{BC}[0..r-1])$
69 $(*, T_{r-1}^{\texttt{base}}, *, (\sigma_{r-1}^{\texttt{ep}}, t_{r-1}), *) \leftarrow \mathsf{Parse}(B_{r-1})$
70 $(\sigma_r^{\texttt{ep}}, t_r) \leftarrow \mathsf{Parse}(\pi_r)$
71 Enforce:
72 $\mathsf{Verify}(pk_{T,i_r}, \texttt{"bs"} \,\|\, r \,\|\, hd_r \,\|\, \pi_r, \sigma_r)$
73 $h_r^{-1} = H(B_{r-1})$, $h_r = H(TX_r)$
74 $\mathsf{Verify}(pk_{T,i_r}, \texttt{"ep"} \,\|\, r \,\|\, \sigma_{r-1}^{\texttt{ep}}, \sigma_r^{\texttt{ep}})$
75 $H(\sigma_r^{\texttt{ep}}) \leq T_{r-1}^{\texttt{base}} \cdot (t_r - t_{r-1}) \cdot stake_{i_r}$
76 Return TRUE

Fig. 4: Chain validation routine of PoTS. It can be executed by any initialized node (no TEE is necessary).

Upon retrieving a blockchain $\mathbf{BC} = (B_0, \ldots, B_R)$, a node verifies the validity of all blocks, going backwards from the most recent one. To ensure that block leaders were truly eligible, however, the node must also ensure that the timestamp t_R of the latest block is consistent with the current time t_{now}, namely that $t_R \leq t_{now}$, meaning otherwise that the last validator faked the block timestamp (cf. line 64 in Figure 4).[5]

The block verification closely follows the operations of the chain extension routine, in reverse order. Namely, for each block B_r, the node verifies the validity of the block signature σ_r under the TEE public key pk_{T,i_r} of the alleged block leader $\mathcal{P}_{i_r}$ (cf. line 72), the hash values $h_r = H(TX_r)$ and $h_r^{-1} = H(B_{r-1})$ (cf. line 73), the consistency of the block timestamp t_r, the validity of the eligibility signature $\sigma^{\texttt{ep}}$ (cf. line 74), and the validity of the eligibility proof w.r.t. the block leader's stake when generating the block (cf. line 75).

We emphasize that no trusted hardware is required for chain validation. That is, a TEE is necessary only for the mining process.

Should multiple valid chains be detected, the validator chooses the one with highest cumulative difficulty CD_R (i.e., the cumulative difficulty of the latest block B_R). This process is similar as the Nxt's fork resolution [7].

Resuming application. All variables used within $\mathcal{A}_T$ are saved in the protected volatile memory of TEE, preventing the attacker from reading or manipulating their values. To allow $\mathcal{A}_T$ resuming these variables after a restart, however, they should also be saved onto the persistent storage of the untrusted system in a way that prevents replay attacks. Namely, $\mathcal{A}_T$ should detect if it is ever resumed with a saved copy that is obsolete. It should also prevent rollback attacks [49], e.g., that a user deliberately crashes the trusted application to reset the counter value ctr_T to the last saved version. We achieve this by using TEE monotonic counters and sealing function. More specifically, in the initialization phase (cf. Figure 2) $\mathcal{A}_T$ additionally initializes a TEE monotonic counter to value 1. Whenever $\mathcal{A}_T$ terminates normally, the value of the monotonic counter is saved along with the other variables through the TEE sealing interface. Then, when the trusted application restarts and is given a copy of sealed data, $\mathcal{A}_T$ unseals the data and obtains the "last" counter value from the copy: If it equals to the reading of the current monotonic counter, the latter will be advanced; otherwise, $\mathcal{A}_T$ terminates abnormally. As a result, in addition to preventing replay attacks, it is guaranteed that an abnormal termination of $\mathcal{A}_T$ will make the application refuse to start until it is re-initialized. Thus, any malicious validator who crashes the TEE in order to reset the round counters will have to register a new $\mathcal{A}_T$ instance (effectively making the attack worthless).

Transacting in PoTS: key refreshing and chain linking. Another crucial component of our protocol consists in generically tweaking the transaction format by introducing two additional fields to incorporate a key-refreshing mechanism and a method to link transactions to blockchain branches.

[4] However, someone else may have in the meantime generated the r^{th} block (or more), thus being eligible does not necessarily translate to actually becoming the leader for that round.

[5] To be precise, $t_R > t_{now}$ implies that *some* of the block timestamps have been faked, not necessarily the latest one, however, it also implies that the leader of the latest block did not properly check the timestamp of the second last block, and so forth going backwards, until the block for which the timestamp has been faked. Regardless of which one of these blocks includes a fake timestamp, all following blocks are invalid, and hence so is the chain.

The first tweak consists in a novel mechanism to refresh the account keys of blockchain users which draws inspiration from forward-secure cryptography [50], i.e., is based on frequent key updates coupled with secure deletion of old keys. Forward security specifically aims at limiting the effect of key exposure, offering the guarantee that data protected in the past—prior to key exposure—remains secure. We adapt this concept to PoTS by letting users update their account key-pair upon issuing each transaction, so that corruption of a user's account cannot be exploited to issue transactions far in the past.

The second modification of the transaction format consists in including to the transaction payload a pointer to the "current" blockchain (where the pointer could be the hash value of the last block of the chain), so that each given transaction is valid only w.r.t. the blockchain it points to. As discussed by Gazi et al. [39], this technique prevents transferring transactions to alternative chains maintained by malicious validators, and it particularly prevents stake-bleeding attacks.

Key refreshing: Whenever a user $\mathcal{P}_i$ wishes to issue a transaction with payload m, they also generate a fresh signing-key pair (sk'_i,pk'_i)[6], include the public key pk'_i in the payload obtaining $m' \leftarrow m \,\|\, pk'_i$, sign m' with the current signing key sk_i obtaining a signature σ', and broadcast $tx' = (m',\sigma')$ to the network. As soon as the transaction appears in a block, the other users register public key pk'_i as the updated account key of $\mathcal{P}_i$. The issuer waits until the transaction is confirmed, i.e., until it appears in a "sufficiently deep" block of the blockchain (depending on the stability parameter from Definition 1), hence it deletes the old signing key sk_i.

Our solution leverages the public-ledger functionality of the blockchain to make the key-update mechanism more efficient than traditional cryptographic techniques for forward security, which typically can only support a fixed numbers of key updates and require storage of relatively large keys. In particular, PoTS uses the blockchain itself a medium to communicate the key updates, and only requires storage of old signing keys for the transaction-confirmation time.

Chain linking: Whenever $\mathcal{P}_i$ wishes to issue a transaction with payload m, they generate a pointer that links to the current blockchain, e.g., $h' \leftarrow H(\mathbf{BC}_i)$, derive an updated payload $m' \leftarrow m \,\|\, h'$, and sign m' instead of m.

Both the key refreshing and the chain linking can be performed simultaneously as described in the generic method of Figure 5.

If $\mathcal{P}_i$ wishes to issue a transaction with body m:
1 $(pk'_i,sk'_i) \leftarrow_{\$} \mathsf{KeyGen}$
2 $h' \leftarrow H(\mathbf{BC}_i)$
3 $m' \leftarrow m \,\|\, pk'_i \,\|\, h'$
4 $\sigma' \leftarrow \mathsf{Sign}(sk_i,m')$
5 $tx' \leftarrow (m',\sigma')$

Fig. 5: Key refreshing and chain linking of PoTS.

V. Security analysis

In this section, we analyze the security of our PoS proposal according to the security model presented in Section III. Specifically, we isolate the crucial security features offered by the various components of PoTS and argue that these features are sufficient to prevent nothing-at-stake, grinding, and long-range attacks.

[6]The next account key pair could also be generated in advance to improve performance.

A. Basic protocol: PoTS-mini

For the sake of analysis, we present our protocol as being built in a modular way, starting from a basic PoS protocol that meets relevant blockchain properties in a weak adversarial model and then gradually enhancing such protocol to make it withstand an increasingly stronger adversary. For the resulting protocol, PoTS, the blockchain properties are also met in the "standard" (i.e., Byzantine) adversarial model.

The "basic" protocol, henceforth referred to as PoTS-mini, deterministically assigns the right to generate a block depending on public values, namely by comparing the current validator's hit (cf. line 39 from Figure 3) with a specified target, both publicly computable. Chain validation and fork resolution rules of PoTS-mini are derived in the natural way, consistently with the leader-election process.

An instantiation of this protocol type is Nxt [7], for which the hit is derived recursively by hashing the previous hit concatenated with public, validator-specific information.

It is reasonable to assume that PoTS-mini offers minimal functionality and security, namely that under the clock synchronization and network assumptions specified in Section III, if all validators execute the protocol faithfully then the properties of *common prefix*, *chain growth*, and *chain quality* are met.

Common prefix: Validators agree on the same (long prefix of the) blockchain since, in each round, either there is a unique leader or the most-difficult-chain rule resolves potential forks.

Chain growth: Since the target increases as time passes, in every round some active validator will eventually be eligible. By adjusting the value of $T^{\mathtt{base}}$, one can regulate the block-generation rate, ensuring that the blockchains of active validators grow in a steady fashion.

Chain quality: The leader election rule weighs validators depending on their stake, so that each validator has a chance to become leader proportional to the owned stake. Thus, the relative number of blocks generated by a given validator over a sufficiently long period is related to the validator's relative stake.

Note that the above assumptions on PoTS-mini do not demand any protection against compromised validators, however, they are expected to hold against network adversaries. We claim that such assumptions are reasonable in the network model specified in Section III, because the network is *partially synchronous* and all messages are cryptographically signed, hence the adversary can neither modify messages nor delay them forever. More-

over, we stress that the synchronicity assumption on the network is necessary for the security of all existing blockchain protocols, which would otherwise be vulnerable to eclipse attacks [51].

Meeting security in the above mentioned adversarial model clearly is not sufficient to yield a secure PoS protocol: as it is, PoTS-mini offers no protection against Byzantine faults and corruptions of users. Indeed, an adversary may for instance adaptively corrupt validators who have a high chance to become leader in the near future, posterior-corrupt a former quorum of validators and rewrite the blockchain history, as well as mount more subtle attacks.

In the rest of the section we identify the components that enhance the security of PoTS-mini, turning it into PoTS, and argue that our solution retains all the blockchain properties in the strong adversarial model outlined in Section III.

B. Turning PoTS-mini into PoTS

We now show how to turn PoTS-mini into a full-fledged secure PoS protocol, PoTS, by (i) letting validators privately determine their eligibility in a private manner by computing an eligibility signature, (ii) computing both eligibility signature and block signature within the TEE, (iii) letting users update their account keys whenever they issue a new transaction, and (iv) linking transactions to a specific branch. In the rest of this section, we isolate the specific (security) properties offered by each of the above components, and indicate the corresponding attack strategies they can mitigate.

When designing a TEE-based blockchain protocol, special care must be taken to protect against rollback attacks [52], [53], [54]. These attacks aim to replace current application data—that has been sealed and saved onto untrusted storage to enable secure resumption of the application—with an older version of it. In the specific case of PoTS, an attacker could feed the trusted application with obsolete values, such as an outdated block height, to enable signing of blocks in the past. As we will see in the course of this section, PoTS prevents rollback attacks by employing monotonic counters, and to further avoid the memory wear out issue of the latter, it lets the trusted application verify that the unsealed data is up-to-date by letting the monotonic counter only track the restarts of the application (c.f. Section IV-C).

Further, it should be noted that any security arguments based on TEEs rely on the trusted hardware being tamper proof. However, side-channel attacks on popular TEE implementations such as Intel SGX [33], [34], [35] and ARM TrustZone [36], [37], [38], as well as the recent Spectre [55] and Foreshadow [15] vulnerabilities on TEE-enabled microprocessors, make this assumption hard to justify in practice. In the case of PoTS, a successful side-channel attack would allow a malicious validator to extract the TEE signing key so that they can work around the checks enforced by the TEE during the eligibility and signing process. Nevertheless, we argue that PoTS, in contrast to other TEE-based PoS proposals (cf. Section VII-B), offers a reasonable level of security even against an adversary that can compromise some TEE devices and obtain the corresponding private keys.

We proceed with incorporating the various security components to PoTS-mini for turning it into PoTS.

Dealing with adaptive corruptions. Since in PoTS-mini the eligibility of validators is determined from public values, everybody could predict who will become leader for any round in the future and, in particular, an adversary could choose the validators to corrupt according to that. To make future leaders unpredictable, PoTS uses an unforgeable signature scheme and lets validators derive their hit by hashing the eligibility signature, i.e., $hit_r = H(\sigma_r^{\text{ep}})$.[7] We formalize this intuition in the following proposition.

Proposition 1 (Privately computable eligibility)**.** *If the signature scheme in use is unforgeable, the adversary cannot compute valid eligibility signatures for honest validators.*

Proof sketch. Every adversary $\mathcal{A}$, who can derive a valid eligibility proof σ_r^{ep} for round r and honest (i.e., non corrupt) validator $\mathcal{P}_i$, can be turned into a successful forger against the signature scheme, violating the assumed unforgeability. □

By Proposition 1 we have that, even if $\mathcal{A}$ can corrupt participants at any time, eligible validators are not visible as such until they broadcast a new block. That is, the adversary cannot take advantage of adaptive corruptions for targeting the next leaders.

Dealing with nothing at stake and grinding attacks. Making leaders unpredictable is not sufficient, on its own, to yield a secure PoS protocol, as it does not discourage dishonest behavior of validators. In particular, determining validators'eligibility via signatures introduces a grinding vulnerability, since a malicious validator could precompute multiple eligibility signatures for the same block and then present the most profitable one. Moreover, dishonest validators could compute several eligibility or block signatures for different, inconsistent blocks in order to maintain alternative chains (nothing at stake).

To overcome these issues, in PoTS eligibility and block signatures are computed within the TEE using a tamper-protected private key which is not known to the validator, in such a way that the TEE only allows one eligibility request, respectively, a block signature request, per round. Therefore, a malicious validator can neither precompute signatures on competing chains nor generate signatures on their own (without the TEE), and hence they have no way to compute the hits in advance on virtual branches for then adapting to the best chain. The following result formalizes the above intuition.

Proposition 2 (Restricting eligibility and block-signing requests)**.** *If the TEE supports remote attestation, isolation, sealing, and monotonic counter properties and the signature scheme is unforgeable, each validator can obtain at most one eligibility signature and corresponding block signature per block height, and only for strictly increasing heights.*

[7] Although in PoTS the signature is effectively computed within the TEE, for the above-mentioned security argument we only rely on the signature scheme's unforgeability. The role of the TEE will be clarified next.

Proof sketch. For the assumed unforgeability of the signature scheme and the isolation property of TEEs, the only way for a validator to compute an eligibility signature σ_r^{ep} for round r is by issuing an eligibility-proof request to the trusted application. Due to the monotonic counter property of TEEs (cf. Section IV-B) and the design of PoTS's block extension routine, the validator can issue at most one eligibility-proof request per round (specifically, this is guaranteed by the check in line 50 from Figure 3). Further, any attempt to mount a rollback attack by powering down the processor or induce abrupt rebooting would not give any advantage to the validator: the sealing property of TEEs guarantees that round counters are saved in encrypted form in the persistent memory of the hosting system and correctly resumed by the trusted application upon restarting the TEE. Therefore, the trusted application recognizes incorrect resumption as suspicious and will force re-initialization of the device for mining, ultimately making the attack worthless. □

By Proposition 2 each validator, when issuing the eligibility signature request for round r, has to "commit" to a given chain, namely the one including eligibility signature σ_{r-1}^{ep} in the latest block. In particular, validators can no longer attempt to extend multiple chains (of the same length) simultaneously, hence *nothing at stake attacks* are prevented.

Further, due to the eligibility signature being the only source of unpredictability in the leader election process, no validator can bias the randomness of the leader election in any predictable way: a malicious node can no longer create the conditions to bias the eligibility process in their favor, because they cannot establish what the favorable conditions are prior to obtaining the eligibility signature from the TEE (moreover, once σ_r^{ep} has been generated for a given round r, any other eligibility signature request for that round will be rejected). That is, *grinding attacks* are prevented, too.

Note that restricting eligibility and block-signature requests still allows validators to switch among chains, as long as the new chain is to be preferred. That is, the restriction does not harm functionality, and this is the case due to the fork-resolution rule which favors the most difficult chain: "natural" rollbacks always yield a more difficult chain, while making a block-signature request for an early round would yield an easier chain, meaning the validator is seeking for an illegitimate rollback.

We point out that Proposition 2 crucially assumes that all TEE devices in the system are tamper proof. In fact, an adversary who manages to compromise a TEE and extract the corresponding signing key could generate multiple eligibility and block signatures for every round. In particular, a compromised TEE allows a validator to maintain multiple, inconsistent chains. We argue that PoTS considerably hardens security against such attacks; namely, due to the complexity of side-channel attacks on TEEs, we can reasonably assume that compromised TEE devices under adversarial control only comprise a small share of the overall number of mining devices. Therefore, we can conclude that even in the presence of some compromised TEEs which are used to mine on inconsistent chains, the liveness of the protocol is not significantly threatened (cf. Section III-A).

Note also that obtaining a TEE's private key allows an adversary to generate an unlimited number of valid signatures on any message of their choosing. In particular, the adversary could grind eligibility signatures. Again, such grinding vulnerability is limited to the compromised TEE devices, which we assume to be only a few. Therefore, exploitation of compromised devices can be detected using statistical methods, for instance by comparing the expected block-generation rate of each given validator with the actual number of blocks they did generate (e.g., as in [56], [57]). In this way, if a validator presents a significantly higher rate than expected, their blocks are eventually rejected by the network.

Dealing with posterior-corruptions and long-range attacks. Enforcing that each validator can generate at most one block per round, and only for strictly increasing heights, can also make certain types of long-range attacks hard to mount. Namely, Proposition 2 implies that a malicious/corrupt validator cannot produce valid blocks in the past, prior to the latest block contributed by themselves. That is, each TEE device only allows to push the blockchain forward (and not moving backwards to cause a rollback). This means in particular that the adversary cannot posterior-corrupt a former quorum of validators and leverage their accounts to generate an alternative history, because none of their TEEs would allow to revert the blockchain to any earlier point than the latest round for which the TEE issued a block signature.

We point out that enforcing block signing within the TEE does not, however, prevent all types of posterior-corruption long-range attacks. Indeed, since any restriction on signing requests only applies to blocks previously signed using a given TEE, nothing prevents an adversary from exploiting a *different* TEE device to generate blocks in the past (for instance, using an idle TEE that did not yet mine). More precisely, a posterior-corruption attack against a former quorum of users/validators would allow the adversary to take over the majority of stake (in the past), transfer it to the adversary's account associated to the idle TEE, and then leverage the acquired stake to cause an illegitimate rollback.

To also mitigate posterior-corruption attacks of this type, PoTS lets users frequently refresh their signing keys and delete expired ones. Namely, each user performs a key refresh whenever they issue a transaction (cf. Figure 5), and they delete the previous signing key as soon as the transaction is confirmed. Therefore, corruption of a user's account may only give the adversary access to the most recently "confirmed" signing key, and to none of the older keys, and hence it limits the effect of posterior corruption.

Proposition 3 (Preventing posterior-corruption stake transfer)**.** *If users securely delete their old account signing keys, the adversary cannot issue transactions on behalf of corrupt users far in the past.*

We emphasize that publishing fresh public keys along with the issued transactions requires users to keep their old key pairs until the transaction is confirmed in the blockchain, as

otherwise the user would lose their signing capabilities should the transaction be reverted or lost. During the time interval necessary to confirm a given transaction, the user is in principle vulnerable to targeted attacks. While an adversary could in principle exploit such "vulnerability window" to posterior-corrupt users holding majority of stake in the past, we argue that such a scenario is extremely unlikely[8]. Note also that forward security crucially relies on the system's users truly (and securely) deleting their old private keys.However, some users might decide to deliberately store expired account keys (e.g., in order to sell these keys) or accidentally keep a copy of them. In PoTS, security against posterior-corruption attacks is guaranteed as long as sufficiently many users do delete their keys. Precisely, PoTS offers security against posterior-corruption long-range attacks provided that the adversary cannot exploit corruption of malicious/careless users to control the majority of stake and transfer all of this stake to their own account. That is, we only require that users who correctly delete expired keys reach a honest stake majority at any time (including in the past).

We finally point out that Proposition 2 no longer holds in the presence of compromised TEEs, opening the possibility to mount posterior-corruption long-range attacks in this case. Notice, however, that in order to generate a valid chain that has higher difficulty than the honest chain, the adversary must have the majority of voting power, and hence the majority of stake, in the past.[9] By the honest stake-majority assumption, the only way for the adversary to gather stake-majority in the past is to posterior-corrupt rich users/validators and transfer their stake (held in the past) to the account associated to the compromised TEE. Due to Proposition 3, however, transferring stake in the past is unfeasible. Thus, the only way for the adversary to exploit compromised TEEs is to posterior-corrupt a quorum of validators (holding majority of stake in the past) and also compromise all of their TEEs, and we can realistically assume that such attacks are hard to mount in our setting.

Therefore, PoTS fully prevents *long-range attacks*.

Dealing with stake-bleeding attacks. Finally, PoTS limits the effect of private mining by letting users link their transactions to the chain they believe to be the honest one (cf. Figure 5), so that these transactions can no longer be included into a different chain. In particular, transactions cannot be misused to enrich the miners of a private blockchain, and hence stake-bleeding attacks [39] are prevented, too.

Summary. To summarize, we showed that PoTS is robust against nothing-at-stake, grinding, and long-range attacks in the adversarial model from Section III and under the assumptions that the hash function in use behaves like a random oracle, the signature scheme is unforgeable, and the majority of TEE-stake is not compromised (at the TEE level) and provide the properties described in Section IV-B. Namely, PoTS retains its robustness even if the adversary compromises some TEE devices, as long as the compromised devices only comprise a small fraction of the validators'set and the stake associated to those devices never amounts to the majority of stake.

We argue that achieving this level of robustness thwarts any reasonable option to invalidate the blockchain properties offered by PoTS-mini. Therefore, PoTS inherits from PoTS-mini the properties of common prefix, chain growth, and chain quality, and realizes a secure blockchain protocol.

VI. Implementation & performance evaluation

In this section, we evaluate the performance of PoTS and compare it to a well known PoS protocol Nxt [7] and a provably secure PoS protocol Ouroboros Praos [21] by means of implementations.

A. Implementation setup

We implemented the various protocol components of PoTS, Nxt, and Ouroboros Praos in Golang. In all our implementations, we use SHA256 as the hash function. For the building blocks of Ouroboros Praos, we implemented the Key Evolving Signature (KES) scheme based on [58] using a 2048-bit modulus, and the Verifiable Random Function (VRF) based on [59] combined with the bilinear pairing library from [60], which uses 256-bit keys. In Nxt and PoTS, we use ECDSA [41] as the signature scheme with a key size of 256 bits.

We instantiate the TEE secure code of PoTS using Intel SGX. In the sequel, we follow Intel SGX nomenclature and refer to the code running inside a TEE as an enclave. The enclave program is written in C and counts up to approximately 250 lines of code overall. It implements the following interfaces: *initialization*, which generates the account keys and provides the remote attestation certificate; *eligibility_proof* and *sign_block*, which compute the eligibility proof and the block signature respectively; *disconnect*, which seals the current state and writes it onto the disk; and *load*, which restores the state from the sealed data. Here, we only measure the performance of *eligibility_proof* and *sign_block* which are involved in the block generation process, as those are the operations significantly affecting the overall performance of the protocol.

For each of the three implemented protocols, PoTS, Nxt, and Ouroboros Praos we deploy the validators on a server equipped with 8-Core Intel Xeon E3-1240 and 32 GB RAM. We prepare the transaction payload of a block, ranging from 1 KB to 1000 KB, for each validator, and we measure the time of generating and verifying a block performed by the validators. We also estimate the overall energy consumption of creating a block in the network based on the time used for block generation and the average power consumption rate of computers. We do this as follows: we assume that every full node is a potential validator, and hence compute the eligibility proof for all validators, while we assume only a handful of them are eligible and compute block signatures only for these. We therefore estimate the energy consumption of creating a block as the total amount of time spent on computing eligibility

[8]To mount such an attack, indeed, all users of a quorum holding majority of stake would have to transfer a considerable amount of their stake at the same time, and the adversary must be able to corrupt all of them instantaneously.

[9]The adversary in principle could also overtake the honest chain by faking block timestamps, but this would be detected due to the assumed weak synchronicity of clocks.

proofs and generating block signatures on all validators, times the power consumption of a commodity PC. For comparison, we compare this result with the running Bitcoin and Ethereum networks. We estimate the energy consumption of these PoW-powered blockchains based on the total network hashrate $\mathcal{H}$, the block time $\mathcal{T}$, and the energy consumption $\mathcal{E}$ of the most efficient ASIC mining devices available on the market . Therefore, we estimate the energy consumed to create a block as $\mathcal{H} \cdot \mathcal{T} \cdot \mathcal{E}$.

B. Performance evaluation

Our evaluation results are shown in Figure 6. Figures 6a and 6b depict the latency incurred in the block generation and verification procedures, respectively, with respect to different block sizes. We further inspect the latency of each *sub-operation* in the block generation and verification process in Figures 6c and 6d. Namely, the time to compute the hash of the transaction payload (Hash), to generate the eligibility proof (EP) and to prepare a valid block signature (BS). Last but not least, we show the energy-consumption estimation of each of the PoS protocols with regards to different difficulty levels of the network, and compare the result with two PoW-based blockchains, Bitcoin and Ethereum, in Figure 6e and Figure 6f.

Block Generation: In all three protocols, the block generation latency increases linearly with the block size as shown in Figure 6a. The linearity is due to the hash computation over the transaction payload when preparing the block header; generating eligibility proof and block signature for the block header, on the other hand, does not depend on the block size. As a result, the performance of the three evaluated protocols only differs in the eligibility proof and block signature generation as shown in Figure 6c. For instance, for a block size of 1KB, block generation in Nxt takes 0.08 ms while PoTS requires 0.42 ms. This additional performance overhead of 0.34 ms results from CPU switching to SGX mode when executing the trusted code. We note that such overhead remains constant, and hence it becomes insignificant for reasonably high block sizes. For example, with 1 MB block size, Nxt and PoTS incur respectively 12.5 ms and 12.9 ms of latency.

Meanwhile, Ouroboros Praos requires 34.3 ms and 45.9 ms when the block size is 1 KB and 1 MB respectively—exhibiting a significant overhead even with large block sizes. This is due to the relatively heavy cryptographic operations used in Praos such as evaluating a VRF or updating the key of the signature scheme.

Block Verification: The performance of block verification resembles that of block generation. Namely, the latency grows linearly with the increase of block size due to the hash computation over the transaction payload, and the difference among the three protocols is due to the eligibility proof and block signature verification process. As shown in Figure 6b, the time to verify a block with 1KB payload in PoTS is 0.2 ms, while it is 0.12 ms in Nxt. Since verification only requires public keys, the TEE enclave is not involved in PoTS to validate blocks. The slight overhead of 0.08 ms comparing to Nxt is caused by the difference in eligibility-check algorithms: Nxt performs a hash computation while PoTS a signature verification. This difference can be observed in Figure 6d, where the eligibility proof verification is 0.08 ms higher in PoTS than in Nxt. Similarly, the difference becomes negligible between Nxt and PoTS as the block size increases: 11.01 ms and 11.11 ms, respectively, are incurred for block size of 1MB. Similar to the performance of block generation, Ouroboros Praos takes significantly longer to verify a block than Nxt and PoTS due to the VRF verification on the eligibility proof. As a result, Praos requires 26.9ms for block size of 1 KB and 37.48ms for block size of 1 MB.

Energy Efficiency of PoS: Figure 6e estimates the energy required to generate a block for the three considered PoS protocols, and also reports Bitcoin's energy consumption for a comparison. We also show in Figure 6f the exponential increase in energy consumption of both Bitcoin and Ethereum networks within the last three years.

Recall that we assume all full nodes compute the eligibility proof and only a handful[10] generate the block signature. For the sake of comparison, we consider the scale of the Bitcoin network and assume the number of full nodes as 9600[11] at the time of writing. Meanwhile, for block generation we take the upper bound for the energy consumption of an average Intel i7 CPU, which is 100 W[12].

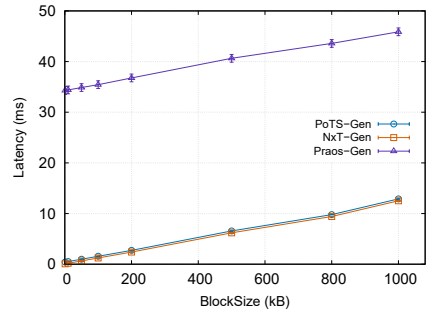
As shown in Figure 6e, PoTS incurs an negligible consumption of 5.1×10^{-5} kWh to create a block with the efforts of all validators in the network. It is two orders of magnitude more energy-efficient than Ouroboros Praos, which yields 4.4×10^{-3} kWh/block, and slightly less efficient than Nxt, which requires 1.4×10^{-6} kWh/block instead. Recall that while PoW requires more computation when the difficulty of the network increases (the number of hash computations for generating a block is exponential to the number of leading zeros l in the difficulty parameter, as it takes on average 2^l of trials to find the solution), the energy consumption of PoS protocols is independent of the difficulty.

Figure 6e shows the increase in energy consumption incurred when generating one block in PoW with respect to the difficulty parameter. We also marked the current energy consumption for Bitcoin and Ethereum network in the graph. At the time of writing, the total hashrate of Bitcoin and Ethereum networks reach 48.2×10^6 TH/s and 275 TH/s, targeting 600 s and 15 s of block time respectively. Equ factor ofipped with the most efficient devices Antminer S9i for Bitcoin mining (94 J/TH) and Antminer E3 for Ethereum mining (4 J/MH), it results in energy consumption of 7.9×10^6 kWh/block and 4.6×10^3 kWh/block, respectively. Finally, in Figure 6f we report the energy consumption of Bitcoin and Ethereum over the last three years (July 2015–July 2018). Recall that both protocols are PoW-

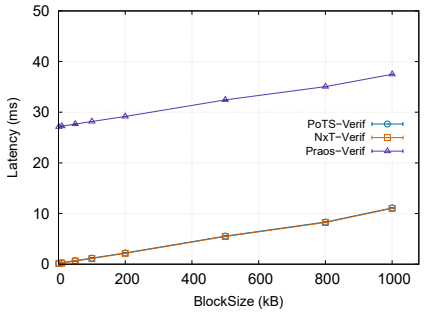
[10] We assume only one node is elected for the sake of simplicity.

[11] According to https://bitnodes.earn.com/dashboard/(accessed 04.09.2018).

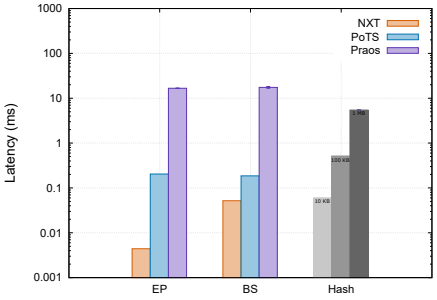
[12] We use the *thermal design power (TDP)* as an upper bound on the energy consumption of an Intel CPU. The TDP of Intel i7 is on average 100W according to https://en.wikipedia.org/wiki/List_of_CPU_power_dissipation_figures#Intel_Core_i7 (accessed 04.09.2018).



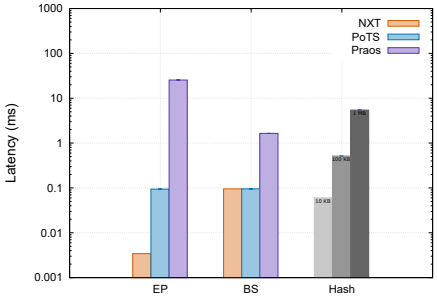
(a) Block generation latency



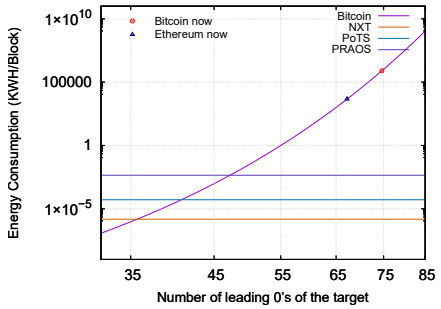
(b) Block verification latency



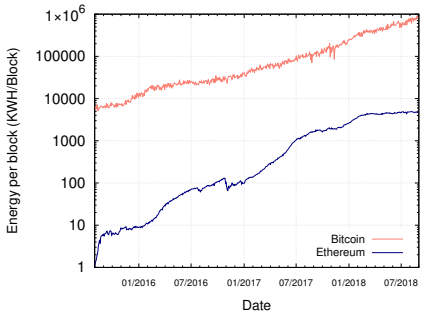
(c) Block generation latency: close-up



(d) Block verification latency: close-up



(e) Energy consumption vs. difficulty



(f) Energy consumption of Bitcoin and Ethereum over time

Fig. 6: Performance evaluation results for PoTS, Praos, and Nxt protocols. Each data point in our plots is averaged over 100 independent measurements; where appropriate, we include the corresponding 95% confidence interval. EP stands for "eligibility proof", BS for "block signature", and Hash for "hash over the transaction set".

based, and hence the difficulty is adjusted as the hashrate of the network increases. As a results, both cryptocurrencies exhibit an exponential growth in energy consumption, as it is visible in Figure 6f. This is in sharp contrast with PoS-based blockchains, for which the energy consumption remains the same over time.

Summary: The results of our performance evaluation demonstrate that PoTS achieves similar performance as Nxt, and that both protocols outperform Ouroboros Praos. Regarding block generation, for small block sizes PoTS is more than 80 times faster than Ouroboros Praos, and incurs an overhead of about 5 times compared to Nxt. Further, PoTS maintains a speed-up over Ouroboros Praos by a factor of 3 even for large block sizes while reaching the same speed of Nxt. Similarly, for block verification PoTS is more than 130 times faster than Ouroboros Praos and it takes about double the time required by Nxt for small block sizes; for large block sizes, it is again 3 times faster than Ouroboros Praos and as fast as Nxt. In addition, all the three PoS protocols are considerably more energy efficient than PoW-based Bitcoin and Ethereum, the difference in power consumption being between 6 and 12 orders of magnitude.

therefore, and in light of the security analysis presented in section V, PoTS offers a strong alternative to currently deployed protocols in terms of performance while guaranteeing strictly stronger security.

VII. Related Work

In this section, we discuss closely related work on consensus and proof of stake protocols.

A. Byzantine Fault Tolerant protocols

Byzantine Fault-Tolerant (BFT) consensus protocols focus on solving the Byzantine General's Problem [61] in distributed systems. PBFT [62] is the first practical BFT protocol working in *asynchronous* networks. It provides guarantees on safety and liveness assuming no more than $\frac{1}{3}$ of the nodes are faulty. Namely, all benign nodes would agree on a total order for the executed requests, and should not be stalled given that the messages cannot be delayed indefinitely. MinBFT [63] and CheapBFT [64] guarantee the same properties while incurring less communication complexity by leveraging monotonic counters in TEE. By assigning sequence numbers to the requests via monotonic counters, even faulty nodes cannot send messages equivocally, thus reducing the communication rounds. FastBFT [65] further reduces the communication costs by efficiently aggregating the votes from nodes via secret-sharing scheme. Nevertheless, these BFT protocols still do not scale well when the number of nodes exceeds a thousand. Moreover, all of them assume a static configuration of the network, i.e., the network is composed of a fixed number of fully connected nodes via authenticated links. This restricts BFT protocols from being adopted by *public* distributed ledgers.

B. Virtual Mining and Proof of Stake consensus

Proof of Elapsed Time (PoET) [66] is one of the first public blockchain consensus protocols that leverages TEEs. To regulate the leader election process, each node relies on the TEE to wait a random period of time before attempting to generate the next block. The node who is able to generate a block first thus becomes the leader. To ensure that nodes truly picks at random the time interval, and then wait sufficiently long, the protocol relies on the trusted timestamping functionality provided by Intel SGX. The security of PoET requires that no adversary acquires the majority of TEE processors, and that TEEs cannot being compromised. Based on the same security assumptions, Proof of Luck (PoL) [67] selects block leaders based on the highest random number generated by TEE. Nodes are required to first wait for a period of time before requesting the TEE to generate the "lucky" number and sign the block.

In contrast to PoET and PoL, which select block leaders uniformly at random from the set of validators, PoTS weighs validators depending on their stake. This distinction is reflected in the modeling assumption to protect against Sybil attacks, which relies on honest majority of validators (and hence of TEE devices) in the case of PoET and PoL, while it requires honest majority of stake for the security of PoTS. While in some scenarios it may be reasonable to assume that no adversary controls the majority of TEE devices, this assumption is hard to justify, for instance, in a relatively small network. Therefore, we argue that relying on honest majority of stake may be a safer choice.

Ouroboros [22] relies on a public leader-election function that randomly selects leaders depending on the recent stake distribution among validators. Importantly, the randomness for the leader election is generated via an MPC coin-tossing protocol, and hence it is provably unbiased as long as sufficiently many protocol participants are honest. In particular, Ouroboros provably prevents stake-grinding attacks. The security analysis of Ouroboros does not explicitly handle posterior corruption, and mitigates long-range attacks by assuming checkpoints.

Ouroboros Praos [21] (henceforth Praos) is based on a similar design of Ouroboros, however, it uses a private leader-election function, which in turns relies on verifiable random functions (VRFs) [68], to achieve security in the presence of fully adaptive corruptions. Praos further employs a (forward secure) key evolving signature scheme for block signing to also protect against posterior corruption of validators.

Notice that determining eligible validators using a VRF, as in Praos, is a specific instance of *cryptographic sortition*, introduced with Micali's Algorand protocol [24] to specifically weaken the effect of adaptive corruptions. Note also that PoTS's eligibility function follows a similar principle, i.e., the eligibility proof is privately computable and publicly verifiable. Due to the properties of the TEE, however, our solution does not need a VRF but only an unforgeable signature scheme. Indeed, while Praos relies on the pseudorandomness and uniqueness properties of the VRF (i) to ensure unpredictability of future leaders for all but the owner of the VRF secret key, and (ii) to make validators "commit" to exactly one VRF output per proof, PoTS achieves the above properties by computing the eligibility signature within the TEE, yielding a significantly more efficient solution [13] (as demonstrated by our performance analysis from Section VI).

Algorand [23], [24] is a PoS protocol targeting *final* consensus realization rather than eventual consensus. It uses cryptographic sortition (implemented through VRFs) to designate a committee of validators selected proportionally to their stake; the committee is then in charge of running a BFT protocol to agree on the next block. An advantage of Algorand is that it provides finality, meaning that forks do not occur (with high probability). On the downside, it requires $2/3$ of the stake being held by honest parties; moreover, the BFT protocol requires a few rounds of interaction among committee members, and it may end up generating an empty block, after a long consensus process, should the block proposer be malicious. This clearly reduces the block-production rate to a much lower value than for eventual-consensus PoS protocols.

Snow White [26] is a PoS protocol specifically targeting robustness under *sporadic participation*. In contrast, Ouroboros and Praos require continuous participation, and Algorand relax this by demanding "lazy" participation—meaning that validators need to be online when they are expected to serve in a committee. Our protocol instead is compatible with a sporadic-participation model, as its security relies on the majority of online stake being in honest hands. The Snow White protocol selects a committee from the set of validators depending on the stake distribution, and then picks the block leader within the committee by comparing publicly computable hits of the committee members—where the hits are computed in a way

[13]The signature is not pseudorandom, however, the randomness for the leader election is then extracted from it via hashing. In addition, the committing feature is directly taken care of because the TEE monotonic counter enforces at most one eligibility-proof request per round.

similar to Nxt [7]. The security analysis relies on checkpoints to mitigate posterior-corruption long-range attacks, and it does not consider nothing at stake attacks and grinding attacks.

VIII. Conclusion

In this work, we presented PoTS, a robust and efficient proof of stake protocol that uses Trusted Execution Environments, along with cryptographic techniques, to enforce sufficiently honest behavior of validators and offer protection in a strong corruption model. PoTS is specifically designed to prevent nothing at stake, grinding, and long-range attacks under realistic assumptions and, as our security analysis asserts, it retains its security even if some TEE platforms are compromised.

When comparing the performance of PoTS against two popular PoS protocols, Nxt and Ouroboros Praos, the former being very efficient but insecure, the latter offering provably-secure guarantees, we showed that our solution outperforms Ouroboros Praos and reaches similar efficiency as Nxt, thus offering an excellent tradeoff between security and performance.

The modular design of PoTS, highlighted in our security analysis, motivates a generic methodology leveraging TEEs to enhance the security of a large class of proof-of-stake protocols in the sense of thwarting all the above-mentioned attacks. We believe that such a generic transformation would turn any sortition-based protocol into a slightly less efficient one which is robust against nothing at stake, grinding, and long-range attacks. We therefore hope that our results motivate further research in this area.

References

[1] A. de Vries, "Bitcoin's Growing Energy Problem," *Joule*, vol. 2, no. 5, pp. 801–805, 2018.

[2] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," 2012, accessed June-2017. [Online]. Available: https://peercoin.net/assets/paper/peercoin-paper.pdf

[3] "Cloak posa v3.0 - a trustless, anonymous transaction system for cloakcoin," accessed June-2017. [Online]. Available: https://bravenewcoin.com/assets/Whitepapers/CloakCoin-posa3wp.pdf

[4] D. Pike, P. Nosker, D. Boehm, D. Grisham, S. Woods, and J. Marston, "Proof-of-stake-time whitepaper," accessed June-2017. [Online]. Available: https://www.vericoin.info/downloads/VeriCoinPoSTWhitePaper10May2015.pdf

[5] P. Vasin, "Blackcoin's proof-of-stake protocol v2," accessed June-2017. [Online]. Available: https://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf

[6] "Novacoin - proof of stake," accessed June-2017. [Online]. Available: https://github.com/novacoin-project/novacoin/wiki/Proof-of-stake

[7] N. Wiki, "Whitepaper:nxt — nxt wiki," 2016, accessed June-2017. [Online]. Available: https://nxtwiki.org/mediawiki/index.php?title=Whitepaper:Nxt

[8] F. Schuh and D. Larimer, "Bitshares 2.0: General overview," accessed June-2017. [Online]. Available: http://docs.bitshares.org/_downloads/bitshares-general.pdf

[9] "Nothing at stake attack ethereum." [Online]. Available: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs#what-is-the-nothing-at-stake-problem-and-how-can-it-be-fixed

[10] "Long range attack ethereum." [Online]. Available: https://blog.ethereum.org/2014/05/15/long-range-attacks-the-serious-problem-with-adaptive-proof-of-work/

[11] "Grinding attack ethereum." [Online]. Available: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs#how-does-validator-selection-work-and-what-is-stake-grinding

[12] V. Buterin, "Slasher: A punitive proof-of-stake algorithm," accessed June-2017. [Online]. Available: https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/

[13] V. Zamfir, "Introducing casper "the friendly ghost"," accessed June-2017. [Online]. Available: https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/

[14] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy*, 2019.

[15] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018.

[16] P. Gazi, A. Kiayias, and A. Russell, "Stake-bleeding attacks on proof-of-stake blockchains," *IACR Cryptology ePrint Archive*, vol. 2018, p. 248, 2018.

[17] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," *IEEE Security & Privacy*, vol. 12, no. 4, pp. 29–37, 2014.

[18] W. Li, S. Andreina, J. Bohli, and G. Karame, "Securing proof-of-stake blockchain protocols," in *DPM/CBT@ESORICS*, ser. Lecture Notes in Computer Science, vol. 10436. Springer, 2017, pp. 297–315.

[19] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 9057. Springer, 2015, pp. 281–310.

[20] R. Pass and E. Shi, "Fruitchains: A fair blockchain," in *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, E. M. Schiller and A. A. Schwarzmann, Eds. ACM, 2017, pp. 315–324. [Online]. Available: http://doi.acm.org/10.1145/3087801.3087809

[21] B. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 10821. Springer, 2018, pp. 66–98.

[22] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," Cryptology ePrint Archive, Report 2016/889, 2016. http://eprint. iacr. org/2016/889, Tech. Rep., 2016.

[23] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies." [Online]. Available: https://people.csail.mit.edu/nickolai/papers/gilad-algorand-eprint.pdf

[24] J. Chen and S. Micali, "Algorand: the efficient and democratic ledger," *arXiv preprint arXiv:1607.01341*, 2016.

[25] "Reaching Agreement in the Presence of Faults," vol. 27, pp. 228–234, 1980. [Online]. Available: http://doi.acm.org/10.1145/322186.322188{%}5Cnhttp://dl.acm.org/ft{_}gateway.cfm?id=322188{&}type=pdf

[26] I. Bentov, R. Pass, and E. Shi, "Snow white: Provably secure proofs of stake." *IACR Cryptology ePrint Archive*, vol. 2016, p. 919, 2016.

[27] V. Buterin, "Validator ordering and randomness in pos." [Online]. Available: http://vitalik.ca/files/randomness.html

[28] M. J. Fischer, N. A. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[29] C. Dwork, N. A. Lynch, and L. J. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[30] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 10211, 2017, pp. 643–673.

[31] R. Pass and E. Shi, "Thunderella: Blockchains with optimistic instant confirmation," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 10821. Springer, 2018, pp. 3–33.

[32] Intel, "Software sealing policies – intel® software guard extensions developer guide," 2017. [Online]. Available: https://software.intel.com/en-us/documentation/sgx-developer-guide

[33] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 2421–2434.

[34] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Using sgx to conceal cache attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2017, pp. 3–24.

[35] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside sgx enclaves with branch shadowing," in *26th USENIX Security Symposium, USENIX Security*, 2017, pp. 16–18.

[36] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "Truspy: Cache side-channel information leakage from the secure world on arm devices." *IACR Cryptology ePrint Archive*, vol. 2016, p. 980, 2016.

[37] B. Kevin, R. Lashermes, J.-L. Lanet, H. Le Bouder, and A. Legay, "How trustzone could be bypassed: Side-channel attacks on a modern system-on-chip," in *Wistp'17, International Conference on Information Security Theory and Practice*, 2017.

[38] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "Armageddon: Cache attacks on mobile devices." in *USENIX Security Symposium*, 2016, pp. 549–564.

[39] P. Gaži, A. Kiayias, and A. Russell, "Stake-bleeding attacks on proof-of-stake blockchains," Cryptology ePrint Archive, Report 2018/248, 2018, https://eprint.iacr.org/2018/248.

[40] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *ACM Conference on Computer and Communications Security*. ACM, 1993, pp. 62–73.

[41] K. D. W, "Digital signature algorithm," 1993, uS Patent 5,231,668.

[42] M. Fersch, E. Kiltz, and B. Poettering, "On the provable security of (EC)DSA signatures," in *ACM Conference on Computer and Communications Security*. ACM, 2016, pp. 1651–1662.

[43] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004, pp. 132–145.

[44] E. Brickell and J. Li, "Enhanced privacy id: A direct anonymous attestation scheme with enhanced revocation capabilities," in *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. ACM, 2007, pp. 21–30.

[45] ——, "Enhanced privacy id from bilinear pairing." *IACR Cryptology ePrint Archive*, vol. 2009, p. 95, 2009.

[46] TCG, "Monotonic counter – tpm main part 1 design principle (specification v1.2)," 2011. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-1-Design-Principles_v1.2_rev116_01032011.pdf

[47] "Create monotonic counter – intel® software guard extensions sdk." [Online]. Available: https://software.intel.com/en-us/sgx-sdk-dev-reference-sgx-create-monotonic-counter

[48] C. Soriente, G. Karame, W. Li, and S. Fedorov, "ReplicaTEE: Enabling Seamless Replication of SGX Enclaves in the Cloud," *CoRR*, 2018. [Online]. Available: https://arxiv.org/abs/1809.05027

[49] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun, "ROTE: rollback protection for trusted execution," in *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, E. Kirda and T. Ristenpart, Eds. USENIX Association, 2017, pp. 1289–1306. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/matetic

[50] G. Itkis, "Forward security, adaptive cryptography: Time evolution," 2004.

[51] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, J. Jung and T. Holz, Eds. USENIX Association, 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[52] R. Strackx and F. Piessens, "Ariadne: A minimal approach to state continuity," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 875–892. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/strackx

[53] B. Parno, J. R. Lorch, J. R. Douceur, J. Mickens, and J. M. McCune, "Memoir: Practical state continuity for protected modules," in *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE, 2011, pp. 379–394.

[54] R. Strackx, B. Jacobs, and F. Piessens, "Ice: A passive, high-speed, state-continuity scheme," in *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM, 2014, pp. 106–115.

[55] Wikipedia, "Spectre (security vulnerability)." [Online]. Available: https://en.wikipedia.org/wiki/Spectre_(security_vulnerability)

[56] "Documentation - Sawtooth v1.0.5." [Online]. Available: https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html#z-test

[57] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. van Renesse, "REM: resource-efficient mining for blockchains," in *USENIX Security Symposium*. USENIX Association, 2017, pp. 1427–1444.

[58] G. Itkis and L. Reyzin, "Forward-secure signatures with optimal signing and verifying," in *Advances in Cryptology — CRYPTO 2001*, J. Kilian, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 332–354.

[59] Y. Dodis and A. Yampolskiy, "A verifiable random function with short proofs and keys," in *Public Key Cryptography - PKC 2005*, S. Vaudenay, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 416–431.

[60] "Bilinear pairing library," https://godoc.org/golang.org/x/crypto/bn256.

[61] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[62] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[63] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo, "Efficient byzantine fault-tolerance," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 16–30, 2013.

[64] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel, "Cheapbft: resource-efficient byzantine fault tolerance," in *Proceedings of the 7th ACM european conference on Computer Systems*. ACM, 2012, pp. 295–308.

[65] J. Liu, W. Li, G. Karame, and N. Asokan, "Scalable byzantine consensus via hardware-assisted secret sharing," *IEEE Transactions on Computers*, 2018.

[66] "Poet 1.0 specification." [Online]. Available: https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/poet.html

[67] M. Milutinovic, W. He, H. Wu, and M. Kanwal, "Proof of luck: an efficient blockchain consensus protocol," in *SysTEX@Middleware*. ACM, 2016, pp. 2:1–2:6.

[68] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE, 1999, pp. 120–130.