Securing Proof-of-Stake Blockchain Protocols

Wenting Li, Sébastien Andreina, Jens-Matthias Bohli, and Ghassan Karame

NEC Laboratories Europe, Germany
firstname.lastname@neclab.eu

Abstract. Proof-of-Stake (PoS) protocols have been actively researched for the past five years. PoS finds direct applicability in open blockchain platforms and has been seen as a strong candidate to replace the largely inefficient Proof of Work mechanism that is currently plugged in most existing open blockchains. Although a number of PoS variants have been proposed, these protocols suffer from a number of security shortcomings; for instance, most existing PoS variants suffer from the *nothing at stake* and the *long range* attacks which considerably degrade security in the blockchain.
In this paper, we address these problems and we propose two PoS protocols that allow validators to generate at most one block at any given "height"—-thus alleviating the problem of nothing at stake and preventing attackers from compromising accounts to mount long range attacks. Our first protocol leverages a dedicated digital signature scheme that reveals the identity of the validator if the validator attempts to work on multiple blocks at the same height. On the other hand, our second protocol leverages existing pervasive Trusted Execution Environments (TEEs) to limit the block generation requests by any given validator to a maximum of one at a given height. We analyze the security of our proposals and evaluate their performance by means of implementation; our evaluation results show that our proposals introduce tolerable overhead in the block generation and validation process when compared to existing PoS protocols.

1 Introduction

The blockchain is gaining increasing attention nowadays motivated by the wide success of the Bitcoin cryptocurrency. To reach distributed agreement, the blockchain relies on consensus protocols which ensure that all nodes in the network share a consistent view on a common distributed ledger. Most existing blockchain systems rely on Bitcoin's Proof-of-Work (PoW) to reach network consensus in permission-less systems that do not require the knowledge of nodes' identities. However, PoW has been often criticized for its huge waste of energy; for instance, it is estimated that Bitcoin miners can consume as much electricity as Ireland in 2005 [16].

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 resources such as the

stake of a node in order to perform leader election and maintain consensus in the network. Since the mining resources are virtual, PoS-based consensus process is instant and results in negligible costs.

Nevertheless, although many PoS variants have been proposed [7, 12, 15, 17–19, 21], PoS is still not widely deployed in existing blockchains. Namely, in spite of their efficiency, PoS-powered blockchains still account for less than 2% 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 the *nothing at stake* and the *long-range* 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. The latter attack (commonly referred to as history attack) consists of an adversary that aims to alter 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 older accounts which no longer have any stake at the moment, but that have accrued majority stake at previous block height h; the attacker can construct a fork starting from block h leveraging these accounts.

To remedy these attacks, a number of proposals suggest the reliance on deposit-based PoS [3, 22] and checkpoints [12, 18, 19, 21]. Deposit-based PoS essentially requires each validator to make a deposit in the system; this deposit will be withdrawn by the system if the validator generates conflicting blocks, thus preventing nothing at stake attacks. Checkpoints, on the other hand, correspond to previous blocks up to which the blockchain does not allow forks. This limits the impact of the long-range attack to some extent, as the earliest attack point has to be after the last checkpoint. Clearly, such solutions do not however completely prevent misbehavior in the blockchain.

In this paper, we address these problems and we propose two PoS variant protocols that are secure against the nothing at stake and the long-range attacks. Our first solution leverages a digital signature scheme that is directly linked to the registered identities of the nodes. In case blocks in parallel forks are mined by the same node, the private key of that node along with his identity will be immediately revealed. As such, this solution complements existing solutions in the area [3, 12, 18, 19, 21, 22] by embedding accountability in case of misbehavior but at the expense of relying on an identity manager. Our second solution solves this shortcoming and leverages Trusted Execution Environments (TEEs) to prevent validators from signing two blocks of the same height in parallel chains. We analyze the security of both solutions and we show that they can effectively prevent malicious validators from generating conflicting blocks with nothing at stake. We also implement prototypes derived from our proposals and evaluate their performance. Our results show that our first solution only introduces an additional ∼500 ms of latency and ∼19 KB of block header payload when generating and verifying the cryptographic proofs compared to existing PoS protocols. On the other hand, our second solution does not incur any meaningful overhead compared to Nxt's PoS protocol.

The remainder of the paper is organized as follows. We overview existing PoS protocols and the main intuition behind the nothing at stake attack and the long range attack in Section 2. We then introduce our solutions and analyze their security provisions in Sections 3 and 4. We evaluate the performance of our proposals in Section 5 and we conclude the paper in Section 6.

2 Background & Related Work

Blockchain is a distributed ledger technology based on a peer-to-peer network. Transactions are broadcast in the network for every node to verify. Most existing blockchains leverage consensus protocols allowing nodes to collaboratively maintain a common ledger of validated transactions.

Proof-of-Stake (PoS) is a consensus protocol dedicated for *open* blockchains— which feature open membership allowing any node to join the network. PoS defines a group of *validators* whose task is to propose the next transaction(s) to be included in the ledger. These proposals are broadcast in the network in the form of *blocks*. Blocks typically build on each other, thereby forming a chain of blocks— hence the blockchain.

Since there are multiple validators in the network, PoS introduces a computational problem for the validators to solve when generating a block to throttle the number of block proposals in the network. Typically, the block generation time is manipulated by the means of a *target* value that denotes the difficulty of the problem to be solved by the validators. Each validator who finds a PoS solution then includes the solution along with the proposed block as a *proof* that he is "eligible" to generate the block. Blocks are deemed correct if their proof is correct with respect to all correct transactions that they confirm. If multiple validators find a solution simultaneously, a *fork* occurs in the block chain.

2.1 Proof-of-Stake (PoS)

In what follows, we summarize the general operations on a validator to maintain consensus defined by a PoS consensus protocol:

$\mathsf{IsEligible}(blkhder, T, key_V, stake_V)$ This qualification function verifies, given the prepared block header $blkhder$ and the target T, whether the validator is eligible to generate the next block given the account information $(key_V, stake_V)$. This function aims to elect a *leader* among the validators to generate the next block.

$\mathsf{GenerateBlock}(blk, T)$ This routine refers to the block generation function. Given a block blk and target T, the validator first checks the predicate $\mathsf{IsEligible}$ and returns a proof prf whether the validator is eligible.

$\mathsf{ValidateBlock}(blk, T, prf)$ This routine corresponds to the block verification function. It returns true if the information in the block blk is correct and if the proof prf is valid for the predicate $\mathsf{IsEligible}$.

$\mathsf{Resolve}(fork_1, \ldots, fork_n)$ This routine is a fork resolution algorithm that returns a unique fork $fork_k$ to work on if multiple forks are detected.

As mentioned earlier, PoS constitutes one of the few workable candidates set to replace that largely inefficient PoW in the near future. PoS leverages virtual resources denoted by the stake of a validator to solve the computation problem. *Stakes* refer to the assets (or cryptocurrencies) owned by a node. The idea is that the more stake a validator has, the more likely he will find a solution to generate a block. Thus, PoS defines the predicate IsEligible as $f(blkhder, key_V) < T \cdot stake_V$, where $f(\cdot)$ is a deterministic function on the block header and the validator's account key. Recall that the account key and the amount of stake is publicly verifiable by all nodes in the network.

GenerateBlock returns *empty* if the validator's $\langle key_V, stake_V \rangle$ does not satisfy the statement of the IsEligible predicate; otherwise it returns a proof $prf = \langle Prf_e, Sig_b \rangle$, where Prf_e is the *eligibility proof* and Sig_b is the *block signature* of the validator. This also implies that validators do not need to search for the PoS solution exhaustively (as in PoW) since the solution only depends on the validator's account information. Meanwhile, ValidateBlock returns true if the proof prf is valid and the validator's account $\langle key_V, stake_V \rangle$ satisfies the IsEligible statement.

The community currently features a number of PoS variants. For instance, Peercoin [12], Cloakcoin [7], and Novacoin [15] use coin age as mining resources to generate blocks in their PoS protocols. Coin age is defined as the accumulated time that a node holds his stake before using them to generate a block. However, relying on coin age discourages nodes from actively participating in the consensus process, as nodes would have incentives to hoard the coins so that they have a better chance to generate a block. VeriCoin [17] uses "stake-time" based on coin age, which also takes into account the activity of the nodes in the network. The stake time starts to degrade at a certain point of time if the nodes do not participate in block generation with their stake. Blackcoin [19] and Nxt [21] only rely on the amount of stake in the consensus protocol. Nxt [21] uses a deterministic algorithm (IsEligible) to elect the leaders in order to mitigate the *grinding* vulnerability [4,10], where adversaries use computational resources to increase their probabilities of being elected as leaders. This leads to another issue that the leader of each valid block is predictable in the network, thus making it vulnerable to planned denial of service attack or selfish-mining strategies. Additionally, an adversary can still perform stake grinding by skipping an opportunity to create a block if he is able to increase his advantage over the future blocks [4]. Blackcoin [19] defines a "stake modifier" which periodically introduces some entropy to the eligibility test, but only limits the period where leaders are predictable until the next update of the stake modifier. BitShares [18] proposes Delegated Proof-of-Stake where the shareholders first vote for a group of delegated witnesses who then generate the blocks in round-robin fashion. Slasher [3] and Casper [22] are deposit-based PoS proposed by Ethereum. They require the nodes to submit a deposit in order to become a validator.

Snow white [2] offers robustness under sporadic node participation with epoch-based committees composed of selected active stakeholders. Ouroboros [11] increases the incentives for honest behavior with a novel reward mechanism.

It also prevents stake grinding while keeping the leaders unpredictable using verifiable secure multi-party computation. However, this approach requires the coordination amongst the validators in the network. Algorand [6, 9] is rather a Byzantine Agreement protocol [1] in a public blockchain setting that does not generate forks (with a high probability). Block proposers and voting committee members are elected based on their stake through Verifiable Random Function [13]. However, this protocol requires at least three rounds of Byzantine Agreement voting if the block proposal is honest assuming a voting committee size of 4000 nodes. If the block proposal is malicious, an empty block will still be generated after a long consensus process.

Example—Nxt's PoS: We now describe Nxt's PoS which emerges as one of the most popular PoS protocols.

In Nxt's PoS, each block contains two additional values: a *base target value* T_b and an *eligibility proof* (called *generation signature* in Nxt) Prf_e, which vary from block to block. Similar to the *difficulty* level used in PoW, the base target value aims to adjust the average block generation time to match the desired value, which is 60 seconds in Nxt. On the other hand, the eligibility proof Prf_e is used to check whether the current validator is eligible to generate the next block. It is computed as the hash value over the public key of validator's account and the eligibility proof of the previous block: $Prf_e^{(i)} = Hash(PubKey_V || Prf_e^{(i-1)})$.

Then, each validator has his own target value T based on the elapsed time $time_e$ since the last block and the current effective stake in his account: $T = T_b \cdot time_e \cdot stake_V$. Finally, the leading l bytes of the eligibility proof Prf_e are checked against the target T. To summarize, the predicate IsEligible is defined as follows:

$$Prefix(Hash(PubKey_V || Prf_e^{(i-1)}), l) < T_b \cdot time_e \cdot stake_V \tag{1}$$

GenerateBlock returns a proof $prf = \langle Prf_e, Sig_b \rangle$, where Sig_b is the *block signature* $Sig_b = Sign(PrivKey_V, blkhder)$.

To verify a block, nodes first check if the block signature is valid before re-evaluating the IsEligible predicate defined in Equation 1. Here, the block signature is required for nodes to verify validator's public key used in the eligibility proof as well as validator's stake.

In case multiple forks are detected, a *cumulative difficulty* value is defined for each block. This difficulty is computed based on the cumulative difficulty of the previous block and the base target value of the current block: $CD^{(i)} = CD^{(i-1)} + \frac{2^{64}}{T_b}$. Resolve returns a fork whose last block exhibits the highest cumulative difficulty.

2.2 The "Nothing at Stake" Attack

Despite the variety of PoS protocols, since generating a block in PoS is no more than generating one signature, validators have incentive to work on multiple forks. In other words, in order to maximize the benefits, validators could generate

conflicting blocks on all possible forks with nothing at stake. This problem is commonly referred to as the *nothing at stake* attack.

This attack slows down the consensus time in the network and thus reduces the efficiency of the system. Moreover, it results in blockchain forks which weaken the ability of the blockchain to resolve double spending attacks and other threats.

Notice that the aforementioned Nxt's PoS does not address the nothing at stake problem. Namely, validators can ignore the fork resolution algorithm and generate blocks on top of multiple forks. Moreover, since the eligibility proof is deterministic for each account, one can easily predict which validators will generate valid blocks in the future. This is often referred to as "transparent forging" and opens an additional attack surface to the blockchain, allowing the attackers to selectively nit-pick the next leader to compromise.

Ouroboros [11] introduces a new reward mechanism that incentivizes the validators to behave honestly. However, it only discourages opportunistic adversaries and cannot prevent targeted attacks that would benefit from (temporarily) forked blockchain such as double-spending attack. Slasher [3] proposes to address this attack by requiring validators to provide a deposit which will be locked for a period. In case conflicting blocks at the same height are signed by the same validator, the misbehaving validator will lose his deposit. In this way, the network punishes the validators who simultaneously create conflicting blocks on multiple forks. BitShares [18] adopts a similar approach to Slasher. Here, if a validator (witness) misbehaves, they will lose their ability to generate blocks in the future. Other deposit-based PoS protocols even penalize the validators if they are voting on the "wrong" fork—assuming that there is only one correct fork at all time. Nevertheless, all such countermeasures freeze a considerable amount of stake in the network. In addition, in spite of the deposit-based mechanism, malicious validators can still profit from targeted attacks as they can create conflicting blocks with double-spending transactions whose value surpasses that of the deposit that they committed in the network.

2.3 The "Long Range" Attack

Long range attacks on PoS (also known as history attacks) refer to the case where an attacker tries to alter the blockchain history by creating a fork from an already generated block. While this attack in theory requires an attacker that controls the majority of stake in the network, long range attacks can be practically instantiated if the attacker controls/compromises accounts that have no stake at the moment, but have a large stake at some past block height h. For example, an account that had 30% stake at block height h and no stake at block height $h+1$ can still use his 30% stake to re-generate another block at height h.

This allows an attacker to create forks from past blocks that can overtake the current chain with (past) majority stake. This can be achieved by compromising the private keys of older accounts which no longer have any stake at the moment, but that have accrued majority stake at previous block height. Notice that accounts that exhibit zero stake might not be as protected as other *active* accounts—which would further facilitate this attack.

Blockchain (PoS)	Secure against	
	Nothing at Stake	Long Range Attack
Cloakcoin	×	×
Novacoin	×	×
Blackcoin	×	○
Peercoin	×	○
Nxt	×	○
Slasher	○	×
Vericoin	○	×
BitShares	○	○
Snow White	×	○
Ouroboros	×	○

Table 1: Resilience of existing PoS protocols against the nothing at stake and long range attacks. Here, ○ denotes refers to the case where a property is partially achieved.

At present, most existing countermeasures [2,12,18,19,21] against long range attack use *checkpoints* to limit the range of such attacks. A checkpoint refers to a block until which the blockchain is regarded as "finalized" and immutable. A number of PoS instantiations [12, 18, 19] rely on a centralized checkpoint server to define a correct chain periodically; on the other hand, Nxt [21] nodes do not accept a change to a fork that differs from a block more than 720 blocks old. Similarly, Snow White's nodes [2] do not accept a (longer) chain which modifies blocks "too far" in the past. However, these approaches require nodes to be synchronized; for instance, nodes that recently join the network can hardly distinguish which chain of the forks is the correct one.

Table 1 summarizes the security of existing PoS protocols against the nothing at stake and long range attacks. We see that the existing protocols are either insecure against these attacks, or only secure under specific conditions.

2.4 System Model

We assume a similar system model to Nxt (cf. Section 2.1). More specifically, we assume a peer-to-peer system where nodes commit their virtual resources in the system; and the stake of nodes is publicly verifiable by all participants in the network.

We rely on a trusted infrastructure with which the nodes interact off-chain. In our first solution, nodes trust identity providers and their certificates. On the other hand, in our second solution, we assume that nodes trust the implementation of the trusted hardware and TEE architectures. Notice that this model can be easily adopted by permission-based blockchain, where participants can quickly agree on a set of trusted infrastructures. However, we stress that our model equally applies to permissionless blockchain deployments by relying on existing public identity providers and/or TEE supported devices.

We further assume that nodes are rational and are only interested in increasing their advantage in the system without being identified. We assume that no node controls majority stake in the network presently; this however does not pre-

vent an adversary from compromising different accounts to accrue stake majority at a prior block height. We further assume that all participants are computationally bounded and cannot break the signature schemes or the guarantees provided by secure proof of knowledge schemes.

Whenever secure hardware and TEE are used, we assume that the adversary cannot compromise the TEE environment. For example, the adversary cannot extract the keys from the trusted applications within TEE or change the behavior of the trusted applications.

In what follows, we present two PoS solutions that are resilient against the aforementioned nothing at stake and long range attacks. To ease the presentation, we describe these solutions as extensions to the well-known Nxt PoS protocol (cf. Section 2.1).

3 Identity-based PoS

As mentioned earlier, the limitation of conventional deposit-based PoS is that even if a malicious node is caught creating conflicting blocks, the punishment of losing his deposit is not severe enough to totally discourage him. For example, an attacker can launch double-spending attacks that involve huge amount of stake through the forks he creates.

We start by showing that the combined use of deposit-based PoS and identity management in the network can alleviate this limitation. Namely, we describe a solution that requires the validators to submit their identity information in a privacy-preserving fashion, while their private keys used for generating blocks are bound to their committed identity information. We show how to conceal the nodes' identities—unless misbehavior happens.

The intuition behind our solution is to bind the randomness of the DSA signature of the block to the block height value. Therefore, once a validator creates two conflicting blocks at the same block height, a node can recover the validator's private key based on the block signatures and reveal the corresponding identity accordingly. Before describing our solution, we start by introducing the building blocks that we will use.

3.1 Building Blocks

Notations. Throughout this section, we denote p, q, g as the public parameters in DSA signature scheme, where p, q are the prime numbers such that $q|p-1$. g is a generator of order q in $\mathbb{Z}_p$. We further introduce a subgroup of $\mathbb{Z}_q$ with prime order w generated by a and h; while h's discrete logarithm to the base a is not known. Each node has one DSA key pair (x, y) for his account, where $0 < x < w$ is the private key and $y = g^x \mod p$ is the public key. Each node also publishes another public key $y' = a^x \mod q$ to the network.

Recall that DSA [20] signature is of the form of (r, s), where r is a random value whose seed k is picked randomly by the signer: $r = (g^k \mod p) \mod q$.

Algorithm 1 Proof of knowledge of the same discrete logarithm

Input: $y_1 = g^a$ and $y_2 = h^a$ where a is not revealed.
Output: A proof s that y_1 and y_2 have the same discrete log a.
1: $r \xleftarrow{\mathcal{R}} \mathbb{Z}_q^*$
2: $t_1 \leftarrow g^r$
3: $t_2 \leftarrow h^r$
4: $c \leftarrow H(t_1||t_2)$
5: $s \leftarrow r - c \cdot a \mod q$
6: Send s to the verifier
▷ Verifier accepts if $g^s \cdot y_1^c = t_1$ and $h^s \cdot y_2^c = t_2$

Recall that DSA requires that k should never be re-used for signing different messages, otherwise the private key is able to be recovered from these signatures.

We denote $r \xleftarrow{\mathcal{R}} \mathbb{Z}_n$ as randomly picking an element r from $\mathbb{Z}_n$. We use H for block height value included in each block header. We distinguish this with $H(\cdot)$ which is a cryptographic hash function.

Verifiable Random Function (VRF). VRF [13] refers to a pseudo-random function which provides a publicly verifiable proof that a given number is correctly generated based a public input and a private key. We need to use VRF in our block signature construction in order to *enforce* the validators to generate the randomness k which is bound to the block height H and prove to the others. Since DSA's security also relies on the fact that k is unpredictable, we require the VRF outputs to be unpredictable too.

Dodis et al. [8] have presented an efficient VRF scheme. However, since this scheme reveals k in the verification process, which is not allowed in DSA, we need to revise and extend it with Proofs of Knowledge [14] to prove that k is correctly generated according to the VRF scheme and is also used to compose r without revealing its value.

3.2 Protocol Specification

We now detail the various procedures used in our solution.
Identity Management: Our Identity-based PoS consensus protocol only allows registered validators who have committed their identity information to generate blocks. We therefore start by defining the setup phase in this protocol.

Our scheme requires a trusted identity provider that can issue identity-linked certificates for everyone, such as the credentials of the citizen's ePass. To commit his identity, a validator first generates his anonymous account key pair (x, y). The validator provides the public key y (along with the proof of knowledge of x) to the identity provider or uses his e-Identity card. He obtains in return an encrypted identity C_{id} under y along with a certificate $Cert_{IP}$ that binds key and encryption together, issued by the identity provider or the e-Identity card respectively. Then, the validator commits his encrypted identity $(C_{id}, Cert_{IP})$ to the blockchain network. A validator will be successfully registered with his account and identity (y, C_{id}) if the certificate from the identity provider is valid.

Notice that during this process, the validator's identity is only revealed to the identity provider or protected inside the e-Identity card. Therefore, committed identities do not violate nodes' privacy. Note that we do not restrict the way how a node looks up registered validators. We can achieve this either by introducing a trusted CA service in the blockchain to verify the committed identities and issue certificates for these validators, or each node can locally perform the validation.

Block Generation and Validation: The block generation and validation process is similar to Nxt's PoS as described in Section 2.1. Miners have to first create an eligibility proof Prf_e to check their eligibility of generating the current block. Subsequently, a block signature Sig_b is included in the created block to authenticate the validator.

In our protocol, in order to prevent a malicious node from signing conflicting blocks, validators construct the eligibility proof and the block signature as follows: a validator first generates the randomness k and $r = g^k$ (cf. Algorithm 2), along with the proof of knowledge $C_k, C_a, \pi_1, \pi_2, [\langle C_r^i, \pi_r^i \rangle]$ that k and r are correctly computed. Here, π_1 is a proof of knowledge of equality of discrete logarithm constructed from the Schnorr Proof of Knowledge as described in [5]. This allows us to prove that e.g., two elements $y_1 = g^a$ and $y_2 = h^a$ have the same discrete logarithm a without revealing a. We describe the non-interactive variant of this proof of knowledge in Algorithm 1. $[\langle C_r^i, \pi_r^i \rangle]$ is standard proof of knowledge with probabilistic result. We skip the details of how to construct π_2 as it is standard Schnorr Proof of Knowledge.

Subsequently, we use k and r to create the eligibility proof $Prf_e = Hash(r, y, Prf_e^{(H-1)})$. If the validator is eligible to generate the next block, the block signature is computed as follows: $Sig_b = Sign(x, blkheader) = (r, s)$.

To validate a block, nodes first verify the proof of knowledge to see if the block signature uses the correct randomness r given the block height H. Then, the block signature is verified similarly to the DSA signature verification process. Finally, the eligibility proof is checked against the IsEligible predicate function.

In case of forks, the resolution algorithm is similar to that of Nxt. Namely, each block contains a value of cumulative difficulty and nodes will adopt the fork chain that has accrued the largest cumulative difficulty (cf. Section 2.1).

3.3 Security Analysis

To show the security of our protocol, it suffices to show that a node can reveal the identity of a malicious validator based on his block signatures on conflicting blocks and all nodes are able to verify whether the block signatures are correctly constructed. Notice that since the nodes' private keys are bound to their identity, nodes will protect (e.g., they will not sell) their unused accounts—thus alleviating long-range attacks.

We first show that the block signatures generated by a malicious validator $M_{\mathcal{A}}$ on different blocks with the same block height H will recover the validator's private key, and therefore reveal his committed identity.

Algorithm 2 Generate k and r used for block signature

Input: Block height H, key pair (x, y')
Output: DSA randomness k, r, and the proof of knowledge $C_k, C_a, \pi_1, \pi_2, [\langle C_r^i, \pi_r^i \rangle]$

1: $k \leftarrow a^{\frac{1}{H+x}} \mod q$ ▷ random function on H
2: $r_0 \xleftarrow{\mathcal{R}} \mathbb{Z}_w$
3: $C_k \leftarrow k \cdot h^{r_0} \mod q$ ▷ commit to k
4: $C_a \leftarrow (C_k)^{H+x} \mod q$
5: $\pi_1 \leftarrow$ PK of same discrete log on C_a and $a^{H+x} = a^H \cdot y'$
▷ C_a is correctly computed as C_k^{H+x}
6: $\pi_2 \leftarrow$ Schnorr PK on $C_a \cdot a^{-1} = h^{r_0 \cdot (H+x)}$
▷ k is correctly computed as $a^{\frac{1}{H+x}}$

◇ π_1, π_2: VRF proof of k masked by r_0

7: $r \leftarrow (g^k \mod p) \mod q$
8: **for** $i \leftarrow 1$ to n **do**
9: $r_1^i \xleftarrow{\mathcal{R}} \mathbb{Z}_w$
10: $C_r^i \leftarrow H(r^{h^{r_0 + r_1^i}} \mod p)$ ▷ commit to r_1^i
11: **end for**
12: $challenge \leftarrow H(C_r^1 || \ldots || C_r^n)$
13: **for** $i \leftarrow 1$ to n **do**
14: **if** bit i of $challenge == 0$ **then**
15: $\pi_r^i \leftarrow r_0 + r_1^i \mod w$ ▷ verify if $C_r^i = H(r^{h^{\pi_r^i}})$
16: **else**
17: $\pi_r^i \leftarrow r_1^i$ ▷ verify if $C_r^i = H(g^{C_k \cdot h^{\pi_r^i}})$
18: **end if**
19: **end for**

◇ π_r^i: proof that $r = g^k$ with high probability

According to the construction of the block signature (r, s), r is deterministic and only depends on the value of the block height H and the validator's private key x. In other words, if $M_{\mathcal{A}}$ generates multiple blocks at the same height, the block signature component r will be the same. According to the DSA signature scheme, this will allow anyone to recover the validator's private key x from two signatures (r, s) and (r, s'). Since validator's identity is encrypted by his public key and committed to the network (cf. Identity Management in Section 3.2), any node that has recovered a validator's private key is able to reveal the corresponding identity. As a result, our block signature scheme discourages the validators from creating conflicting blocks. In addition, the proof of knowledge which comes along with the block signature proves that k and r are computed correctly according to the VRF function (cf. Algorithm 2 line 1) without revealing the value of k.

We use multiple proof of knowledge (PK) schemes to achieve this. First, the PK of equality of discrete logarithm π_1 proves that C_a is indeed C_k to the power of $(H + x)$ with the help of a^{H+x}. Then, the Schnorr PK π_2 (cf. line 4) together with π_1 proves that k is correctly computed as $a^{\frac{1}{H+x}}$. This is because if $C_a \cdot a^{-1}$

cannot cancel out a, then the prover is not able to know the discrete logarithm of $h^{r_0 \cdot (H+x)}$. Finally, based on the standard PK $[\langle C_r^i, \pi_r^i \rangle]$, the validator proves that r is correctly computed from k with the help of the committed value C_k, for $r^{h^{r_0}} = g^{C_k}$. Here, the proof of knowledge is repeated n times (line 13—19) that can prevent cheating with a high probability of $P = 1 - 2^{-n}$.

We also point out that the identity of an honest node will not be revealed, as there is at most one signature per block height, thus the randomness of the DSA signature is chosen securely. It is straightforward to show that if the attacker is able to retrieve the identity of an honest node, he has to acquire the private key of the node based on the DSA signatures.

Notice that according to [5], the VRF function $a^{\frac{1}{H+x}}$ outputs an unpredictable random number, therefore k can be used as the randomness for the DSA signature. Moreover, DSA also requires that the value of k should never be revealed. Our proof of knowledge schemes guarantee that the verifiers do not have the knowledge of k.

Finally, our eligibility proof is computed based on the randomness component r used in the block signature. This modification allows the eligibility proof remain deterministic as our r is constructed deterministically given a certain block height; moreover, it eliminates the problem of transparent forging (i.e., forger of each block is predictable in the network) as r is also unpredictable. Compared to Ouroboros [11], our construction achieves *grinding*-resilience without requiring the validators to coordinate in the network.

4 TEE-based PoS

In the previous section, we introduced a solution that strongly penalizes misbehavior by exposing the identities of validators. Although this solution can indeed deter misbehavior of the rational nodes in the network, it requires the reliance on an identity provider. In this section, we introduce another PoS protocol that drops this requirement and leverages Trusted Execution Environments (TEEs) to enforce security.

4.1 Protocol Specification

Specification of the Setup Stage: We require all validators in the network to be equipped with secure hardware to run trusted applications within TEE. TEEs 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 define an isolated environment running in parallel with the rich operating system. They additionally provide standard cryptographic functionalities and restrict the memory access from the hosting OS—thus ensuring secure execution for the code running inside TEE.

We require that the eligibility proof and the block signature (cf. Section 2.1) are generated by a trusted application within TEE. We denote it in the following

as the *trusted application*. To enforce this requirement, the validators need to prove to the network that they are hosting legitimate trusted application for block signing and their account keys are protected by the TEE and not accessible from outside of TEE. This can be achieved by distributing the trusted application as part of the client wallet application when nodes first join the network; and the account key pairs are only generated inside the TEE during the application initialization. The validators should also allow remote attestation for other nodes to verify the integrity of the deployed trusted application. Here, we require the platform certificates of validators' secure hardware to be publicly verifiable.

PoS Protocol Specification: The main intuition behind our solution is to use the trusted application to restrict the signing operations on the blocks. More specifically, we rely on TEE's monotonic counter to guarantee that there will be at most one block generated at each height of the block chain; recall that monotonic counters refer to increase-only registers that are resilient by design to replay attacks.

In our trusted application, we track the block height information of each signing request using monotonic counters. More specifically, we reserve two registers in the trusted application CTR_{ep} and CTR_{bs} which are implemented using monotonic counters. CTR_{ep} tracks the block height value submitted by the eligibility proof requests, while CTR_{bs} tracks the block height value submitted by the block signature requests.

To construct an eligibility proof, the validator submits the eligibility proof of the previous block along with the block height H of the current block to the trusted application. The latter checks whether $CTR_{ep} < H$; if so, the validator computes the eligibility proof as follows: $Prf_e = Sign(\text{“}ep\text{”}, x, H, Prf_e^{(H-1)})$,where x is the validator's private key; otherwise the request will be rejected. Meanwhile, the trusted application updates its register CTR_{ep} with the height information H. If validator's eligibility proof is lower than the target (cf. Equation 1), the validator is eligible to generate the next block and becomes a *leader*. The validator can therefore submit a request of block signature to the trusted application with the information of the block header[1]. Similarly, the trusted application checks if $CTR_{bs} < H$ is true, then a block signature is returned to the validator as $Sig_b = Sign(\text{“}bs\text{”}, x, blkhdr)$, and the register CTR_{bs} is updated. The block validation process is similar as in Nxt (cf. Section 2.1). Nodes first verify if the signatures are correct. Then, they check if the validator is eligible to generate the block given the eligibility proof in his block. Similarly, we keep the fork resolution algorithm the same as Identity-based PoS. Figure 1a depicts the interaction between the validator's wallet application (which is untrusted) and the trusted application for block signature. Here, a signing request with increasing block height can be processed successfully while a request with the same or smaller block height will be rejected.

Protecting against Chain Switching: Notice that this aforementioned process does not restrict the validators to only work on one particular fork of the

[1] The block height value is included in the block header

block chain. For example, while it is prohibited that a validator generates a block at height H on both fork F and F', it is allowed that a validator generates a block at height H on fork F and later on at height $H+1$ on fork F'. The validators are allowed to do so as in some cases they might initially work on a wrong fork (i.e., that differs from the output of the fork resolution algorithm) because of network partitioning; therefore the validators are allowed to later switch to work on another fork.

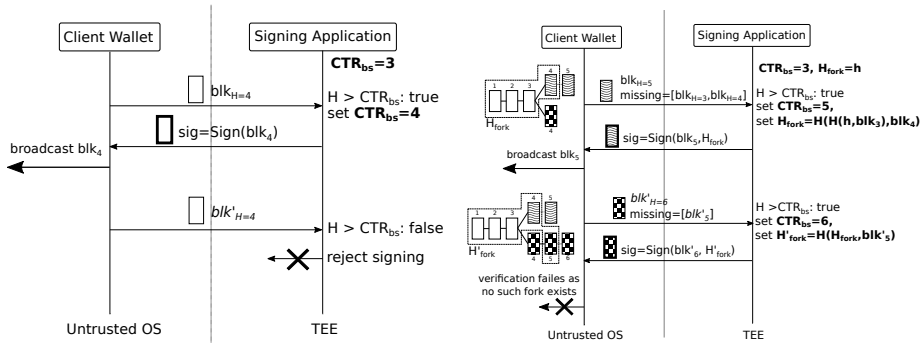
In other cases when the blockchain is able to define a *correct* chain to work on, we should further restrict the node to only work on one fork and cannot switch to other forks. To enforce this requirement, we extend our protocol to identify a fork by tracking the blocks of the fork. Specifically, we reserve a register H_{fork} in the trusted application to keep the accumulated hash of all the confirmed blocks of a fork that the validator submits in the block signature requests. This accumulated hash can uniquely identify a fork as it contains the information of all the confirmed blocks in the fork.

To generate the block signature, the validator submits block headers $[blkhdr_m, \ldots, blkhdr_n]$ that were confirmed since the last block signature request to the trusted application. The latter computes and updates the accumulated hash as $H^{(n)}_{fork} = H(\ldots(H(H^{(m-1)}_{fork}, blkhdr_m), \ldots, blkhdr_n)$. This value is then included in the block signature, so that all nodes in the network are able to verify it: $Sig_b = Sign(\text{“}bs\text{”}, x, H_{fork}, blkhdr_{n+1})$.

Figure 1b depicts an example where the validator requests the block signature to the trusted application. Here, we assume that the last signed block is at height $H = 3$. We also assume there are two forks F with blocks $[blk_1, blk_2, blk_3, blk_4, blk_5]$ and F' with blocks $[blk_1, blk_2, blk_3, blk'_4, blk'_5]$. In the first attempt, the validator submits a signing request for fork F at height $H = 5$. The validator sends the confirmed blocks since the last signing request blk_3 and blk_4 along with the request. To this point, the trusted application records the accumulated hash of fork F up to blk_4. In the second request, the validator switches to another fork F' and requests a block signature at height $H = 6$. Block blk'_5 is sent to the trusted application along with the request. As a result, the trusted application updates the register H_{fork} to include blk'_5 and adds this value to the block signature. However, this block signature will not be accepted by the network, since H'_{fork} identifies a fork composed by non-existing blocks $[blk_1, blk_2, blk_3, blk_4, blk'_5]$.

4.2 Security Analysis

Notice that the account private keys are protected by the TEE, which prevents an adversary from acquiring additional accounts to mount long-range attacks. Therefore, to prove that our TEE-based PoS is secure, it suffices to show that: *(i)* a validator is unable to acquire more than one eligibility proof or block signature for a block at a certain height, and *(ii)* a validator is unable to acquire valid block signatures for blocks on other forks if a validator has already worked on a different fork.



(a) Misbehaved validator requests to sign two blocks at the same height $H = 4$.

(b) Misbehaved validator requests to sign the second block for another fork.

We first show that validators can only acquire at most one eligibility proof or block signature for a block at a certain height. Since the account keys are unique and protected by the trusted application whose integrity is attested by the network, no one can generate a valid eligibility proof or block signature except the trusted application (recall that we assume that the adversary cannot compromise the TEE). For each signature request, the trusted application will first compare the block height with what was signed last time before advancing the monotonic counter. The TEE's monotonic counter guarantees that the recorded block height value is never repeated: an attacker can neither restart the counter nor replay a previous block height value. Therefore, TEE-based PoS ensures that a validator has only one chance to create an eligibility proof or a block signature for a block at a certain height. This additionally prevents a malicious validator from exhaustively searching the space of valid eligibility proof by changing the volatile fields of a block such as the transactions set and the timestamp. Moreover, recall that the forger of each block is predictable in the original Nxt's PoS. Here, we use digital signatures to generate eligibility proofs in an unpredictable manner—thus eliminating the problem of transparent forging.

We further show that our protocol extension prevents validators from generating blocks for different forks. Suppose any two forks $fork$ and $fork'$ contain two different confirmed blocks blk_m and blk'_m at height m. Given the previous guarantee, a validator can only generate one block signature for blocks at height m. We assume it is block blk_m and therefore blk_m is included in the register value H_{fork}. Later on, the validator would like to generate a block for $fork'$ at height n. However, since blk'_m can never be included in H_{fork} (as blk_m is already included in H_{fork}), the validator cannot generate a valid block signature that includes the information blk'_m for fork $fork'$, which is required for successful block validation.

5 Performance Evaluation

We implemented both PoS variants as well as Nxt's PoS protocol [21]. To ensure a fair comparison, we use in our implementation the same block structure as in Nxt; our protocols only differ in the algorithms to compute and verify a block.

5.1 Experimental Setup

Our implementation is based on Golang. We use Intel SGX to provide hardware security support and implement the trusted application of TEE-based PoS as an SGX enclave. In all solutions, we use SHA256 as the hash function. For Identity-based PoS, we implement our own DSA signature scheme for the block signature with security parameters of 256-bit w, 2048-bit q, 2072-bit p, and n=80 challenges for the proof of knowledge; for TEE-based PoS, we use 256-bit ECDSA as the signature scheme.

We deploy our validator on a server equipped with 8-Core Intel Xeon E3-1240 and 32 GB RAM. We vary the block size[2] from 1 KB to 1000 KB and measure the performance of signing and verifying a block.

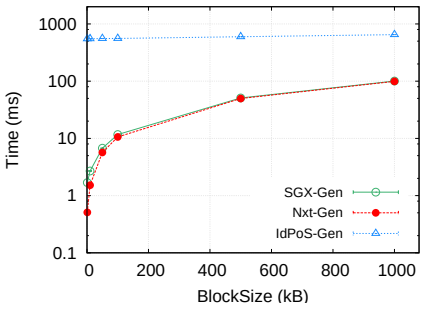
5.2 Performance Evaluation

We measure the block generation time and block verification time of each protocol with respect to the block size. For Identity-based PoS, we additionally compare the size overhead induced by the proof of knowledge.

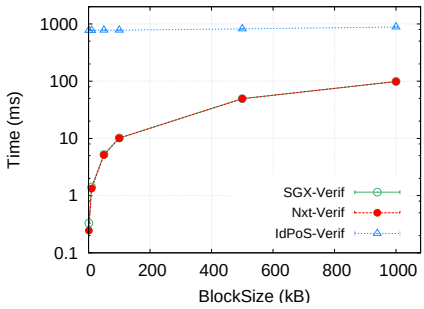
Impact on the Block Generation Time: We first measure the latency to generate a block for each protocol. Figure 2a shows that for small block sizes, Nxt is slightly faster than TEE-based PoS. For example, when the block size is 1KB, it takes 0.42 ms for Nxt to generate a block, compared to 1.7 ms for TEE-based PoS. This is due to the fact that TEE-based PoS requires context switching for the SGX enclave and leverages digital signature instead of hash functions to compute the eligibility proof. However, this difference becomes less pronounced as the block grows bigger; in these cases, the computation of the hashes dominates the time to generate a block. For instance, when the block size is as big as 500 KB, the block generation time for TEE-based PoS is equivalent to Nxt's (cf. Figure 2c).

While TEE-based PoS is almost as efficient as Nxt, the block generation process is however slower for Identity-based PoS. Even for small block sizes of 1 KB, Identity-based PoS requires 548 ms to generate a block. This overhead is mostly due to the generation of the proof of knowledge for the block signature; in this case, approximately 98% of the time is consumed in the generation of the proof. In addition, the size of the block header also increases dramatically due to the data field of the employed proof of knowledge schemes; the size grows to 19 KB for Identity-based PoS compared to only 200 B for Nxt.

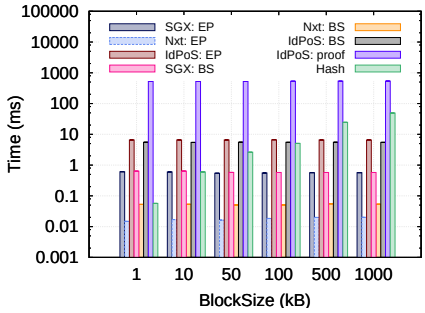
[2] We denote block size as size of the transaction set.



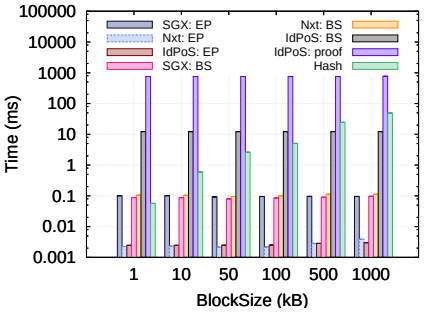
(a) Block generation time vs. the block size.



(b) Block verification time vs. the block size.



(c) Repartition of the latency incurred during the block generation.



(d) Repartition of the latency incurred during the block verification.

Fig. 2: Performance evaluation results. Each data point in our plots is averaged over 4000 independent measurements; where appropriate, we include the corresponding 95% confidence interval. EP stands for eligibility proof, BS for block signature, proof for proof of knowledge, and Hash for hash over the transaction set.

Impact on the Block Verification Time: We now assess the impact of our protocols on the resulting block verification process. Our evaluation results are depicted in Figure 2b and 2d.

Our results show that TEE-based PoS and Nxt require almost identical block verification times. For instance, when the block size is 1 KB, TEE-based PoS requires 0.34 ms while Nxt takes 0.28 ms. Recall that signature verification does not need to be performed inside the SGX enclave—which explains the comparable verification time for block signature. However, verifying eligibility proof is slightly slower in TEE-based PoS, as the eligibility proof used in TEE-based PoS consists of a digital signature rather than a cryptographic hash.

As expected, the block verification for Identity-based PoS incurs considerable verification times as the operation to verify the proof of knowledge is time consuming; our results show that it takes 772 ms to verify the proof of knowledge. For block sizes of 1000 KB, this sums up to 870 ms to verify a block, which is almost 8 times slower than Nxt.

6 Conclusion

Although a number of PoS consensus protocols have been proposed for open blockchain platforms in the past five years, these protocols still suffer from a number of security shortcomings which prevents their large scale adoption in existing open blockchains.

In this paper, we propose two PoS protocols that are secure against a number of threats including the nothing at stake attack and long range attack. The idea of both protocols is to restrict the validators to generate at most one block at a given block height. Our first protocol is a software-based solution with an enhanced signature scheme which binds the randomness of the signature to the block height value. Signing multiple blocks at the same height will thus reveal validator's private key as well as his identity. The limitation of this approach, however, is the storage and computation overhead, as the block header needs to include extra information over the proof of knowledge to verify that the signature randomness is correctly computed. Our second protocol is a hardware-based solution which relies on the tamper-resistant hardware and the trusted application for block generation. The trusted application records the information about the last issued blocks and prevents the validator from repeatedly generating an already generated block. We also extend our protocol to further prevent validators from working on different forks.

We implemented both protocols and evaluated their performance when compared to Nxt's PoS protocol. Our results show that the overhead incurred by TEE-based PoS is negligible compared to Nxt's PoS. On the other hand, by obviating the reliance on secure hardware support, Identity-based PoS results in more significant overhead when generating and verifying of the proofs of knowledge. We argue however that the performance of Identity-based PoS is still acceptable when compared to reasonable block creation time. We therefore hope that our results motivate further research in this area.

References

1. Reaching Agreement in the Presence of Faults 27, 228–234 (1980), http://doi.acm.org/10.1145/322186.322188{%}5Cnhttp://dl.acm.org/ft{_}gateway.cfm?id=322188{&}type=pdf
2. Bentov, I., Pass, R., Shi, E.: Snow white: Provably secure proofs of stake. IACR Cryptology ePrint Archive 2016, 919 (2016)
3. Buterin, V.: Slasher: A punitive proof-of-stake algorithm, https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/, accessed June-2017
4. Buterin, V.: Validator ordering and randomness in pos, http://vitalik.ca/files/randomness.html
5. Camenisch, J., Michels, M.: Proving in zero-knowledge that a number is the product of two safe primes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 107–122. Springer (1999)
6. Chen, J., Micali, S.: Algorand: the efficient and democratic ledger. arXiv preprint arXiv:1607.01341 (2016)

7. Cloak posa v3.0 - a trustless, anonymous transaction system for cloakcoin, https://bravenewcoin.com/assets/Whitepapers/CloakCoin-posa3wp.pdf, accessed June-2017
8. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: International Workshop on Public Key Cryptography. pp. 416–431. Springer (2005)
9. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies https://people.csail.mit.edu/nickolai/papers/gilad-algorand-eprint.pdf
10. Ethereum - prrof of stake faq - how does validator selection work, and what is stake grinding?, https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ#how-does-validator-selection-work-and-what-is-stake-grinding
11. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. Tech. rep., Cryptology ePrint Archive, Report 2016/889, 2016. http://eprint. iacr. org/2016/889 (2016)
12. King, S., Nadal, S.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake (2012), https://peercoin.net/assets/paper/peercoin-paper.pdf, accessed June-2017
13. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: Foundations of Computer Science, 1999. 40th Annual Symposium on. pp. 120–130. IEEE (1999)
14. Non-interactive zero-knowledge proof, https://en.wikipedia.org/wiki/Non-interactive_zero-knowledge_proof, accessed June-2017
15. Novacoin - proof of stake, https://github.com/novacoin-project/novacoin/wiki/Proof-of-stake, accessed June-2017
16. O'Dwyer, K.J., Malone, D.: Bitcoin mining and its energy footprint (2014)
17. Pike, D., Nosker, P., Boehm, D., Grisham, D., Woods, S., Marston, J.: Proof-of-stake-time whitepaper, https://www.vericoin.info/downloads/VeriCoinPoSTWhitePaper10May2015.pdf, accessed June-2017
18. Schuh, F., Larimer, D.: Bitshares 2.0: General overview, http://docs.bitshares.org/_downloads/bitshares-general.pdf, accessed June-2017
19. Vasin, P.: Blackcoin's proof-of-stake protocol v2, https://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper.pdf, accessed June-2017
20. W, K.D.: Digital signature algorithm (1993), uS Patent 5,231,668
21. Wiki, N.: Whitepaper:nxt — nxt wiki (2016), https://nxtwiki.org/mediawiki/index.php?title=Whitepaper:Nxt, accessed June-2017
22. Zamfir, V.: Introducing casper the friendly ghost, https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/, accessed June-2017