Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol

Aggelos Kiayias* Alexander Russell† Bernardo David‡ Roman Oliynykov§

December 21, 2016

Abstract

We present "Ouroboros," the first blockchain protocol based on *proof of stake* with rigorous security guarantees. We establish security properties for the protocol comparable to those achieved by the bitcoin blockchain protocol. As the protocol provides a "proof of stake" blockchain discipline, it offers qualitative efficiency advantages over blockchains based on proof of physical resources (e.g., proof of work). We showcase the practicality of our protocol in real world settings by providing experimental results on transaction processing time obtained with a prototype implementation in the Amazon cloud. We also present a novel reward mechanism for incentivizing the protocol and we prove that given this mechanism honest behavior is an approximate Nash equilibrium, thus neutralizing attacks such as selfish mining and block withholding.

1 Introduction

A primary consideration regarding the operation of blockchain protocols based on proof of work (PoW)—such as bitcoin [17]—is the energy required for their execution. At the time of this writing, generating a single block on the bitcoin blockchain requires a number of hashing operations exceeding 2^{60}, which results in striking energy demands. Indeed, early calculations placed the energy requirements of the protocol in the order of magnitude of a country; see, e.g., [19].

This state of affairs has motivated the investigation of alternative blockchain protocols that would obviate the need for proof of work by substituting it with another, more energy efficient, mechanism that can provide similar guarantees. It is important to point out that the proof of work mechanism of bitcoin facilitates a type of randomized "leader election" process that elects one of the miners to issue the next block. Furthermore, provided that all miners follow the protocol, this selection is performed in a randomized fashion proportionally to the computational power of each miner. (Deviations from the protocol may distort this proportionality as exemplified by "selfish mining" strategies [9, 23].)

A natural alternative mechanism relies on the notion of "proof of stake" (PoS). Rather than miners investing computational resources in order to participate in the leader election process, they instead run a process that randomly selects one of them proportionally to the *stake* that each possesses according to the current blockchain ledger.

*University of Edinburgh, IOHK. akiayias@inf.ed.ac.uk. Work partly supported by H2020 Project #653497 PANORAMIX.

†University of Connecticut. acr@cse.uconn.edu.

‡Aarhus University, IOHK, bernardo.david@iohk.io. Work partly supported by European Research Council Starting Grant 279447.

§IOHK, roman.oliynykov@iohk.io.

In effect, this yields a self-referential blockchain discipline: maintaining the blockchain relies on the stakeholders themselves and assigns work to them (as well as rewards) based on the amount of stake that each possesses as reported in the ledger. Aside from this, the discipline should make no further “artificial” computational demands on the stakeholders. In some sense, this sounds ideal; however, realizing such a proof-of-stake protocol appears to involve a number of definitional, technical, and analytic challenges.

Previous work. The concept of PoS has been discussed extensively in the bitcoin forum.[1] Proof-of-stake based blockchain design has been more formally studied by Bentov et al., both in conjunction with PoW [4] as well as the sole mechanism for a blockchain protocol [3]. Although Bentov et al. showed that their protocols are secure against some classes of attacks, they do not provide a formal model for analysing PoS based protocols or security proofs relying on precise definitions. Heuristic proof-of-stake based blockchain protocols have been proposed (and implemented) for a number of cryptocurrencies.[2] Being based on heuristic security arguments, these cryptocurrencies have been frequently found to be deficient from the point of view of security. See [3] for a discussion of various attacks.

It is also interesting to contrast a PoS-based blockchain protocol with a classical consensus blockchain that relies on a *fixed* set of authorities (see, e.g., [7]). What distinguishes a PoS-based blockchain from those which assume static authorities is that stake changes over time and hence the trust assumption evolves with the system.

Another alternative to PoW is the concept of *proof of space* [2, 8], which has been specifically investigated in the context of blockchain protocols [20]. In a proof of space setting, a “prover” wishes to demonstrate the utilization of space (storage / memory); as in the case of a PoW, this utilizes a physical resource but can be less energy demanding over time. A related concept is *proof of space-time* (PoST) [16]. In all these cases, however, an expensive physical resource (either storage or computational power) is necessary.

The PoS Design challenge. A fundamental problem for PoS-based blockchain protocols is to simulate the leader election process. In order to achieve a randomized election among stakeholders, entropy must be introduced into the system, and mechanisms to introduce entropy may be manipulated by the adversary. For instance, independently of the solution, an adversary controlling a set of stakeholders may choose to simulate the protocol execution trying different sequences of stakeholder participants so that it finds a favorable chain continuation that biases the leader election. To prevent this manipulation, honest stakeholders must be able to add sufficient entropy and counter any lookahead performed by the adversary.

Our Results. We present “Ouroboros”, a provably secure proof of stake system. To the best of our knowledge this is the first blockchain protocol of its kind with a rigorous security analysis. In more detail, our results are as follows.

First, we provide a model that formalizes the problem of realizing a PoS-based blockchain protocol. The model we introduce is in the spirit of [12], focusing on *persistence* and *liveness*, two formal properties of a robust transaction ledger. *Persistence* states that once a node of the system proclaims a certain transaction as “stable”, the remaining nodes, if queried and responding honestly,

[1]See “Proof of stake instead of proof of work”, Bitcoin forum thread. Posts by user “QuantumMechanic” and others. (`https://bitcointalk.org/index.php?topic=27787.0.`).

[2]A non-exhaustive list includes NXT, Neucoin, Blackcoin, Tendermint, Bitshares.

will also report it as stable. Here, stability is to be understood as a predicate that will be parameterized by some security parameter k that will affect the certainty with which the property holds. (E.g., "more than k blocks deep".) *Liveness* ensures that once an honestly generated transaction has been made available for a sufficient amount of time to the network nodes, say u time steps, it will become stable. The conjunction of liveness and persistence provides a robust transaction ledger in the sense that honestly generated transactions are adopted and become immutable. Our model is suitably amended to facilitate PoS-based dynamics.

Second, we describe a novel blockchain protocol based on PoS. Our protocol assumes that parties can freely create accounts and receive and make payments, and that stake shifts over time. We utilize a secure multiparty implementation of a coin-flipping protocol to produce the randomness for the leader election process. This distinguishes the approach from other previous solutions that either defined such values deterministically based on the current state of the blockchain or used collective coin flipping as a way to introduce entropy [3]. Also, unique to our approach is the fact that the system ignores round-to-round stake modifications. Instead, a snapshot of the current set of stakeholders is taken in regular intervals called *epochs*; in each such interval a secure multiparty computation takes place utilizing the blockchain itself as the broadcast channel. Specifically, in each epoch a set of randomly selected stakeholders form a committee which is then responsible for executing the coin-flipping protocol. The outcome of the protocol determines the set of next stakeholders to execute the protocol in the next epoch as well as the outcomes of all leader elections for the epoch.

Third, we provide a set of formal arguments establishing that no adversary can break persistence and liveness. Our protocol is secure under a number of plausible assumptions: (1) the network is highly synchronous, (2) the majority of the selected stakeholders is available as needed to participate in each epoch, and (3) the stakeholders do not remain offline for long periods of time. At the core of our security arguments is a probabilistic argument regarding a combinatorial notion of "forkable strings" which we formulate, prove and verify experimentally.

Fourth, we turn our attention to the incentive structure of the protocol. We present a novel reward mechanism for incentivizing the participants to the system which we prove to be an (approximate) Nash equilibrium. In this way, attacks like *block withholding* and *selfish-ming*, [9, 23], are mitigated by our design.

Fifth, we introduce a stake delegation mechanism that can be seamlessly added to our blockchain protocol. Delegation is particularly useful in our context as we would like to allow our protocol to scale even in a setting where the set of stakeholders is highly fragmented. In such cases, the delegation mechanism can enable the stakeholder to delegate her "voting rights", i.e., the right of participating in the committees running the leader selection protocol in each epoch. As in *liquid democracy*, (a.k.a. delegative democracy [11]), stakeholders have the ability to revoke their delegative appointment when they wish independently of each other.

Sixth, given our model and protocol description we also explore how various attacks considered in practice can be addressed within our framework. Specifically, we discuss double spending attacks, transaction denial attacks, 51% attacks, nothing-at-stake, desynchronization attacks and others.

Finally, we survey our prototype implementation and report on benchmark experiments run in the Amazon cloud that showcase the power of our proof of stake blockchain protocol in terms of performance.

Paper overview. We lay out the basic model in Sec. 2. To simplify the analysis of our protocol, we present it in four stages that are described in Section 3. In short, in Sec. 4 we describe and analyze the protocol in the static setting; we then transition to the dynamic setting in Sec. 5.

Our incentive mechanism and the equilibrium argument is presented in Sec. 6 and our delegation mechanism in Sec. 7. Following this, in Sec. 8 we discuss the resilience of the protocol under various particular attacks of interest. Finally, in Sec. 9 we discuss performance results obtained from a prototype implementation running in the Amazon cloud.

2 Model

Time, slots, and synchrony. We consider a setting where time is divided into discrete units called *slots*. A ledger, described in more detail below, associates with each time slot (at most) one ledger *block*. Players are equipped with (roughly synchronized) clocks that indicate the current slot. This will permit them to carry out a distributed protocol intending to collectively assign a block to this current slot. In general, each slot sl_r is indexed by an integer $r \in \{1, 2, \ldots\}$, and we assume that the real time window that corresponds to each slot has the following properties.

- The current slot is determined by a publicly-known and monotonically increasing function of current time.
- Each player has access to the current time. Any discrepancies between parties' local time are insignificant in comparison with the length of time represented by a slot.
- The length of the time window that corresponds to a slot is sufficient to guarantee that any message transmitted by an honest party at the beginning of the time window will be received by any other honest party by the end of that time window (even accounting for small inconsistencies in parties' local clocks). In particular, while network delays may occur, they never exceed the slot time window.

Transaction Ledger Properties. A protocol Π implements a robust transaction ledger provided that the ledger that Π maintains is divided into "blocks" (assigned to time slots) that determine the order with which transactions are incorporated in the ledger. It should also satisfy the following two properties.

- **Persistence.** Once a node of the system proclaims a certain transaction tx as *stable*, the remaining nodes, if queried, will either report tx in the same position in the ledger or they will not report as stable any transaction in conflict to tx. Here the notion of stability is a predicate that is parameterized by a security parameter k; specifically, a transaction is declared *stable* if and only if it is in a block that is more than k blocks deep in the ledger.
- **Liveness.** If all honest nodes in the system attempt to include a certain transaction, then after the passing of time corresponding to u slots (called the transaction confirmation time), all nodes, if queried and responding honestly, will report the transaction as stable.

In [15] it was shown that persistence and liveness can be derived from the following three elementary properties provided that protocol Π derives the ledger from a data structure in the form of a blockchain.

- **Common Prefix (CP); with parameters $k, \ell \in \mathbb{N}$.** The chains $\mathcal{C}_1, \mathcal{C}_2$ possessed by two external observers at the onset of the slots $sl_1 < sl_2$ with sl_2 at most ℓ slots ahead of sl_1, are such that $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$, where $\mathcal{C}_1^{\lceil k}$ denotes the chain obtained by removing the last k blocks from $\mathcal{C}_1$, and $\preceq$ denotes the prefix relation.

- **Chain Quality (CQ); with parameters $\mu \in (0,1] \to (0,1]$ and $k \in \mathbb{N}$.** For any subset S of (possibly malicious) stakeholders with relative stake α and any portion of length k in a chain possessed by an honest party at the onset of a certain slot, the ratio of blocks originating from members of S can be at most $\mu(\alpha)$.
- **Chain Growth (CG); with parameters $\tau \in (0,1], s \in \mathbb{N}$.** Consider the chains $\mathcal{C}_1, \mathcal{C}_2$ possessed by two honest parties at the onset of two slots sl_1, sl_2 with sl_2 at most s slots ahead of sl_1. Then it holds that $\text{len}(\mathcal{C}_2) - \text{len}(\mathcal{C}_1) \geq \tau \cdot s$. We call τ the speed coefficient.

Some remarks are in place. Regarding the notion of common prefix, we point out that if $\ell = 0$ this notion coincides with the common prefix property as originally formulated in [12]. A stronger formulation of common prefix would set ℓ to be the lifetime of the system itself; this amounts to strong common prefix [15] that includes the self-consistence property of [22]. Positing a bound ℓ smaller than the system's lifetime indicates that forks deeper than k blocks might be feasible in the chains of honest parties (or even of the same party) if the parties are observed between two rounds that are more than ℓ slots away. This relaxation is necessary to be able to prove the common prefix property in the PoS setting. With foresight, anticipating the implication of persistence from common prefix, we will need the additional assumption that no honest stakeholder is offline for too many rounds.

Regarding chain quality, the function μ satisfies $\mu(\alpha) \geq \alpha$ for protocols of interest. In an ideal setting, μ would be the identity function: in this case, the percentage of malicious blocks in any sufficiently long chain segment is proportional to the cumulative stake of a set of (malicious) stakeholders.

It is worth noting that for bitcoin we have $\mu(\alpha) = \alpha/(1-\alpha)$, and this bound is in fact tight—see [12], which argues this guarantee on chain quality. The same will hold true for our protocol construction.

Finally chain growth concerns the rate at which the chain grows (for honest parties). As in the case of bitcoin, the *longest* chain plays a preferred role in our protocol; this provides an easy guarantee of chain growth.

Security Model. We adopt the model introduced by [12] for analysing security of blockchain protocols enhanced with an ideal functional $\mathcal{F}$. We denote by $\mathsf{VIEW}^{P,\mathcal{F}}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa)$ the view of party P after the execution of protocol Π with adversary $\mathcal{A}$, environment $\mathcal{Z}$, security parameter κ and access to ideal functionality $\mathcal{F}$. We note that multiple different "functionalities" can be encompassed by $\mathcal{F}$.

We stress that contrary to [12], our analysis is in the "standard model", and without a random oracle functionality. Nevertheless we do employ a "diffuse" functionality that operates in a similar fashion with the additional property of sender anonymity. Specifically the operation of diffusion is as follows.

- Diffuse functionality. It maintains a incoming string for each party U_i that participates. A party, if activated, is allowed at any moment to fetch the contents of its incoming string hence one may think of this as a mailbox. Furthermore, parties can give the instruction to the functionality to diffuse a message. The functionality keeps rounds and all parties are allowed to diffuse once in a round. Rounds do not advance unless all parties have diffused a message. The adversary, when activated, can also interact with the functionality and is allowed to read all inboxes and all diffuse requests and deliver messages to the inboxes in any order it prefers. At the end of the round, the functionality will ensure that all inboxes contain all messages that have been diffused (but not necessarily in the same order they have been requested to be diffused). In order to preserve sender anonymity the functionality will strip the

- Key and Transaction functionality. The key registration functionality is initialized with n users, $U_1, \ldots, U_n$ and their respective stake $s_1, \ldots, s_n$; given such initialization, the functionality will consult with the adversary and will accepted a (possibly empty) sequence of $(\mathsf{Corrupt}, U)$ messages and mark the corresponding users U as corrupt. For the corrupt users the functionality will allow the adversary to set their public-keys while for honest users the functionality will sample public/secret-key pair and record it. Subsequently, any sequence of the following actions may take place: (i) A user may request to retrieve its public and secret-key, whereupon, the functionality will return it to the user. (ii) The whole directory of public-keys may be required in whereupon, the functionality will return it to the requesting user. (iii) A new user may be requested to be created by a message (Create, U) from the environment, in which case the functionality will follow the same procedure as before: it will consult the adversary regarding the corruption status of U and will set its public and possibly secret-key depending on the corruption status. The functionality will return the public-key back to the environment upon successful completion of this interaction. (v) A transaction may be requested on behalf of a certain user by the environment, by providing a template for the transaction (which should contain a unique nonce) and a recipient. The functionality will adjust the stake of each stakeholder accordingly. (iv) An existing user may be requested to be corrupted by the adversary via a message $(\mathsf{Corrupt}, U)$. A stakeholder can only be corrupted after a delay of D slots after its stake becomes 0.

Given the above we will assume that the execution of the protocol is with respect to a functionality $\mathcal{F}$ that is incorporating the above two functionalities as well as possibly additional functionalities to be explained below. Note that a corrupted stakeholder U will relinquish its entire state to $\mathcal{A}$; from this point on, the adversary will be activated in place of the stakeholder U. Beyond any restrictions imposed by $\mathcal{F}$, the adversary can only corrupt a stakeholder if it is given permission by the environment $\mathcal{Z}$ running the protocol execution. The permission is in the form of a message $(\mathsf{Corrupt}, U)$ which is provided to the adversary by the environment. In summary, regarding activations we have the following.

- At each slot sl_j, the environment $\mathcal{Z}$ is allowed to activate any subset of stakeholders it wishes. Each one of them will possibly produce messages that are to be transmitted to other stakeholders.

- The adversary is activated last in each sl_j, is allowed to read all messages sent by honest parties and may deliver them in the next slot to each stakeholder in any order it wishes, potentially including messages of its own. Adversarial messages may be delivered only to a selected set of honest stakeholders (at $\mathcal{A}$'s discretion).

- If a stakeholder is not activated in a certain slot then all the messages written to its communication tape are lost.

It is easy to see that the model above confers such sweeping power on the adversary that one cannot establish any significant guarantees on protocols of interest. It is thus important to restrict the environment suitably (taking into account the details of the protocol) so that we may be able to argue security. With foresight, the restrictions we will impose on the environment are as follows.

Restrictions imposed on the environment. The environment, which is responsible for activating the honest parties in each round, will be subject to the following constraints.

- At each slot there will be an identified set of elected shareholders, and the adversary will be permitted to corrupt only a minority of those.
- At each slot there will be a uniquely identified party, called the *slot leader*. If a stakeholder is honest and is the slot leader at a certain slot, the environment will activate it in the slot before and in the slot that it is the slot leader.
- In each round there will be at least one honest activated stakeholder (independently of whether it is a slot leader).
- There will be a parameter $k \in \mathbb{Z}$ that will signify the maximum number slots that an honest shareholder can be offline.

3 Our Protocol: Overview

We first provide a general overview of our protocol design approach. The protocol's specifics depend on a number of parameters as follows: (i) k is the number of blocks a certain transaction should have "on top of it" in order to be considered confirmed by honest stakeholders, (ii) ϵ is the advantage in terms of stake of the honest stakeholders against the adversarial ones; (iii) D is the corruption delay that is imposed on the adversary, i.e., an honest stakeholder will be corrupted after D slots when a corrupt message is delivered by the adversary during an execution; (iv) L is the lifetime of the system, measured in slots; (v) R is the length of an epoch, measured in slots.

We present our protocol description in four stages successively improving the adversarial model it can withstand. In all stages an "ideal functionality" $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ is available to the participants. The functionality captures the resources that are available to the parties as preconditions for the secure operation of the protocol (e.g., the genesis block will be specified by $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$).

1. (case for static stake; $D = L$). In the first stage, the trust assumption is static and remains with the initial set of stakeholders. There is an initial stake distribution which is hardcoded into the genesis block that includes the public-keys of the stakeholders, $\{(\mathsf{vk}_i, s_i)\}_{i=1}^n$. Based on our restrictions to the environment, honest majority with advantage ϵ is assumed among those initial stakeholders. Specifically, the environment initially will allow the corruption of a number of stakeholders whose relative stake represents $\frac{1-\epsilon}{2}$ for some $\epsilon > 0$. The environment allows party corruption by providing tokens of the form $(\mathsf{Corrupt}, U)$ to the adversary; note that due to the corruption delay imposed in this first stage any further corruptions will be against parties that have no stake initially and hence the corruption model is akin to "static corruption." $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ will subsequently sample ρ which will seed a "weighted by stake" stakeholder sampling and in this way lead to the election of a subset of m keys $\mathsf{vk}_{i_1}, \ldots, \mathsf{vk}_{i_m}$ to form the committee that will possess honest majority with overwhelming probability in m, (this uses the fact that the relative stake possessed by malicious parties is $\frac{1-\epsilon}{2}$; a dependency of m to ϵ^{-2} will be imposed at this stage). In more detail, the committee will be selected implicitly by appointing a stakeholder with probability proportional to its stake to each one of the L slots. Subsequently, stakeholders will issue blocks following the schedule that is determined by the slot assignment. The longest chain rule will be applied and it will be possible for the adversary to fork the blockchain views of the honest parties. Nevertheless, we will prove with a Markov chain argument that the probability that a fork can be maintained over a sequence of n slots drops exponentially with $\sqrt{n}$, cf. Theorem 4.11.

2. (dynamic state with a beacon, epoch period of R slots long, $D = R \ll L$). The central idea for the extension of the lifetime of the above protocol, is to consider the sequential composition of several invocations of it. We detail a way to do that, under the assumption that a trusted beacon emits a uniformly random string in regular intervals. More specifically, the beacon, during slots $\{j \cdot R + 1, \ldots, (j + 1)R\}$, reveals the j-th random string that seeds the leader election function. The critical difference compared to the static state protocol is that the stake distribution is allowed to change and is drawn from the blockchain itself. This means that at a certain slot sl that belongs to the j-th epoch (with $j \geq 2$), the stake distribution that is used is the one reported in the most recent block with time stamp at most $j \cdot R - k$.

   Regarding the evolving stake distribution, transactions will be continuously generated and transferred between stakeholders by the environment and players will incorporate posted transactions in the blockchain based ledgers that they maintain. In order to accomodate the new accounts that are being created, the $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ functionality enables a new $(\mathsf{vk}, \mathsf{sk})$ to be created on demand and assigned to a new party U_i. Specifically, the environment can create new parties who will interact with $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ for their public/secret-key in this way treating it as a trusted component that maintains the secret of their wallet. Note that the adversary can interfere with the creation of a new party, corrupt it, and supply its own (adversarially created) public-key instead. As before, the environment, may request transactions between accounts from stakeholders and it can also generate transactions in collaboration with the adversary on behalf of the corrupted accounts. Recall that our assumption is that at any slot, in the view of any honest player, the stakeholder distribution satisfies honest majority with advantage ϵ (note that different honest players might perceive a different stakeholder distribution in a certain slot). Furthermore, the stake can shift by at most σ statistical distance over a certain number of slots. The statistical distance here will be measured considering the underlying distribution to be the weighted by stake sampler and how it changes over the specified time interval. The security proof can be seen as an induction in the number of epochs L/R with the base case supplied by the proof of the static stake protocol. In the end we will argue that in this setting, a $\frac{1-\epsilon}{2} - \sigma$ bound in adversarial stake is sufficient for security of a single draw (and observe that the size of committee, m, now should be selected to overcome also an additive term of size $\ln(L/R)$). The corruption delay remains at $D = R$ which can be selected arbitrarily smaller than L, thus enabling the adversary to perform adaptive corruptions as long as this is not instantaneous.

3. (dynamic state without a beacon, epoch period of R slots long, $R = \Theta(k)$ and delay $D \in (R, 2R) \ll L$). In the third stage, we remove the dependency to the beacon, by introducing a secure multiparty protocol with "guaranteed output delivery" that simulates it. In this way, we can obtain the long-livedness of the protocol as described in the stage 2 design but only under the assumption of the stage 1 design, i.e., the mere availability of an initial random string and an initial stakeholder distribution. The core idea is the following: given we guarantee that an honest majority among elected stakeholders will hold with very high probability, we can further use this elected set as participants to an instance of a secure multiparty computation (MPC) protocol. This will require the choice of the length of the epoch to be sufficient so that it can accommodate a run of the MPC protocol. From a security point of view, the main difference with the previous case, is that the output of the beacon will become known to the adversary before it may become known to the honest parties. Nevertheless, we will prove that the honest parties will also inevitably learn it after a short number of slots. To account for the fact that the adversary gets this headstart (which it may exploit by performing adaptive corruptions) we increase the wait time for corruption from R

to a suitable value in $(R, 2R)$ that negates this advantage.

4. (stakeholder delegates and input endorsers). In the final stage of our design, we augment the protocol with two new roles for the entities that are running the protocol. First, the *delegation* feature allows stakeholders to transfer committee participation to selected delegates that assume the responsibility of the stakeholders in running the protocol (including participation to the MPC and issuance of blocks). Delegation naturally gives rise to "stake pools" that can act in the same way as mining pools in bitcoin. Finally, input-endorsers create a second layer of transaction endorsing prior to block inclusion. This mechanism enables the protocol to withstand deviations such as block-withholding and selfish mining and enables us to show that honest behaviour is an approximate Nash equilibrium.

In Section 4 we present the stage 1 of the design (the static protocol). Then in Sections 5.1 and 5.2 we present the two stages of the design for dynamic stake (using a beacon and with an MPC simulating the beacon). Finally, the enhancement with Input endorsers and incentives are discussed in Sections 5.4 and 6, while stake delegation is introduced in Section 7.

4 Our Protocol: Static State

4.1 Basic Concepts and Protocol Description

We begin by describing the blockchain protocol π_{SPoS} in the "static stake" setting, where leaders are assigned to blockchain slots with probability proportional to their (fixed) initial stake. To simplify our presentation, we abstract this leader selection process, treating it simply as an "ideal functionality" that faithfully carries out the process of randomly assigning stakeholders to slots. In the following section, we explain how to instantiate this functionality with a specific secure computation.

We remark that—even with an ideal leader assignment process—analyzing the standard "longest chain" preference rule in our PoS setting appears to require significant new ideas. The challenge arises because large collections of slots (epochs, as described above) are assigned to stakeholders at once; while this has favorable properties from an efficiency (and incentive) perspective, it furnishes the adversary a novel means of attack. Specifically, an adversary in control of a certain population of stakeholders can, at the beginning of an epoch, choose when standard "chain update" broadcast messages are delivered to honest parties with full knowledge of future assignments of slots to stakeholders. In contrast, adversaries in typical PoW settings are constrained to make such decisions in an online fashion. We remark that this can have a dramatic effect on the ability of an adversary to produce alternate chains; see the discussion on "forkable strings" below for detailed discussion.

In the static stake case, we assume that a fixed collection of n stakeholders $U_1, \ldots, U_n$ interact throughout the protocol. Stakeholder U_i possesses s_i stake before the protocol starts. For each stakeholder U_i a verification and signing key pair $(\mathsf{vk}_i, \mathsf{sk}_i)$ for a signature scheme is generated and we assume without loss of generality that each verification key vk_i is known by all stakeholders. Before describing the protocol, we establish basic definitions following the notation of [12].

Definition 4.1 (Genesis Block)**.** *The* genesis block B_0 *contains the list of stakeholders identified by their public-keys, their respective stakes* $\{(\mathsf{vk}_1, s_1), \ldots, (\mathsf{vk}_n, s_n)\}$ *and auxiliary information* ρ*.*

With foresight we note that the auxiliary information ρ will be used to seed the slot leader election process.

Definition 4.2 (Block). *A* block *B generated at a slot $sl_i \in \{sl_1, \ldots, sl_R\}$ contains the current state $st \in \{0,1\}^\lambda$, data $d \in \{0,1\}^*$, the slot number sl_i and a signature $\sigma = \mathsf{Sign}_{\mathsf{sk}_i}(st, d, sl)$ computed under sk_i corresponding to the stakeholder U_i generating the block.*

Definition 4.3 (State). *A* state *is a string $st \in \{0,1\}^\lambda$.*

Definition 4.4 (Blockchain). *A* blockchain *(or simply* chain*) relative to the genesis block B_0 is a sequence of blocks $B_1, \ldots, B_n$ associated with a strictly increasing sequence of slots for which the state st_i of B_i is equal to $H(B_{i-1})$, where H is a prescribed collision-resistant hash function. The* length *of a chain* $\text{len}(\mathcal{C})$ *is its number of blocks. The rightmost block is the* head *of the chain, denoted* $\text{head}(\mathcal{C})$*. We treat the empty string ε as a legal chain and by convention set* $\text{head}(\varepsilon) = \varepsilon$*.*

Let $\mathcal{C}$ be a chain of length n and k be any non-negative integer. We denote by $\mathcal{C}^{\lceil k}$ the chain resulting from removal of the k rightmost blocks of $\mathcal{C}$. If $k \geq \text{len}(\mathcal{C})$ we define $\mathcal{C}^{\lceil k} = \varepsilon$. We let $\mathcal{C}_1 \preceq \mathcal{C}_2$ indicate that the chain $\mathcal{C}_1$ is a prefix of the chain $\mathcal{C}_2$.

Definition 4.5 (Epoch). *An* epoch *is a set of R adjacent slots $S = \{sl_1, \ldots, sl_R\}$.*

(The value R is a parameter of the protocol we analyze in this section.)

Definition 4.6 (Adversarial Stake Ratio). *Let U_A be the set of stakeholders controlled by the adversary, the* adversarial stake ratio *is defined as*

$$\alpha = \frac{\sum_{j \in U_A} s_j}{\sum_{i=1}^{n} s_i}$$

where n is the total number of stakeholders and s_i is stakeholder U_i's stake.

Slot Leader Selection. In the protocol described in this section, for each $0 < j \leq R$, a *slot leader* E_j is determined who has the (sole) right to generate a block at sl_j. Specifically, for each slot a stakeholder U_i is selected as the slot leader with probability p_i proportional to its stake registered in the genesis block B_0; these assignments are independent between slots. In this static stake case, the genesis block as well as the procedure for selecting slot leaders are determined by an ideal functionality $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$, defined in Figure 1. This functionality is parameterized by the list $\{(\mathsf{vk}_1, s_1), \ldots, (\mathsf{vk}_n, s_n)\}$ assigning to each stakeholder its respective stake, a distribution $\mathcal{D}$ that provides auxiliary information ρ and a *leader selection function* F defined below.

Definition 4.7 (Leader Selection Process). *A* leader selection process $(\mathcal{D}, \mathsf{F})$ *is a pair consisting of a distribution and a deterministic function such that, when $\rho \leftarrow \mathcal{D}$ it holds that for all $sl_j \in \{sl_1, \ldots, sl_R\}$, $\mathsf{F}(\rho, sl_j)$ outputs $U_i \in \{U_1, \ldots, U_n\}$ with probability*

$$p_i = \frac{s_i}{\sum_{k=1}^{n} s_k}$$

where s_i is the stake held by stakeholder U_i (we call this "weighing by stake"); furthermore the family of random variables $U_i = F(\rho, sl_j)$ are independent.

We note that weighing by stake sampling can be implemented in a straightforward manner. For instance, a simple process operates as follows. Let $\tilde{p}_i = s_i / \sum_{j=i}^{n} s_i$. First, the process flips a $\tilde{p}_1$-biased coin to see whether the first stakeholder is selected; subsequently, for all $j \geq 2$, it flips a $(1-\tilde{p}_1)\ldots(1-\tilde{p}_{j-1})\tilde{p}_j$ biased coin to see whether the j-th stakeholder is selected. When we implement it as a function $F(\cdot)$ a sufficient amount of randomness should be allocated to simulate the biased coin flips. If we implement the above with λ precision for each individual coin flip, then selecting a stakeholder will require $n\lceil \log \lambda \rceil$ random bits in total. Note that using a pseudorandom number generator (PRG) one may use a shorter "seed" string and then stretch it using the PRG to the appropriate length.

Functionality $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$

$\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ is parameterized by the public keys and respective stakes of the stakeholders $\{(\mathsf{vk}_1, s_1), \ldots, (\mathsf{vk}_n, s_n)\}$, a distribution $\mathcal{D}$ and a function F so that $(\mathcal{D}, \mathsf{F})$ is a leader selection process. $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ interacts with stakeholders $U_1, \ldots, U_n$ as follows:

- Upon receiving $(\mathsf{genblock\_req}, U_i)$ from stakeholder U_i, $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ proceeds as follows. If $B_0 = \emptyset$, $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ samples $\rho \leftarrow D$ and sets $B_0 = ((\mathsf{vk}_1, s_1), \ldots, (\mathsf{vk}_n, s_n), \rho)$. Finally, $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ sends $(\mathsf{genblock}, B_0, \mathsf{F})$ to U_i.

Figure 1: Functionality $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$.

A Protocol in the $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$-hybrid model. We start by describing a simple PoS based blockchain protocol considering static stake in the $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$-hybrid model, i.e., where the genesis block B_0 (and consequently the slot leaders) are determined by the ideal functionality $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$. The stakeholders $U_1, \ldots, U_n$ interact among themselves and with $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ through Protocol π_{SPoS} described in Figure 2.

The protocol relies on a $\mathsf{maxvalid}_S(\mathcal{C}, \mathbb{C})$ function that chooses a chain given the current chain $\mathcal{C}$ and a set of valid chains $\mathbb{C}$ that are available in the network. In the static case we analyze the simple "longest chain" rule. (In the dynamic case the rule is parameterized by a common chain length; see Section 5.)

Function $\mathsf{maxvalid}_S(\mathcal{C}, \mathbb{C})$: Returns the longest chain from $\mathbb{C} \cup \{\mathcal{C}\}$. Ties are broken in favor of $\mathcal{C}$, if it has maximum length, or arbitrarily otherwise.

Protocol π_{SPoS}

π_{SPoS} is a protocol run by stakeholders $U_1, \ldots, U_n$ interacting among themselves and with $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ over a sequence of slots $S = \{sl_1, \ldots, sl_R\}$. π_{SPoS} proceeds as follows:

1. **Initialization** When π_{SPoS} starts, each stakeholder $U_i \in \{U_1, \ldots, U_n\}$ sends $(\mathsf{genblock\_req}, U_i)$ to $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$, receiving $(\mathsf{genblock}, B_0, \mathsf{F})$ as answer. U_i sets an internal blockchain $\mathcal{C} = B_0$ and a initial internal state $st = H(B_0)$.
2. **Chain Extension** For every slot $sl_j \in S$, every online stakeholder U_i performs the following steps:
   (a) Collect all valid chains received via broadcast into a set $\mathbb{C}$, verifying that for every chain $\mathcal{C}' \in \mathbb{C}$ and every block $B' = (st', d', sl', \sigma') \in \mathcal{C}'$ it holds that $\mathsf{Vrf}_{\mathsf{vk}'}(\sigma', (st', d', sl')) = 1$, where vk' is the verification key of the stakeholder $U' \leftarrow \mathsf{F}(\rho, sl')$. U_i calls the function $\mathsf{maxvalid}_S(\mathcal{C}, \mathbb{C})$ to select a new internal chain $\mathcal{C} \in \mathbb{C}$ and sets state $st = H(\mathrm{head}(\mathcal{C}))$.
   (b) If U_i is the slot leader determined by $\mathsf{F}(\rho, sl_j)$, it generates a new block $B = (st, d, sl_j, \sigma)$ where st is its current state, $d \in \{0,1\}^*$ is the transaction data and $\sigma = \mathsf{Sign}_{\mathsf{sk}_i}(st, d, sl_j)$ is a signature on (st, d, sl_j). U_i extends $\mathcal{C}$ by appending B, obtains $\mathcal{C} = \mathcal{C}|B$ and broadcasts the new $\mathcal{C}$.

Figure 2: Protocol π_{SPoS}.

4.2 Forkable Strings

In our security arguments, we will treat strings over $\{0,1\}^*$ as an abstraction to indicate which slots—among a particular window of slots—have been assigned to adversarial stakeholders.

Definition 4.8 (Characteristic String). *Fix an execution with genesis block B_0, adversary $\mathcal{A}$, and environment $\mathcal{Z}$. Let $S = \{sl_i, \ldots, sl_j\}$, where $i < j$, be a sequence of slots of length $|S| = n$. The* characteristic string *$w \in \{0,1\}^n$ of S is defined so that $w_k = 1$ if and only if the adversary controls the slot leader of slot sl_k.*

For such a characteristic string $w \in \{0,1\}^*$ we say that the slot i is *adversarial* if $w_i = 1$ and *honest* otherwise.

We start with some intuition on our approach to analyze the protocol. Let $w \in \{0,1\}^n$ be a characteristic string for a sequence of slots S. Consider two observers that (i.) go offline immediately prior to the commencement of S, (ii.) have the same view $\mathcal{C}_0$ of the current chain prior to the commencement of S, and (iii.) come back online at the last slot of S and request an update of their chain. A fundamental concern in our analysis is the possibility that such observers can be presented with a "diverging" view over the sequence S: specifically, the possibility that the adversary can force the two observers to adopt two different chains $\mathcal{C}_1, \mathcal{C}_2$ whose common prefix is $\mathcal{C}_0$.

We observe that not all characteristic strings permit this. For instance the (entirely honest) string 0^n ensures that the two observers will adopt the same chain $\mathcal{C}$ which will consist of n new blocks on top of the common prefix $\mathcal{C}_0$. On the other hand, other strings do not guarantee such common extension of $\mathcal{C}_0$; in the case of 1^n, it is possible for the adversary to produce two completely different histories during the sequence of slots S and thus furnish to the two observers two distinct chains $\mathcal{C}_1, \mathcal{C}_2$ that only share the common prefix $\mathcal{C}_0$. In the remainder of this section, we establish that strings that permit such "forkings" are quite rare—indeed, we show that they have density $2^{-\Omega(\sqrt{n})}$ so long as the fraction of adversarial slots is $1/2 - \epsilon$.

To reason about such "forkings" of a characteristic string $w \in \{0,1\}^n$, we define below a formal notion of "fork" which captures the relationship between the chains broadcast by *honest* slot leaders during an execution of the protocol π_{SPoS}. In preparation for the definition, we recall that honest players always choose to extend a maximum length chain among those available to the player on the network. Furthermore, if such a maximal chain $\mathcal{C}$ includes a block previously broadcast by an honest player, the prefix of $\mathcal{C}$ prior to this honest block must entirely agree with the chain broadcast by this previous honest player. Thus any chain $\mathcal{C}$ broadcast by an honest player must consist of a (perhaps empty) sequence of adversarial blocks that extend a chain produced by a previous honest player. It follows that the chains broadcast by honest players form a natural directed tree. The fact that honest players reliably broadcast their chains and always build on the longest available chain introduces a second important property of this tree: the "depths" of distinct blocks added by honest players during the protocol must all be distinct. The definition of "fork" below formalizes these requirements and will be the basis for our arguments establishing that "forkable" strings are scarce.

Forks and forkable strings. We define, below, the basic combinatorial structures we use to reason about the possible views observed by honest players during a protocol execution with this characteristic string.

Definition 4.9 (Fork). *Let $w \in \{0,1\}^n$ and let $H = \{i \mid w_i = 0\}$ denote the set of honest indices. A* fork *for the string w is a directed, rooted tree $F = (V, E)$ with a labeling $\ell : V \to \{0, 1, \ldots, n\}$ so that*

- *each edge of F is directed away from the root;*
- *the root $r \in V$ is given the label $\ell(r) = 0$;*
- *the labels along any directed path in the tree are strictly increasing;*

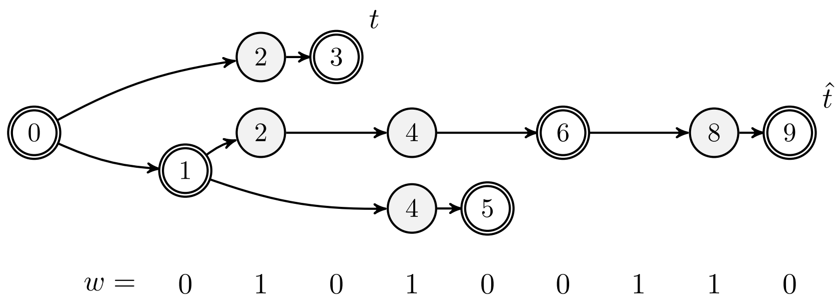


Figure 3: A fork F for the string $w = 010100110$; vertices appear with their labels and honest vertices are highlighted with double borders. Note that the depths of the (honest) vertices associated with the honest indices of w are strictly increasing. Two tines are distinguished in the figure: one, labeled $\hat{t}$, terminates at the vertex labeled 9 and is the longest tine in the fork; a second tine t terminates at the vertex labeled 3. The quantity $\mathrm{gap}(t)$ indicates the difference in length between t and $\hat{t}$; in this case $\mathrm{gap}(t) = 4$. The quantity $\mathrm{reserve}(t) = |\{i \mid \ell(v) < i \leq |w| \text{ and } w_i = 1\}|$ indicates the number of adversarial indices appearing after the label of the last vertex v of the tine; in this case $\mathrm{reserve}(t) = 3$. As each leaf of F is honest, F is closed.

- *each honest index $i \in H$ is the label of exactly one vertex of F;*
- *the function $\mathbf{d} : H \to \{1, \ldots, n\}$, defined so that $\mathbf{d}(i)$ is the depth in F of the unique vertex v for which $\ell(v) = i$, is strictly increasing. (Specifically, if $i, j \in H$ and $i < j$, then $\mathbf{d}(i) < \mathbf{d}(j)$.)*

As a matter of notation, we write $F \vdash w$ to indicate that F is a fork for the string w.

If a vertex v of a fork is labeled with an adversarial index (i.e., $w_{\ell(v)} = 1$) we say that the vertex is adversarial; otherwise, we say that the vertex is honest. In particular, for concreteness we declare the root vertex to be honest. We say that a fork is *trivial* if it contains a single vertex, the root.

Note that the label $\ell(v)$ of any non-root vertex v is an index of the string w and that adversarial indices may, in general, appear as the label of many (or no) vertices of the tree.

A path in a fork originating at the root is called a *tine*. For a tine t we let $\mathrm{length}(t)$ denote its *length*, equal to the number of edges on the path. The *height* of a fork (as usual for a tree) is defined to be the length of the longest tine. We write $t_1 \preceq t_2$ if t_1 is a prefix of the tine t_2. As described above, a fork effectively determines the structure of the chains (tines) broadcast by the honest players during a run of the protocol. See Figure 3 for an example, which also demonstrates some of the quantities defined above and in the next section.

Definition 4.10. *We say that a fork is* flat *if it has (at least) two edge-disjoint tines of length equal to the height of the fork. A string $w \in \{0,1\}^*$ is said to be* forkable *if there is a flat fork $F \vdash w$.*

Our goal is to establish the following upper bound on the number of forkable strings.

Theorem 4.11. *Let $\epsilon \in (0, 1)$ and let w be a string drawn from $\{0,1\}^n$ by independently assigning each $w_i = 0$ with probability $(1+\epsilon)/2$. Then* $\Pr[w \textit{ is forkable}] = 2^{-\Omega(\sqrt{n})}$.

Structural features of forks: closed forks, prefixes, and margin. We begin by defining a natural notion of inclusion for two forks:

Definition 4.12 (Fork prefixes)**.** *If w' is a prefix of the string $w \in \{0,1\}^*$, $F' \vdash w'$, and $F \vdash w$, we say that F' is a* prefix *of F, written $F' \sqsubseteq F$, if F' is a consistently-labeled subgraph of F.*

Specifically, every vertex and edge of F' appears in F and, furthermore, the labels given to any vertex appearing in both F' and F are identical.

If $F' \sqsubseteq F$, each tine of F' appears as the prefix of a tine in F. In particular, the labels appearing on any tine terminating at a common vertex are identical and, moreover, the depth of any honest vertex appearing in both F' and F is identical.

In many cases, it is convenient to work with forks that do not "commit" anything beyond final honest indices.

Definition 4.13 (Closed forks)**.** *A fork is* closed *if each leaf is honest. For concreteness, we define the trivial fork, consisting solely of a root vertex, as closed.*

Note that a closed fork has a unique longest tine (as all maximal tines terminate with an honest vertex, and these must have distinct depths). Note, additionally, that if w' is a prefix of w and $F \vdash w$, then there is a unique closed fork $F' \vdash w'$ for which $F' \sqsubseteq F$.

Definition 4.14 (Gap, reserve and margin)**.** *Let $F \vdash w$ be a closed fork and let $\hat{t}$ denote the (unique) tine of maximum length in F. We define the* gap *of a tine t, denoted* gap(t)*, to be the difference in length between $\hat{t}$ and t; thus*

$$\mathrm{gap}(t) = \mathrm{length}(\hat{t}) - \mathrm{length}(t)\,.$$

We define the reserve *of a tine t to be the number of adversarial indices appearing in w after the largest honest index in t; specifically, if t is given by the path $(r, v_1, \ldots, v_k)$, where r is the root of F, we define*

$$\mathrm{reserve}(t) = |\{i \mid w_i = 1 \text{ and } i > \ell(v_k)\}|\,.$$

We remark that this quantity depends both on F and the specific string w associated with F. For two tines t and t', we write $t \sim t'$ if they share an edge in F; otherwise we write $t \not\sim t'$. Finally, we define the margin *of a fork F by the rule*

$$\mathrm{margin}(F) = \max_{t \not\sim \hat{t}} \left(\mathrm{reserve}(t) - \mathrm{gap}(t)\right)\,.$$

The relevance of margin to the notion of forkability is reflected in the following proposition.

Proposition 4.15. *A string w is forkable if and only if there is a closed fork $F \vdash w$ with* margin$(F) \geq 0$.

Proof. If w has no honest indices, then the trivial fork consisting of a single root node is flat, closed, and has non-negative margin; thus the two conditions are equivalent. Consider a forkable string w with at least one honest index and let $\hat{\imath}$ denote the largest honest index of w. Let F be a flat fork for w. As mentioned above, there is a unique closed fork $\overline{F} \vdash w$ obtained from F by removing any adversarial vertices from the ends of the tines of F. Note that the tine $\hat{t}$ containing $\hat{\imath}$ is the longest tine in $\overline{F}$, as this is the largest honest index of w. On the other hand, F is flat, in which case there are two edge-disjoint tines t_1 and t_2 with length at least that of $\hat{t}$. At least one of these—call it t—must be edge-disjoint with $\hat{t}$ and it follows that the prefix of t in $\overline{F}$ has reserve no less than its gap, as desired.

On the other hand, suppose w has a closed fork with margin$(F) \geq 0$, and let $\hat{t}$ denote the longest tine. As margin$(F) \geq 0$, there is a particular tine t, edge-disjoint with $\hat{t}$, for which reserve(t) − gap$(t) \geq 0$ and we can produce a flat fork by simply adding gap(t) vertices labeled with these subsequent adversarial indices to the tine t. □

In light of this proposition, for a string w we may focus our attention on the quantity

$$\operatorname{margin}(w) = \max_{\substack{F \vdash w,\\ F \text{ closed}}} \operatorname{margin}(F)\,.$$

(Note that this overloads the notation margin($\cdot$) so that it applies to both forks and strings, but the setting will be clear from context.)

The full proof of Theorem 4.11 appears in Appendix A. We devote the remainder of this section to an in-depth account of the proof of Theorem 4.11 in the "two-tine" case, when forks may have no more than two tines. This shows off the basic features of the proof without the more elaborate bookkeeping required for the general case.

Intuition for the proof of Theorem 4.11: The two-tine case. The two-tine proof relies on the fact that margin can be given a purely syntactic description when margin is maximized over only two-tine forks. We say that a fork is a 2-fork if it can be expressed as the union of two tines and, for the remainder of this section we work with the quantity

$$\operatorname{margin}_2(w) = \max_{\substack{F \vdash w,\\ F \text{ closed 2-fork}}} \operatorname{margin}(F)\,.$$

(Note that $\operatorname{margin}_2(w) \leq \operatorname{margin}(w)$.)

We provide a syntactic description of $\operatorname{margin}_2()$ over the set

$$T_0 = \{w0 \mid w \in \{0,1\}^*\} \cup \{\epsilon\}$$

of strings ending with an honest index (to which we add the string ϵ for convenience). Specifically, define the function $\mathbf{rm} : T_0 \to \mathbb{Z}$ by the recursive rule

$$\begin{aligned} \mathbf{rm}(\epsilon) &= 0\,,\\ \mathbf{rm}(w'1^s0) &= \begin{cases} s-1, & \text{if } -s \leq \mathbf{rm}(w') \leq 0,\\ \mathbf{rm}(w') + s - 1, & \text{otherwise.} \end{cases} \end{aligned}$$

(Note that any nonempty string $w \in T_0$ can be written uniquely as $w'1^s0$ for a string $w' \in T_0$.) We will later extend this definition to all strings.

Theorem 4.16. *For every $w \in T_0$, $\mathbf{rm}(w) = \operatorname{margin}_2(w)$.*

Proof. In preparation for the proof, we make some general comments about forks of "neighboring" strings in T_0. Consider a closed 2-fork F' for a string $w' \in T_0$; appending the string 1^s0 to the end of w' results in another string $w = w'1^s0 \in T_0$ with exactly one more honest index. We consider the various closed 2-forks $F \vdash w$ which are consistent with F' in the sense that $F' \sqsubseteq F$. As both F and F' are closed and w contains single honest index not appearing in w', these two forks differ only along the longest tine of F which, in fact, terminates with a vertex labeled by the new honest index at the end of w. In particular, F may be constructed from F' by adding to the end of a tine t (of F') a directed path labeled with some of the symbols appearing in the suffix 0^s1. Note that a time t of F' may only be augmented in this way if $s \geq \operatorname{gap}(t) - \operatorname{reserve}(t)$ (as the augmented tine must necessarily become the longest tine in F). The corresponding effect of such augmentation on margin() depends on whether or not the augmented tine is $\hat{t}$, the longest tine of F'. We consider these two cases separately: when $\hat{t}$ is augmented, we call this an *extension*; in the other case, where the augmented tine t shares no edges with $\hat{t}$, we call this a *crossover*.

Extension. It is always possible to add the new honest index to the longest tine $\hat{t}$ of F' (as $\text{gap}(\hat{t}) = 0$). This corresponds to augmenting $\hat{t}$ by adding a directed path of vertices labeled with indices that correspond to a string of the form 1^r0 for some $r \leq s$ (so that these symbols can be drawn from the appended string 1^s0). Note that this changes the gap by $r+1$ while s is added to the reserve. Thus $\text{margin}(F) = \text{margin}(F') + s - (r+1)$.

Crossover. If $s + \text{margin}(F') \geq 0$, the new honest index may be added to the shorter tine (which shares no nontrivial prefix with $\hat{t}$), creating a "crossover." The shorter tine t may be augmented by adding a directed path labeled with the string $1^{\text{gap}(t)}1^r0$ for some $0 \leq r \leq s + \text{reserve}(t) - \text{gap}(t)$. Note in this case that the new gap is $1 + r$ and the reserve is s. Thus $\text{margin}(F) = s - (r+1)$.

Returning to the statement of the theorem, we first prove that for any $w \in T_0$, $\mathbf{rm}(w) \geq \text{margin}_2(w)$. The proof proceeds by induction on the number of honest indices in w. If w has no honest indices, then $w = \epsilon$ and $\mathbf{rm}(w) = 0 = \text{margin}_2(w)$, as desired. If w has at least one honest index, we may write $w = w'1^s0$ for some $w' \in T_0$ and, by induction, we may assume that $\mathbf{rm}(w') \geq \text{margin}_2(w')$. Let $F \vdash w$ be closed and let $F' \vdash w'$ be the unique closed fork for which $F' \sqsubseteq F$. Then

$$\text{margin}(F') \leq \text{margin}_2(w') \leq \mathbf{rm}(w') .$$

As described above, F is either formed by extending F' or by crossover, and we handle these two cases separately. If F is formed by extending F', then

$$\text{margin}(F) = \text{margin}(F') + s - (r+1) \leq \mathbf{rm}(w') + (s-1) ,$$

where r is determined as in the description of the "extend" case above. Note, however, that $\mathbf{rm}(w) \geq \mathbf{rm}(w') + (s-1)$ in this case: either $-s \leq \mathbf{rm}(w') \leq 0$ so that $\mathbf{rm}(w') + s - 1 \leq s - 1 = \mathbf{rm}(w)$ or $\mathbf{rm}(w')$ is outside this range so that $\mathbf{rm}(w) = \mathbf{rm}(w') + s - 1$. Thus

$$\text{margin}(F) \leq \mathbf{rm}(w') + (s-1) \leq \mathbf{rm}(w) ,$$

as desired. Alternatively, suppose that F is formed from F' by crossover, which requires that

$$0 \leq \text{margin}(F') + s \leq \text{margin}_2(w') + s .$$

Then $\text{margin}(F) = s - (r+1) \leq s - 1$ where r is determined as in the description of the "crossover" case above. Note, however, that $\mathbf{rm}(w) \geq s-1$ in this case: either $\mathbf{rm}(w') \leq 0$ so that $\mathbf{rm}(w) = s-1$ or $\mathbf{rm}(w') > 0$ so that $\mathbf{rm}(w) = \mathbf{rm}(w') + (s-1) \geq s-1$. In any case,

$$\text{margin}(F) = s - (r+1) \leq s - 1 \leq \mathbf{rm}(w) ,$$

as desired.

It remains to prove that for any $w \in T_0$, $\mathbf{rm}(w) \leq \text{margin}_2(w)$. For this purpose, for each string $w \in T_0$ we construct a particular "canonical" 2-fork F_w for which $\text{margin}(F_w) = \mathbf{rm}(w)$. We define F_ϵ to be the trivial fork, and note that $\mathbf{rm}(\epsilon) = \text{margin}(F_\epsilon)$. In general, the fork F_w is defined by induction: writing $w = w'1^s0$ the fork F_w is determined from $F_{w'}$ depending on the quantities $\mathbf{rm}(w')$ and s. If $-s \leq \mathbf{rm}(w') \leq 0$, F_w is obtained from $F_{w'}$ by a crossover in which $r = 0$—specifically, the shorter tine t is extended by $\text{gap}(t) - \text{reserve}(t)$ adversarial vertices and a vertex corresponding to the final honest index of w. Otherwise, the longer tine $\hat{t}$ of $F_{m'}$ is extended by adding a single vertex corresponding to the new honest index in w. In either case, we find that $\text{margin}(F_w) = \mathbf{rm}(w)$ by construction. See Figure 4 for an example of the canonical forking for a particular string. $\square$

We remark that the "canonical two-tine" fork F_w defined in the proof above has minimal depth, in the sense that the longest tine always has length equal to the number of honest indices in w.

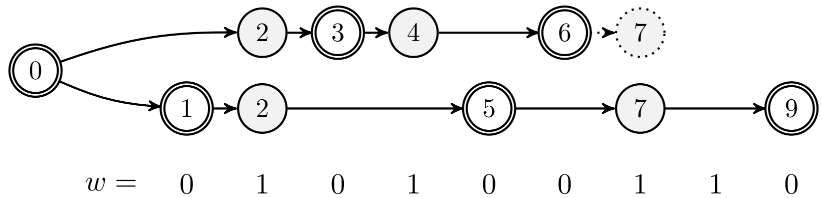


Figure 4: The "canonical two-tine" fork F_w for the string $w = 010100110$. Nodes use the same conventions as in Figure 3. The upper tine of the canonical two-tine fork terminates at the vertex labeled 6; the addition of the adversarial vertex labeled 7 to the upper tine results in a flat fork for this string.

Bounding the density of forkable strings; the Markov chain for $\mathrm{margin}_2()$**.** We extend the definition of $\mathbf{rm}$ to the set of all strings $\{0,1\}^*$. For a string $w \in \{0,1\}^*$, let $\mathbf{tail}(w)$ denote the number of trailing 1's in the string (that is, $\max\{k \mid w = w'1^k \text{ for some } w'\}$); then (re-)define $\mathbf{rm}(w)$ by the rule

$$\begin{aligned}
\mathbf{rm}(\epsilon) &= 0\,,\\
\mathbf{rm}(w0) &= \mathbf{rm}(w) - 1\,,\\
\mathbf{rm}(w1) &= \begin{cases} \mathbf{rm}(w)+1 & \text{if } \mathbf{rm}(w)+1 \neq 0,\\ \mathbf{tail}(w1) & \text{if } \mathbf{rm}(w)+1 = 0. \qquad (\dagger)\end{cases}
\end{aligned}$$

It is easy to establish by induction that these two definitions of $\mathbf{rm}$ agree on T_0. (The "crossover case" in the original definition of $\mathbf{rm}$, when $-s \leq \mathbf{rm}(w') \leq 0$, is correctly accounted for by the case (†) above.)

In general, $\mathbf{rm}(w)$ is *not* equal to $\mathrm{margin}_2(w)$ on strings $w \notin T_0$. The rule (†) above can inflate $\mathbf{rm}()$ in preparation for correctly accounting for margin when a following honest index is added. However, for any w we have $\mathrm{margin}_2(w) \leq \mathbf{rm}(w)$, which is enough for our density bounds.

Lemma 4.17. *For all* $w \in \{0,1\}^*$, $\mathrm{margin}_2(w) \leq \mathbf{rm}(w)$.

Proof. For a string $w \in \{0,1\}^*$, we may (uniquely) express $w = w'1^s$ for a string $w' \in T_0$. If $s = 0$ then $w \in T_0$, and $\mathbf{rm}(w) = \mathrm{margin}_2(w)$ by Theorem 4.16. Note that appending an adversarial index to the end of a string w' increases the reserve of every tine in a closed fork $F \vdash w'$ by exactly one. Thus $\mathrm{margin}_2(w'1^s) = \mathrm{margin}_2(w') + s$. On the other hand, from the recursive description of $\mathbf{rm}$ above it is easy to check that $\mathbf{rm}(w'1^s) \geq \mathbf{rm}(w') + s$. (Note that $\mathbf{tail}(w'1) \geq 1 \geq \mathbf{rm}(w') + 1 = 0$ in the case when rule (†) applies.) Thus $\mathbf{rm}(w) \geq \mathrm{margin}_2(w))$ for all $w \in \{0,1\}^*$. □

Recall that a string is forkable if and only if $\mathrm{margin}(w) \geq 0$. To continue our discussion of 2-tine forks, we say that a string is 2-forkable if $\mathrm{margin}_2(w) \geq 0$. We wish to prove a two-tine analogue of Theorem 4.11:

$$\Pr[w \text{ is 2-forkable}] = \Pr[\mathrm{margin}_2(w) \geq 0] = 2^{-\Omega(\sqrt{n})}\,,$$

where $w \in \{0,1\}^n$ is chosen by independently selecting each $w_i \in \{0,1\}$ so that

$$\Pr[w_i = 0] = \frac{1+\epsilon}{2} = 1 - \Pr[w_i = 1]\,.$$

Specifically, consider the random variables $X_t = \mathbf{rm}(w_1 \dots w_t)$. Our goal is to analyze

$$\Pr[\mathrm{margin}_2(w) \geq 0] \leq \Pr[X_n \geq 0]$$

and the proof focuses on the last of these quantities. Note that if it were not for the exotic behavior around zero (that is, the case that $\mathbf{rm}(w)+1=0$), the random variables X_t would simply describe a biased random walk. In particular, they would arise from the familiar Markov chain of Figure 5. where $p=(1+\epsilon)/2$ and $q=1-p$.

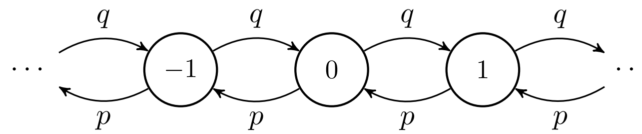
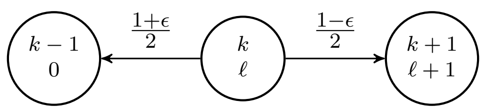
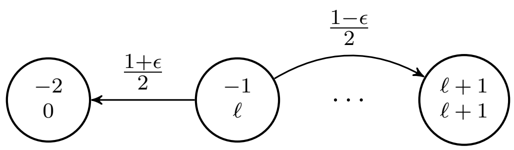


Figure 5: The simple biased walk.

With the exotic transition $\mathbf{rm}(w1)=\mathbf{tail}(w1)$ (when $\mathbf{rm}(w)=-1$), we note that this process is no longer strictly Markovian, as this transition depends on the number of "recent" 1 symbols in the sequence. We can reflect this with a richer Markov chain over the state space $\mathbb{Z}\times\mathbb{Z}$, which simultaneously maintains $\mathbf{rm}(m)$ and $\mathbf{tail}(m)$. This permits the chain to correctly handle the exotic rule associated with $\mathbf{rm}()=-1$; the chain is described in Figure 6.



(a) The dynamics at nodes with $k\neq -1$.



(b) The dynamics at $k=-1$.

Figure 6: Diagram of the lifted Markov chain. The first coordinate, k, maintains the current $\mathbf{rm}()$; the second coordinate, ℓ, maintains the current $\mathbf{tail}()$. Transitions with probability $(1+\epsilon)/2$ correspond to addition of a 0 to the string; those with probability $(1-\epsilon)/2$ correspond to addition of a 1.

The basic event we wish to analyze is the event that after n steps on this Markov chain, the resulting margin is negative, that is

$$\Pr[X_n\geq 0]$$

for the random variables X_t defined above. Write $w=w^{(1)}\cdots w^{(\sqrt{n})}$ where $\lfloor\sqrt{n}\rfloor\leq|w^{(i)}|\leq\lceil\sqrt{n}\rceil$ for each i. Fix $\delta<\epsilon$ to be a small constant. Let $L^{(i)}$ denote the event that there is a contiguous sequence of "1" symbols of length exceeding $\delta\sqrt{n}$ in the string $w^{(i)}$. Then $\Pr[L^{(i)}]\leq\sqrt{n}2^{-\delta\sqrt{n}}=2^{-\Omega(\sqrt{n})}$. We remark that these events are independent for distinct values of i, as they involve non-overlapping sets of symbols of w.

Let $m_t=\mathbf{rm}(w^{(1)}\cdots w^{(t-1)})$. We define three events based on this margin:

Hot We let Hot_t denote the event that $m_t\geq 2\delta\sqrt{n}$ or $L^{(t-1)}$ occurred.

Volatile We let Vol_t denote the event that $-2\delta\sqrt{n}\leq m_t<2\delta\sqrt{n}$ and $L^{(t-1)}$ did not occur.

Cold We let Cold_t denote the event that $m_t<-2\delta\sqrt{n}$.

(We assume, by convention, that $L^{(-1)}$ does not occur.) Note that for each t, exactly one of these events occurs—they partition the probability space. Then we will establish that

$$\Pr[\mathsf{Cold}_{t+1} \mid \mathsf{Cold}_t] \geq 1 - 2^{-\Omega(\sqrt{n})}\,, \tag{1}$$

$$\Pr[\mathsf{Cold}_{t+1} \mid \mathsf{Vol}_t] \geq \Omega(\epsilon)\,, \tag{2}$$

$$\Pr[\mathsf{Hot}_{t+1} \mid \mathsf{Vol}_t] \leq 2^{-\Omega(\sqrt{n})}\,. \tag{3}$$

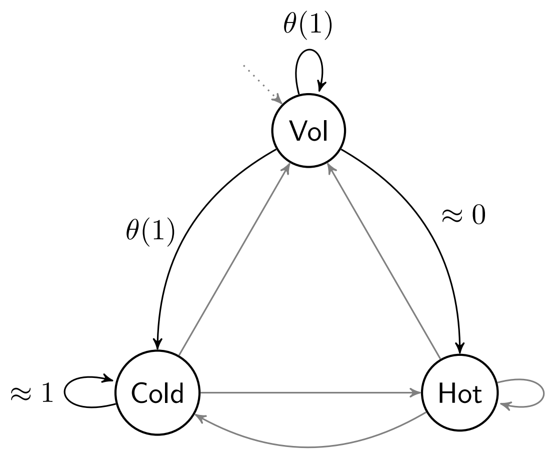


Figure 7: An illustration of the transitions between Cold, Vol, and Hot.

Note that the event Vol_1 occurs by definition. We wish to show that the system is very likely to eventually become cold, and stay that way. Note that the probability that the system ever transitions from volatile to hot is no more than $2^{-\Omega(\sqrt{n})}$ (as transition from Vol to Hot is bounded above by $2^{-\Omega(\sqrt{n})}$, and there are no more than $\sqrt{n}$ possible transition opportunities). Note, also, that while the system is volatile, it transitions to cold with constant probability during each period. In particular, the probability that the system is volatile for the entire process is no more that $2^{-\Omega(\sqrt{n})}$. Finally, note that the probability that the system ever transitions out of the cold state is no more than $2^{-\Omega(\sqrt{n})}$ (again, there are at most $\sqrt{n}$ possible times when this could happen, and any individual transition occurs with probability $2^{-\Omega(\sqrt{n})}$). It follows that the system is cold at the end of the process with probability $1 - 2^{-\Omega(\sqrt{n})}$.

In preparation for establishing the three inequalities (6), (7), and (8), we note two facts about the simple biased walk (of Figure 5) with $p = (1+\epsilon)/2$ and $\epsilon > 0$.

Constant escape probability. As $\epsilon > 0$, the probability that an infinite walk beginning at state 0 ever visits the state 0 again is a constant less than 1 (depending only on ϵ). (See, e.g., [13, Chapter 12].)

Concentration. Consider s steps of the Markov chain beginning at state 0; then the resulting value is tightly concentrated around $-\epsilon s$. Specifically, let $Z_1, \ldots, Z_S$ be i.i.d. $\{\pm 1\}$-valued random variables with $\Pr[Z_s = 1] = (1-\epsilon)/2$, in which case the expected value $\mathbb{E}[\sum_{s=1}^{S} Z_s] = -\epsilon S$. Then

$$\Pr\left[\sum_s Z_s > -\frac{\epsilon S}{2}\right] = 2^{-\Omega(S)}\,. \tag{4}$$

(The constant hidden in the $\Omega()$ notation depends only on ϵ. See, e.g., [1, Cor. A.1.14].)

Inequality (6) follows directly from the concentration statement above: Note that unless the Markov chain visits a state for which $k = -1$, it behaves like the simple unbiased walk. Let $D^{(i)}$ denote the event that there is a contiguous sequence of symbols x in $w^{(i)}$ for which $\#1(x) - \#0(x) \geq \delta\sqrt{n}$. By the same Chernoff bound of (4) above, $\Pr[D^{(i)}] \leq 2^{-\Omega(\sqrt{n})}$. (Observe that such an event can only happen if the sequence of symbols has length at least $\delta\sqrt{n}$, in which case the Chernoff bound can be applied.) As the chain starts with $m_t < 2\delta\sqrt{n}$, unless D_t occurs, the Markov chain cannot possibly visit the state -1. It follows, again from (4), that the probability that $m_{t+1} \geq 2\delta\sqrt{n}$ is $2^{-\Omega(\sqrt{n})}$. (In fact, the Chernoff bound shows that with high probability, the value of m_{t+1} has significantly decreased.) Finally, the probability of L_t is $2^{-\Omega(\sqrt{n})}$. Thus the probability of $\overline{\mathsf{Cold}_{t+1}}$ is $2^{-\Omega(\sqrt{n})}$, as desired.

As for inequality (7), note that if the system starts with $m_t \leq -1$ then with constant probability it will never visit $k = -1$ during (the rest of) $m^{(t)}$ and, conditioned on that, will end with a margin $< -2\delta\sqrt{n}$ with probability $1 - 2^{-\Omega(\sqrt{n})}$ (by (4)). If, on the other hand, the system starts with $m_t > -1$ (but less than $2\delta\sqrt{n}$), again by a Chernoff bound it will visit the node $(k = -1, \ell = 0)$ with probability $1 - 2^{-\Omega(\sqrt{n})}$ during the first half of the string $w^{(t)}$. As in the other case, the probability that it never returns to margin -1 and ends up below $-2\delta\sqrt{n}$ is a constant. The result follows.

Finally, consider inequality (8). Note that, first of all, by the union bound, $L^{(t)}$ occurs with probability no more than $\sqrt{n}2^{-\Omega(\sqrt{n})} = 2^{-\Omega(\sqrt{n})}$. The other way for the event Hot_{t+1} to occur is that the margin, initially smaller than $2\delta\sqrt{n}$, "escapes" to a value exceeding this. We separate this analysis into two cases: if the initial margin is positive (or zero), note with probability at least $1 - 2^{-\Omega(\sqrt{n})}$, the margin will return to 0 during $w^{(t)}$ by (4). After this, note that assuming that neither $L^{(t)}$ or $D^{(t)}$ occur, the maximum possible margin that can appear in the remainder of $w^{(t)}$ is $2\delta\sqrt{n}$, as desired. (The factor of 2 arises due to the possibility that a transition through zero induces a **tail**() of size $\delta\sqrt{n}$.) On the other hand, if the initial margin is negative, as $L^{(t-1)}$ did not occur, the same argument concludes that the maximum final margin is no more than $2\delta\sqrt{n}$.

Experiments In order to gain further insight regarding the density of forkable strings we explicitly computed the number of 2-forkable strings of various sizes and densities. The experiments were run on a cluster of 4 servers equipped with Hexa-core Intel Xeon E5-2420 at 1.90GHz, 16GB RAM, and one 1TB SATA disk, running CentOS 7 Linux.

Our results are presented in Figure 8. As one can observe, as n grows the ratio of 2-forkable strings decays (for $\epsilon > 0$).

4.3 Common Prefix

Recall that the chains constructed by honest players during an execution of π_{SPoS} correspond to tines of a fork, as defined and studied in the previous sections. The random assignment of slots to stakeholders given by $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ guarantees that the coordinates of the associated characteristic string w follow the binomial distribution with probability equal to the adversarial stake. Thus Theorem 4.11 establishes that no execution of the protocol π_{SPoS} can induce two tines (chains) of maximal length with no common prefix.

In the context of π_{SPoS}, however, we wish to establish a much stronger common prefix property: The chains reported by any two honest players must have a "recent" common prefix, in the sense that removing a small number of blocks from the shorter chain results in a prefix of the longer chain.

To formally articulate and prove this property, we introduce some further definitions regarding tines and forks. We say that a tine is *honest* if it ends with an honest vertex. We also borrow the

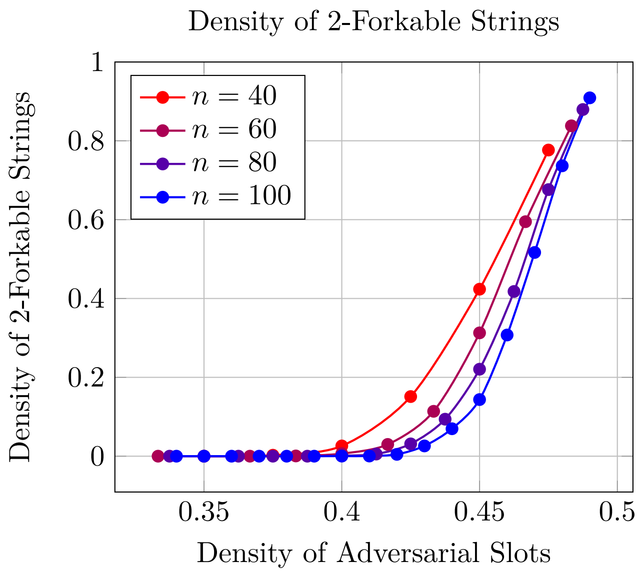


Figure 8: Graph of the density of 2-forkable strings as a function of the density of adversarial slots for $n = 40, 60, 80, 100$.

"truncation operator," described earlier in the paper for chains: for a tine t we let $t^{\lceil k}$ denote the tine obtained by removing the last k edges; if $\text{length}(t) \leq k$, we define $t^{\lceil k}$ to consist solely of the root. Finally, let F be a fork for a string $w \in \{0,1\}^*$. For two honest tines t_1 and t_2 of F, define their *divergence* to be the quantity

$$\text{div}(t_1, t_2) = \min_i(\text{length}(t_i) - \text{length}(t_1 \cap t_2)),$$

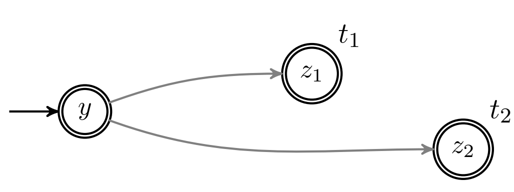
where $t_1 \cap t_2$ denotes the common prefix of t_1 and t_2. (Observe that if $\text{div}(t_1, t_2) \leq k$ and, say $\text{length}(t_1) \leq \text{length}(t_2)$, the tine $t_1^{\lceil k}$ is a suffix of t_2.) Then define the divergence of w to be the maximum such divergence over all pairs of honest tines over all possible forks for w:

$$\text{div}(w) = \max_{F \vdash w} \max_{\substack{t_1, t_2 \\ \text{honest} \\ \text{tines of } F}} \text{div}(t_1, t_2).$$

We first establish that a string with large divergence must have a large forkable substring. We then apply this in Theorem 4.19 below to conclude that characteristic strings arising from π_{SPoS} are unlikely to have large divergence and, hence, possess the common prefix property.

Theorem 4.18. *Let $w \in \{0,1\}^*$ with $\text{div}(w) \geq k$. Then there is a forkable substring $\check{w}$ of w with $|\check{w}| \geq k$.*

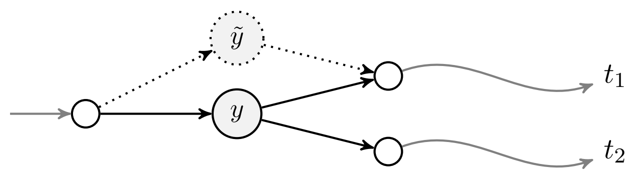
Proof. Consider a fork $F \vdash w$ and a pair of honest tines (t_1, t_2) so that $\text{div}(t_1, t_2) = \text{div}(w)$; we assume the tines are identified so that $\text{length}(t_1) < \text{length}(t_2)$. Let z_i denote the last vertex on the tine t_i and let y denote the last vertex on the tine $t_1 \cap t_2$ as in the diagram below.



Without loss of generality we may assume that $\ell(z_2)$, the honest index labeling t_2, is in fact the first honest index in w appearing after $\ell(z_1)$. To justify this, let β denote the first honest index of w after $\ell(z_1)$ and let x denote the unique vertex of F for which $\ell(x) = \beta$. Note that $\mathbf{d}(x) > \mathbf{d}(z_1)$. If the tine t ending at x shares an edge with t_1 after y, then t is disjoint from t_2 after y and it follows that $\operatorname{div}(t, t_2) > \operatorname{div}(t_1, t_2)$, a contradiction. Thus t shares no edges with t_1 after y, and $\operatorname{length}(t) > \operatorname{length}(t_1)$; it follows that $\operatorname{div}(t_1, t) = \operatorname{div}(t_1, t_2)$ and we may assume $t_2 = t$ in the remainder of the argument.

Let $\alpha = \ell(y)$ denote the label of y and, as indicated above, let $\beta = \ell(z_2)$. We wish to show that the string $\check{w} = w_{\alpha+1} \dots w_{\beta-1}$ is forkable. Our strategy will be to construct a flat fork for $\check{w}$ by showing that—after applying some minor restructuring to F—treating the vertex y as the root for the portion of F labeled with indices of $\check{w}$ yields a flat fork.

Observe, that y cannot be adversarial: otherwise it is easy to construct a fork $\tilde{F} \vdash w$ and a pair of tines that achieve larger divergence. Specifically, construct $\tilde{F}$ from F by adding a new (adversarial) vertex $\tilde{y}$ to F for which $\ell(\tilde{y}) = \ell(y)$, adding an edge to $\tilde{y}$ from the vertex preceding y, and replacing the edge of t_1 following y with one from $\tilde{y}$; then the other relevant properties of the fork are maintained, but the divergence of the resulting tines has increased by one. (See the diagram below.)



A similar argument implies that the fork $F_0 \vdash w_1 \dots w_\alpha$, obtained by including only the vertices with labels less than or equal to $\alpha = \ell(y)$, does not contain two distinct vertices of depth $\mathbf{d}(y)$. In the presence of another vertex $\tilde{y}$ (of F_0) with depth $\mathbf{d}(y)$, "redirecting" t_1 through $\tilde{y}$ (as in the argument above) would result in a fork with larger divergence.

Finally, we observe that the fork F may be "pinched" so that any tine terminating at an honest vertex associated with an index in $\check{w}$ passes through the vertex y. In particular, consider a tine t ending at such an honest vertex v (with $\alpha < \ell(v) < \beta$). Consider the vertex x on this tine for which $\mathbf{d}(x) = \mathbf{d}(y)$. If $x = y$, this tine already has the property we wish to ensure; otherwise $x \neq y$ and it follows that x is adversarial (as it has the same depth as y, which is honest). Consider now $\ell(x)$: note that it is not possible for $\ell(x) < \ell(y)$, because then x would be the end of a tine in F_0 with depth equal to that of y, which was ruled out above. It follows that $\ell(x) > \ell(y)$, in which case the edge in this tine terminating at x can safely be redirected to originate from y; note that this surgery does not change the depth of any vertices on the tine and maintains the increasing label condition. By applying this surgery (perhaps to multiple tines) we arrive at a fork F^* for w so that the induced subgraph of F^* including the vertex y and all vertices labeled with $\check{w}$ is a rooted tree. Subtracting $\ell(y)$ from the labels in F^* indeed yields a fork $\check{F}$ for the string $\check{w}$. Note that the tine t_1 in this fork contains the deepest honest vertex (as the next honest index of w appears beyond the end of the string $\check{w}$). By trimming all tines longer than t_1 so they have length exactly that of t_1, we obtain a flat fork for $\check{w}$, as desired.

Note, finally, that $|\check{w}|$ is at least the length of any particular tine in F^*; thus $|\check{w}| \geq \operatorname{length}(t_1) - \operatorname{div}(w) \geq k$. □

Theorem 4.19. *Let $k, R \in \mathbb{N}$ and $\epsilon \in (0,1)$. The probability that the π_{SPoS} protocol, when executed with a $(1-\epsilon)/2$ fraction of adversarial stake, violates the common prefix property with parameters*

k, R throughout a period of R is no more than $\exp(-\Omega(\sqrt{k}) + \ln R)$; the constant hidden by the $\Omega()$ notation depends only on ϵ.

Proof. The characteristic string $w \in \{0,1\}^R$ for such an execution of π_{SPoS} is determined by assigning each $w_i = 1$ independently with probability $(1-\epsilon)/2$.

Observe that an execution of π_{SPoS} violates the common prefix property with parameters k, R precisely when the fork F induced by this execution has $\mathrm{div}(F) \geq k$. Thus we wish to show that the probability that $\mathrm{div}(w) \geq k$ is no more than $\exp(-\sqrt{k} + \log R)$. Let Bad denote the event that $\mathrm{div}(w) \geq k$.

It follows from Theorem 4.18 that if $\mathrm{div}(w) \geq k$, there is a forkable substring $\check{w}$ of length at least k. Thus

$$\begin{aligned}\Pr[\mathsf{Bad}] &\leq \Pr\left[\exists \alpha, \beta \in \{1, R\} \text{ so that } \alpha + k - 1 \leq \beta \text{ and } w_\alpha \dots w_\beta \text{ is forkable}\right] \\ &= \sum_{1 \leq \alpha \leq R} \underbrace{\sum_{\alpha+k-1 \leq \beta \leq R-1} \Pr[w_\alpha \dots w_\beta \text{ is forkable}]}_{(*)} .\end{aligned}$$

According to Theorem 4.11 the probability that a string of length t drawn from this distribution is forkable is no more than $\exp(-c\sqrt{t})$ for a positive constant c. Note that for any $\alpha \geq 1$,

$$\sum_{t=\alpha+k-1}^{R} e^{-c\sqrt{t}} \leq \sum_{t=k}^{\infty} e^{-c\sqrt{t}} \leq \int_{k-1}^{\infty} e^{-c\sqrt{t}} = (2/c^2)(1 + c\sqrt{k-1})e^{-c\sqrt{k-1}} = e^{-\Omega(\sqrt{k})}$$

and it follows that the sum $(*)$ above is $\exp(-\Omega(\sqrt{t}))$. Thus

$$\Pr[\mathsf{Bad}] \leq R \cdot \exp(-\Omega(\sqrt{k})) \leq \exp(\ln R - \Omega(\sqrt{k}))\,,$$

as desired. □

4.4 Chain Growth and Chain Quality

We will start with the chain growth property.

Theorem 4.20. *The π_{SPoS} protocol satisfies the chain growth property with parameters $\tau = 1 - \alpha, s \in \mathbb{N}$ throughout an epoch of R slots with probability at least $1 - \exp(-\epsilon^2 s + \ln R)$ against an adversary holding an $\alpha - \epsilon$ portion of the total stake.*

Proof. Define $\mathsf{Ham}_a(\alpha)$ to be the event that the Hamming weight ratio of the characteristic string that corresponds to the slots $[a, a+s-1]$ is up to α. Given that the adversarial stake is $\alpha - \epsilon$, each of the k slots has probability $\alpha - \epsilon$ being assigned to the adversary and thus the probability that the Hamming weight is more than αs drops exponentially in s. Specifically, using the additive version of the Chernoff bound, we have that $\mathbf{Pr}[\neg\mathsf{Ham}_a(\alpha)] \leq \exp(-2\epsilon^2 s)$. It follows that,

$$\mathbf{Pr}[\mathsf{Ham}_\alpha] \geq 1 - \exp(-2\epsilon^2 s).$$

Given the above we know that when Ham_α happens there will be at least $(1-\alpha)s$ honest slots in the period of s rounds. Given that each honest slot enables an honest party to produce a block, all honest parties will advance by at least that many blocks. Using a union bound, it follows that the speed coefficient can be set to $\tau = (1-\alpha)$ and it is satisfied with probability at least $1 - \exp(-2\epsilon^2 s + \ln(R))$. □

Having established the chain growth property we now turn our attention to the chain quality property. Recall that the chain-quality property parameterized with k and it states that every k blocks in a chain observed at a certain slot the blocks corresponding to a set of stakeholders that hold cumulative stake ratio β are $\tau\beta$. In the next theorem we establish bounds for the parameter τ.

Theorem 4.21. *The π_{SPoS} protocol satisfies the chain quality property with parameters $\mu = \alpha/(1-\alpha)$, $k \in \mathbb{N}$ throughout an epoch of R slots with probability at least*

$$1 - \exp\bigl(-\epsilon^2(1-\alpha)^{-1}k + \ln R\bigr)$$

where $\alpha - \epsilon$ is the ratio of the cumulative stake of the set of malicious stakeholders.

Proof. First, using a similar argumentation as in the chain growth Theorem 4.20, we know that in a segment of s rounds the honest parties would advance by at least $(1-\alpha)s$ blocks. Furthermore the adversary can produce at most αs blocks in the same period. It follows that in the chain of any honest party one would find at most $\alpha/(1-\alpha)$ ratio of blocks originating from the adversary with probability $1 - \exp(-\epsilon^2 s + \ln R)$ among the blocks produced in the period that corresponds to that segment. It suffices to choose $s \geq (1-\alpha)^{-1}k$. In this case we know that there will be at least k blocks produced in any segment of s rounds. $\square$

5 Our Protocol: Dynamic Stake

5.1 Using a Trusted Beacon

In the static version of the protocol in the previous section, we assumed that stake was static during the whole execution (i.e., one epoch), meaning that stake changing hands inside a given epoch does not affect leader election. Now we put forth a modification of protocol π_{SPoS} that can be executed over multiple epochs in such a way that each epoch's leader election process is parameterized by the stake distribution at a certain designated point of the previous epoch, allowing for change in the stake distribution across epochs to affect the leader election process. As before, we construct the protocol in a hybrid model, enhancing the $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ ideal functionality to now provide randomness and auxiliary information for the leader election process throughout the epochs (the enhanced functionality will be called $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$). We then discuss how to implement $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ using only $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ and in this way reduce the assumption back to the simple common random string selected at setup.

Before describing the protocol for the case of dynamic stake, we need to explain the modification of $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ so that multiple epochs are considered. The resulting functionality, $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$, allows stakeholders to query it for the leader selection data specific to each epoch. $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ is parameterized by the initial stake of each stakeholder before the first epoch e_1 starts; in subsequent epochs, parties will take into consideration the stake distribution after the previous epoch's first $R-k$ slots, where k is the number of slots needed to achieve common prefix as supplied by the parties. Notice that it is necessary to consider the stake distribution of previous epochs only in the slots where it is guaranteed that common prefix is achieved. In any case, the functionality $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ provides only a random string and leaves the interpretation according to the stakeholder distribution to the party that is calling it. The functionality $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ is defined in Figure 9.

We now describe protocol π_{DPoS}, which is a modified version of π_{SPoS} that updates its genesis block B_0 (and thus the leader selection process) for every new epoch. The protocol also adopts an adaptation of the static $\mathsf{maxvalid}_S$ function, defined so that it narrows selection to those chains which share common prefix. Specifically, it adopts the following rule, parameterized by a prefix length k:

Functionality $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$

$\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ is parameterized by the public keys and respective stakes of the stakeholders $\mathbb{S}_0 = \{(\mathsf{vk}_1, s_1^0), \ldots, (\mathsf{vk}_n, s_n^0)\}$ before epoch e_1 starts, a distribution $\mathcal{D}$ and a leader selection function F. $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ interacts with stakeholders $U_1, \ldots, U_n$ as follows:

- Upon receiving $(\mathsf{genblock\_req}, U_i)$ from stakeholder U_i it operates as functionality $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ on that message.
- Upon receiving $(\mathsf{genblock\_req}, U_i, e_j, \mathbb{S}_j)$ from stakeholder U_i, if $j \geq 2$ is the current epoch, $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ proceeds as follows. If $B_0^j = \emptyset$, $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ samples $\rho^j \leftarrow \mathcal{D}$ and sets $B_0^j = \langle \mathbb{S}_j, \rho^j \rangle$. Finally, $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ sends $(\mathsf{genblock}, B_0^j, \mathsf{F})$ to U_i.

Figure 9: Functionality $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$.

Function $\mathsf{maxvalid}(\mathcal{C}, \mathbb{C})$. Returns the longest chain from $\mathbb{C} \cup \{\mathcal{C}\}$ that does not fork from $\mathcal{C}$ more than k blocks. If multiple exist it returns $\mathcal{C}$, if this is one of them, or it returns the one that is listed first in $\mathbb{C}$.

Protocol π_{DPoS} is described in Figure 10 and functions in the $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$-hybrid model.

Protocol π_{DPoS}

π_{DPoS} is a protocol run by a set of stakeholders, initially equal to $U_1, \ldots, U_n$, interacting among themselves and with $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ over a sequence of L slots $S = \{sl_1, \ldots, sl_L\}$. π_{DPoS} proceeds as follows:

1. **Initialization** When π_{SPoS} starts, each stakeholder $U_i \in \{U_1, \ldots, U_n\}$ sends $(\mathsf{genblock\_req}, U_i)$ to $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$, receiving $(\mathsf{genblock}, B_0, \mathsf{F})$ as answer. U_i sets an internal blockchain $\mathcal{C} = B_0$ and a initial internal state $st = H(B_0)$.
2. **Chain Extension** For every slot $sl \in S$, every online stakeholder U_i performs the following steps:
   (a) If a new epoch e_j, with $j \geq 2$, has started, U_i defines $\mathbb{S}_j$ to be the stakeholder distribution $R-k$ blocks into the epoch e_{j-1} as reflected in $\mathcal{C}$ and sends $(\mathsf{genblock\_req}, U_i, e_j, \mathbb{S}_j)$ to $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$, receiving $(\mathsf{genblock}, B_0^j, \mathsf{F})$ as answer. U_i includes B_0^j in the auxiliary data to be included in the next block to be produced, $\mathsf{aux} = B_0^j$, and parameterizes the leader selection function F with ρ^j from $B_0^j = \langle \mathbb{S}_j, \rho^j \rangle$.
   (b) Collect all valid chains received via broadcast into a set $\mathbb{C}$, verifying that for every chain $\mathcal{C}' \in \mathbb{C}$ and every block $B' = (st', d', sl', \sigma') \in \mathcal{C}'$ it holds that $\mathsf{Vrf}_{\mathsf{vk}'}(\sigma', (st', d', sl')) = 1$, where vk' is the verification key of the stakeholder $U' \leftarrow \mathsf{F}(\rho^{j'}, sl')$ with F parameterized by $e_{j'}$ be the epoch in which the slot to which B' belongs to (as determined by sl'). U_i calls the function $\mathsf{maxvalid}(\mathcal{C}, \mathbb{C})$ to select a new internal chain $\mathcal{C} \in \mathbb{C}$ and sets state $st = H(\text{head}(\mathcal{C}))$.
   (c) If U_i is the slot leader determined by $\mathsf{F}(\rho^j, sl)$ in the current epoch e_j, it generates a new block $B = (st, d, sl, \sigma)$ where st is its current state, $d \in \{0,1\}^*$ is data including aux and $\sigma = \mathsf{Sign}_{\mathsf{sk}_i}(st, d, sl)$ is a signature on (st, d, sl). U_i extends $\mathcal{C}$ by appending B, obtains $\mathcal{C} = \mathcal{C}||B$ and broadcasts the new $\mathcal{C}$.

Figure 10: Protocol π_{DPoS}

5.2 Simulating a Trusted Beacon

While protocol π_{DPoS} handles multiple epochs and takes into consideration changes in the stake distribution, it still relies on $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ to perform the leader selection process. In this section, we show how to implement $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ through Protocol π_{DLS}, which allows the stakeholders to compute the randomness and auxiliary information necessary in the leader election.

Recall, that the only essential difference between $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ and $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ is the continuous generation of random strings $\rho^2, \rho^3, \ldots$ for epochs $e_2, e_3, \ldots$. The idea is simple, protocol π_{DLS} will use a coin tossing protocol to generate unbiased randomness that can be used to define the values $\rho^j, j \geq 2$ bootstrapping on the initial random string and initial honest stakeholder distribution. However, notice that the adversary could cause a simple coin tossing protocol to fail by aborting. Thus, we build a coin tossing scheme with "guaranteed output delivery."

Commitments and Coin Tossing. A coin tossing protocol allows two or more parties to obtain a uniformly random string. A classic approach to construct such a protocol is by using commitment schemes. In a commitment scheme, a *committer* carries out a *commitment phase*, which sends evidence of a given value to a *receiver* without revealing it; later on, in an *opening phase*, the committer can send that value to the receiver and convince it that the value is identical to the value committed to in the commitment phase. Such a scheme is called *binding* if it is hard for the committer to convince the receiver that he was committed to any value other than the one for which he sent evidence in the commitment phase, and it is called *hiding* if it is hard for the receiver to learn anything about the value before the opening phase. We denote the commitment phase with randomness r and message m by $\mathsf{Com}(r, m)$ and the opening as $\mathsf{Open}(r, m)$.

In a standard two-party coin tossing protocol [5], one party starts by sampling a uniformly random string u_1 and sending $\mathsf{Com}(r, u_1)$. Next, the other party sends another uniformly random string u_2 in the clear. Finally, the first party opens u_1 by sending $Open(r, u_1)$ and both parties compute output $u = u_1 \oplus u_2$. Note, however, that in this classical protocol the committer may selectively choose to "abort" the protocol (by not opening the commitment) once he observes the value u_2. While this is an intrinsic problem of the two-party setting, we can avoid this problem in the multi-party setting by relying on a verifiable secret sharing scheme and an honest majority amongst the protocol participants.

Verifiable Secret Sharing (VSS). A secret sharing scheme allows a *dealer* P_D to split a secret σ into n *shares* distributed to parties $P_1, \ldots, P_n$, such that no adversary corrupting up to t parties can recover σ. In a Verifiable Secret Sharing (VSS) scheme [10], there is the additional guarantee that the honest parties can recover σ even if the adversary corrupts the shares held by the parties that it controls and even if the dealer itself is malicious. We define a VSS scheme as a pair of efficient dealing and reconstruction algorithms $(\mathsf{Deal}, \mathsf{Rec})$. The dealing algorithm $\mathsf{Deal}(n, \sigma)$ takes as input the number of shares to be generated n along with the secret σ and outputs shares $\sigma_1, \ldots, \sigma_n$. The reconstruction algorithm Rec takes as input shares $\sigma_1, \ldots, \sigma_n$ and outputs the secret σ as long as no more than t shares are corrupted (unavailable shares are set to $\perp$ and considered corrupted). Schoenmakers [24] developed a simple VSS scheme based on discrete logarithms suitable for our purposes.

Constructing Protocol π_{DLS}. The main problem to be solved when realizing $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ with a protocol run by the stakeholders is that of generating uniform randomness for the leader selection process while tolerating adversaries that may try to interfere by aborting or feeding incorrect information to parties. In order to generate uniform randomness ρ^j for epoch e_j, $j \geq 2$, the elected stakeholders for epoch e_{j-1} will employ a coin tossing scheme for which all honest parties are guaranteed to receive output as long as there is an honest majority. The protocol has two stages, commit and reveal which are split into phases. The stages of the protocol are presented in

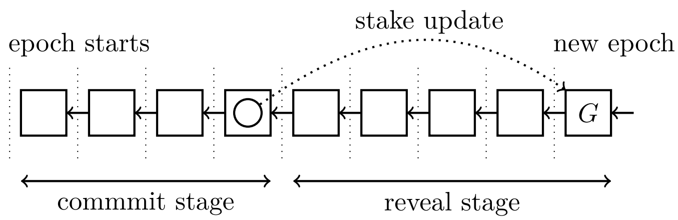


Figure 11: The two stages of the protocol π_{DPoS} that use the blockchain as a broadcast channel.

Figure 11. In the *Commitment Phase* covers the whole commitment stage, and proceeds as follows: for $1 \leq i \leq n$, stakeholder U_i samples a uniformly random string $u_i \in \{0,1\}^{R \log \tau}$ and randomness r_i for the underlying commitment scheme, generates shares $\sigma_1^i, \ldots, \sigma_n^i$, and posts $\mathsf{Com}(r_i, u_i)$ to the blockchain together with the encryptions of the all the shares under the public-key of each respective shareholder. After $4k$ slots, players remove the k most recent blocks of their chain, and if commitments from a majority of stakeholders are posted on the blockchain and shares from a majority of stakeholders have been received, the reveal stage starts (in the other case the protocol halts). In the reveal stage there are two phase: the *Reveal Phase* and the *Recovery Phase*. In the reveal phase, for $1 \leq i \leq n$, stakeholder U_i posts $\mathsf{Open}(r_i, u_i)$ to the blockchain. After $4k$ slots players remove the most recent k blocks and identify all stakeholders that have issued openings of the form $\mathsf{Open}(r_i, u_i)$. In the final *Recovery Phase*, lasting $2k$ slots, if a stakeholder U^a that initially submitted a commitment is identified as not posting an opening to its commitment, the honest parties can post all shares $\sigma_1^a, \ldots, \sigma_n^a$ in order to use $\mathsf{Rec}(\sigma_1^a, \ldots, \sigma_n^a)$ to reconstruct u^a. Finally, each stakeholder uses the values u_i obtained in the second round to compute $\rho^j = \sum_i u_i$. Protocol π_{DLS} is described in figure Figure 12. We remark that it is possible to run the reveal and recovery phases in parallel, however for improved efficiency we choose to run them sequentially.

Protocol π_{DLS}

π_{DLS} is a protocol run by stakeholders $U_1, \ldots, U_n$ interacting among themselves over a sequence of L slots $S = \{sl_1, \ldots, sl_L\}$ and proceeds as follows for every epoch e_j that lasts $R = 10k$ slots:

1. **Commitment Phase** ($4k$ slots) When epoch e_j starts, for $1 \leq i \leq n$, stakeholder U_i samples a uniformly random string u_i and randomness r_i for the underlying commitment scheme, generates shares $\sigma_1^i, \ldots, \sigma_n^i \leftarrow \mathsf{Deal}(n, u_i)$ and encrypts each share σ_k^i under stakeholder U_k's public-key. Finally, U_i posts the encrypted shares and commitments $\mathsf{Com}(r_i, u_i)$ to the blockchain.
2. **Reveal Phase** ($4k$ slots) After slot $4k$, for $1 \leq i \leq n$, stakeholder U_i opens its commitment by posting $\mathsf{Open}(r_i, u_i)$ to the blockchain, provided that the chain $\mathcal{C}^{\lceil k}$ contains commitments from the majority of U_i.
3. **Recovery Phase** ($2k$ slots) After slot $8k$, for any stakeholder U^a that has not participated in the reveal phase, i.e., it has not posted in $\mathcal{C}^{\lceil k}$ an $\mathsf{Open}(r_a, u_a)$ message, for $1 \leq i \leq n$, U_i submits its share σ_i^a for insertion to the blockchain. When all shares $\sigma_1^a, \ldots, \sigma_n^a$ are available, each stakeholder U_i can compute $\mathsf{Rec}(\sigma_1^a, \ldots, \sigma_n^a)$ to reconstruct u_a (independently of whether U^a opens the commitment or not).
   - Given input $(\mathsf{genblock\_req}, U_i, e_j, \mathbb{S}_j)$, U_i uses the values above to compute $\rho^j = \sum_{l \in \mathbb{L}} u_l$ where $\mathbb{L}$ is the subset of stakeholders that participated in epoch e_j. It returns $(\mathsf{genblock}, B_0, \mathbb{S}_j)$ with $B_0 = (\mathbb{S}_j, \rho^j)$.

Figure 12: Protocol π_{DLS}.

Security Proof Sketch. (Reducing $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ to $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ via protocol π_{DLS}). As before, we consider an adversary who holds a strict minority portion of the total stake. We set the epoch to be $R = 10k$ and observe that in the first $4k$ slots, i.e., the Commitment Phase, the Chain Growth property proven in Theorem 4.20 guarantees that at least $2k$ blocks will be added to the chain of all honest parties, the common prefix property, proven in Theorem 4.19, ensures that the remaining blockchain after chopping off k blocks will have at least k blocks and the chain quality property proven in Theorem 4.21 guarantees that a constant fraction of these blocks were generated by honest stakeholders with overwhelming probability. Thus, we know that at least one honest block containing all the honest parties' commitments was generated and included in the joint view of all honest parties. By a similar argument, by the end of the Reveal Phase, we are guaranteed to have all openings included in the blockchain and agreed upon by the honest stakeholders. In case a stakeholder does not post a valid opening to its commitment, yet by the same argument (but without needing common prefix), we have a guarantee that by the end of the Recovery at least $n/2 + 1$ shares will be posted (due to the honest majority) and agreed upon by the honest stakeholders, allowing them to recover the original input to the unopened commitments. Notice that if a majority of the stakeholders are honest, they either obtain enough values u_l to compute a uniformly random string ρ^j by the end of the reveal phase or manage to recover such value if the respective stakeholders do not open their commitments.

5.3 Robust Transaction Ledger

Recall that in the dynamic stake case, we would have to conceive a way to prevent deep forks. To see this, consider a player who is offline and joins the system after a number of epochs have passed. Even if in the system execution the current set of stakeholders satisfies honest majority, it could be the case that honest majority is violated in one of the previous epochs by this time and hence the adversary may produce an alternative history consistent with the view of an honest party. In order to capture the interaction between security and the modification of stake we introduce the following property.

Definition 5.1. *Consider two slots sl_1, sl_2, an honest player U and an execution $\mathcal{E}$. The stake shift w.r.t. U between sl_1, sl_2 is the statistical distance of the two weighted by stake distributions that are defined using the stake reflected in the chain $\mathcal{C}$ of U in the most recent blocks before sl_1 and sl_2 respectively as reflected in the execution $\mathcal{E}$.*

Taking into account the definition above we can now express the following theorem about the common prefix property.

Theorem 5.2. *Fix parameters $k, R, L \in \mathbb{N}, \epsilon, \sigma \in (0,1)$. Let R be the epoch length and L the total lifetime of the system. Assume the adversary is restricted to $\frac{1-\epsilon}{2} - \sigma$ relative stake and that the π_{SPOS} protocol satisfies the common prefix property with parameters R, k and probability of error ϵ_{CP}, the chain quality property with parameters $\mu \geq 1/k, k$ and probability of error ϵ_{CQ} and the chain growth property with parameters $\tau \geq 1/2, k$ and probability of error ϵ_{CG}.*

Then, the π_{DPOS} protocol satisfies the persistence with parameters k and liveness with parameters $u = 2k$ throughout a period of L slots with probability $1 - R(\epsilon_{\mathsf{CQ}} + \epsilon_{\mathsf{CG}} + \epsilon_{\mathsf{CG}})$, assuming that σ is the maximum stake shift over $6k$ slots, corruption delay $D \geq 2R - 4k$ and no honest player is offline for more than k slots.

Proof. (sketch) Observe that with probability of error $\epsilon_{\mathsf{CQ}} + \epsilon_{\mathsf{CG}} + \epsilon_{\mathsf{CG}}$ the π_{SPOS} protocol executed in the first epoch, given the assumptions imposed to the environment, will enable the parties to use the blockchain as a broadcast channel to simulate the trusted beacon and produce the randomness required to seed the leader election in the next epoch (this combines Theorems 4.19, 4.21, 4.20 and the results of the previous section). The corruption delay of $D \geq 2R - 4k$ will ensure that the adversary does not receive an adaptive corruption advantage for learning the next beacon value ahead of time. It follows that with probability $1 - (L/R)(\epsilon_{\mathsf{CQ}} + \epsilon_{\mathsf{CG}} + \epsilon_{\mathsf{CG}})$ all epochs in the lifetime of the system will be seeded correctly and the π_{SPOS} protocol can be bootstrapped and will continue to operate properly in each next epoch. □

5.4 Input Endorsers

We next present an extension of our basic protocol that assigns two different roles to stakeholders and introduces our incentive structure. As before in each epoch there is a set of elected stakeholders that runs the secure multiparty coin flipping protocol and are the slot leaders of the epoch. Together with those there is a (not necessarily disjoint) set of stakeholders called the endorsers. Now each slot has two types of stakeholders associated with it; the slot leader who will issue the block as before and the slot *endorser* who will endorse the input to be included in the block. Moreover, contrary to slot leaders, we can elect multiple slot endorsers, say m, for each slot. While this seems like an insignificant modification it gives us a room for improvement because of the following reason: endorsers' contributions will be acceptable even if they are d slots late, where $d \in \mathbb{N}$ is a parameter.

Note that in case no valid endorser input is available when the slot leader is about to issue the block, the leader will go ahead and issue an empty block, i.e., a block without any actual inputs (e.g., transactions in the case of a transaction ledger). Note that slot endorsers just like slot leaders are selected by weighing by stake and thus they are a representative sample of the stakeholder population. In the case of a transaction ledger the same transaction might be included by many input endorsers simultaneously. In case that a transaction is multiply present in the blockchain its first occurrence only will be its "canonical" position in the legder.

The enhanced protocol, π_{DPOSwE}, can be easily seen to have the same persistence and liveness behaviour as π_{DPOS}: the modification with endorsers does not provide any possibility for the adversary to prevent the chain from growing, accepting inputs, or being consistent. However, if we measure chain quality in terms of number of *endorsed inputs* included this produces a more favorable result: it is easy to see that the number of endorsed inputs originating from a set of stakeholders S in any k-long portion of the chain is proportional to the relative stake of S with high probability. This stems from the fact that it is sufficient that a single honest block is created for all the endorsed inputs of the last d slots to be included in it. Assuming $d \geq 2k$, any set of stakeholders S will be an endorser in a subset of the d slots with probability proportional to its cumulative stake, and thus the result follows.

6 Incentive Structure

While the fundamental blockchain analysis we perform is in the cryptographic setting of [12], we include in this section a discussion regarding the incentive structure of our system. For these purposes we adopt a standard game-theoretic framework, analogous to those successfully applied for other blockchains (see [14] for a recent analysis of bitcoin).

As in bitcoin, shareholders that issue blocks are incentivized to participate in the protocol by collecting transaction fees. Contrary to bitcoin, of course, one does not need to incentivize shareholders to invest computational resources. Rather, *availability* is incentivized. Any shareholder, at minimum, must be online in the following circumstances.

- In the slot prior to a slot she is the elected shareholder so that she queries the network and obtains the currently longest blockchain.
- In the slot during which she is the elected shareholder so that she issues the block.
- In a slot during the commit stage of an epoch where she is supposed to issue the VSS commitment of her random string.
- In a slot during the reveal stage of an epoch where she is supposed to issue the required opening shares as well as the opening to her commitment.
- In general, in sufficient frequency, to check whether she is an elected shareholder for the next or current epoch.
- In a slot during which she is the elected input endorser so that she issues the endorsed input (e.g., the set of transactions).

In order to incentivize the above actions in the setting of a transaction ledger, fees can collected from those that issue transactions to be included in the ledger which can then be transfered to the block issuers. In bitcoin, for instance, fees can be collected by the miner that produces a block of transactions as a reward. In our setting, similarly, a reward can be given to the parties that are issuing blocks and endorsing inputs. The reward mechanism does not have to be immediate as advocated in [21]. In our setting, it is possible to collect all fees of transactions included in a sequence of k blocks in a pool and then distribute that pool to all shareholders that participated during these k slots. For example, all input endorsers that were active may receive reward proportional to the number of inputs they endorsed during the period of k rounds (independently of the actual number of transactions they endorsed).

Other ways to distribute transaction fees are also feasible (including the one that is used by bitcoin itself—even though the bitcoin method is known to be vulnerable to attacks, e.g., the selfing-mining attack).

We proceed to make the reward mechanism more explicit in the case that the endorsed inputs to the ledger are sequences of transactions. First we set the reward interval to match the epoch duration, i.e., $d = 6k$. Let $\mathcal{C}$ be a chain consisting of blocks $B_0, B_1, \ldots$. Consider the sequence of blocks of the μ-th epoch denoted by $B_1, \ldots, B_s$ with timestamps in $\{\mu d + 1, \ldots, (\mu + 1)d\}$ that contain an $r \geq 0$ sequence of endorsed inputs. Suppose that the total reward pool R, as defined by transaction fees, is equal to the sum of the transaction fees that are included in the endorsed inputs. If a transaction occurs multiple times (as part of different endorsed inputs) or even in conflicting versions, only the first occurrence of the transaction is taken into account (and is considered to be part of the ledger) in the calculation of R, where the total order used is induced by the order the endorsed inputs are included in $\mathcal{C}$. In the sequence of these blocks, we identify by $L_1, \ldots, L_d$ the slot leaders corresponding to the slots of the epoch and by $E_1, \ldots, E_r$ the input endorsers that contributed the sequence of r endorsed inputs. Subsequently, the i-th stakeholder U_i can claim[3] a reward up to the amount $(|\{j \mid U_i = E_j\}|/(2r) + |\{j \mid U_i = L_j\}|/(2d))R$.

Observe that the above reward mechanism has the following features: (i) it rewards elected committee members for just being committee members, independently of whether they issued a block or not, (ii) it rewards the input endorsers with the inputs that they have contributed. We proceed to show that our system is a δ-Nash (approximate) equilibrium, cf. [18, Section 2.6.6].

[3]Claiming a reward is performed by issuing a "coinbase" type of transaction at any point after k blocks in a subsequent epoch to the one that a reward is being claimed from.

Specifically, the theorem states that any party deviating from the protocol can get at most $(1+\delta)$ of the rewards compared to the participants following the protocol.

Proposition 6.1. *The honest strategy in the protocol is a δ-Nash equilibrium provided that all players command a stake less than $(1-\epsilon)/2-\sigma$ for some constants $\epsilon, \sigma \in (0,1)$ as in Theorem 5.2, for $\delta > 0$.*

Proof. (sketch) Consider a rational player U restricted as in the statement of the theorem, that engages in a protocol execution together with a number of other players that follow the protocol faithfully for a total number of L epochs. We will show that any deviation from the protocol will not result in substantially higher rewards for U. Observe that based on Theorem 5.2, no matter the strategy of U, the protocol will enable all honest users to get at least $(1-\delta)$ fraction of the rewards they are entitled to as slot leaders and input endorsers with overwhelming probability in k. The latter stems from persistence and liveness: at least a number of honest blocks will be included in each epoch and as a result all input endorsers will have an opportunity to get their rewards. Given the above, player U will be rewarded proportionally to how many times she is selected as a slot leader (independent of her strategy), and moreover proportionally to how many inputs she contributes as an endorser. It thus follows that any other strategy that U may follow will lead to the same, or smaller amount of rewards. □

7 Stake Delegation

As discussed in the previous section, stakeholders must be online in order to generate blocks when they are selected as slot leaders. However, this might be unattractive to stakeholders with a small stake in the system. Moreover, requiring that a majority of elected stakeholders participate in the coin tossing protocol for refreshing randomness introduces a strain on the on the stakeholders and the network, since it might require broadcasting and storing a large number of commitments and shares.

We mitigate these issues by providing a method for reducing the size of the group of stakeholders that engage in the coin tossing protocol. Instead of the elected stakeholders directly forming the committee that will run coin tossing, a group of delegates will act on their behalf. In more detail, we put forth a *delegation scheme*, whereby stakeholders will authorize other entities, called delegates, who may be stakeholders themselves, to represent them in the coin tossing protocol. A delegate may participate in the protocol only if it represents a certain number of stakeholders whose aggregate stake exceeds a given threshold. Such a participation threshold ensures that a "fragmentation" attack, that aims to increase the delegate population in order to hurt the performance of the protocol, cannot incur a large penalty.

7.1 Minimum Committee Size

To appreciate the benefits of delegation, recall that in the basic protocol (π_{DPoS}) a committee member selected by weighing by stake is honest with probability $1/2+\epsilon$ (this being the fraction of the stake held by honest players). Thus, the number of honest players selected by k invocations of weighing by stake is a binomial distribution. We are interested in the probability of a malicious majority, which can be directly controlled by a Chernoff bound. Specifically, if we let Y be the number of times that a malicious committee member is elected then

$$\begin{aligned}\Pr[Y \geq k/2] &= \Pr[Y \geq (1+\delta)(1/2-\epsilon)k] \\ &\leq \exp(-\min\{\delta^2,\delta\}(1/2-\epsilon)k/4) \\ &< \exp(-\delta^2(1/2-\epsilon)k/4)\end{aligned}$$

for $\delta = 2\epsilon/(1-2\epsilon)$. Assuming $\epsilon < 1/4$, it follows that $\delta < 1$.

Consider the case that $\epsilon = 0.05$; then we have the bound $\exp(-0.00138 \cdot k)$ which provides an error of 1/1000 as long as $k \geq 5000$. Similarly, in the case $\epsilon = 0.1$, we have the bound $\exp(-0.00625k)$ which provides the same error for $k \geq 1100$.

We observe that in order to withstand a significant number of epochs, say 2^{15} (which, if we equate a period with one day, will be 88 years), and require error probability 2^{-40}, we need that $k \geq 32648$.

In cases where the wealth in the system is not concentrated among a small set of stakeholders the above choice is bound to create a very large committee. (Of course, the maximum size of the committee is k.)

7.2 Delegation Scheme.

To facilitate a smaller committee size we introduce a simple delegation idea that can enable stakeholders to delegate the committee participation rights to other entities that they trust. The concept of delegation is simple: any stakeholder can allow a *delegate* to generate blocks on her behalf. In the context of our protocol, where a slot leader signs the block it generates for a certain slot, such a scheme can be implemented in a straightforward way based on *proxy signatures* [6].

A stakeholder can transfer the right to generate blocks by creating a *proxy signing key* that allows the delegate to sign messages of the form (st, d, sl_j) (i.e., the format of messages signed in Protocol π_{DPoS} to authenticate a block). In order to limit the delegate's block generation power to a certain range of epochs/slots, the stakeholder can limit the proxy signing key's valid message space to strings ending with a slot number sl_j within a specific range of values. The delegate can use a proxy signing key from a given stakeholder to simply run Protocol π_{DPoS} on her behalf, signing the blocks this stakeholder was elected to generate with the proxy signing key. This simple scheme is secure due to the *Verifiability* and *Prevention of Misuse* properties of proxy signature schemes, which ensure that any stakeholder can verify that a proxy signing key was actually issued by a specific stakeholder to a specific delegate and that the delegate can only use these keys to sign messages inside the key's valid message space, respectively.

We remark that while proxy signatures can be described as a high level generic primitive, it is easy to construct such schemes from standard digital signature schemes through delegation-by-proxy as shown in [6]. In this construction, a stakeholder signs a certificate specifying the delegates identity (e.g., its public key) and the valid message space. Later on, the delegate can sign messages within the valid message space by providing signatures for these messages under its own public key along with the signed certificate. As an added advantage, proxy signature schemes can also be built from aggregate signatures in such a way that signatures generated under a proxy signing key have essentially the same size as regular signatures [6].

An important consideration in the above setting is the fact that a stakeholder may want to withdraw her support to a stakeholder prior to its proxy signing key expiration. Observe that proxy signing keys can be uniquely identified and thus they may be revoked by keeping a certificate revocation list within the blockchain.

7.2.1 Eligibility threshold

Delegation as described above can ameliorate fragmentation that may occur in the stake distribution. Nevertheless, this does not prevent a malicious stakeholder from dividing its stake to multiple accounts and, by refraining from delegation, induce a very large committee size. To address this, as mentioned above, a threshold T, say 1%, may be applied. This means that any delegate representing less a fraction less than T of the total stake is automatically barred from being a committee member. This can be facilitated by redistributing the voting rights of delegates representing less than T to other delegates in a deterministic fashion (e.g., starting from those with the highest stake and breaking ties according to lexicographic order).

7.2.2 Size of the committee of delegates

Suppose that a committee has been formed, $C_1, \ldots, C_m$, from a total of k draws of weighing by stake. Each committee member will hold k_i such votes where $\sum_{i=1}^{m} k_i = k$. Based on the eligibility threshold above it follows that $m \leq T^{-1}$ (the maximum possible value is the case when all stake is distributed in T^{-1} delegates each holding T of the stake).

8 Attacks Discussion

We next discuss a number of practical attacks and indicate how they are reflected by our modeling and mitigated.

Double spending attacks In a double spending attack, the adversary wishes to revert a transaction that is confirmed by the network. The objective of the attack is to issue a transaction, e.g., a payment from an adversarial account holder to a victim recipient, have the transaction confirmed and then revert the transaction by, e.g., including in the ledger a second conflicting transaction. Such an attack is not feasible under the conditions of Theorem 5.2. Indeed, persistence ensures that once the transaction is confirmed by an honest player, all other honest players from that point on will never disagree regarding this transaction. Thus it will be impossible to bring the system to a state where the confirmed transaction is invalidated (assuming all preconditions of the theorem hold).

Transaction denial attacks In a transaction denial attack, the adversary wishes to prevent a certain transaction from becoming confirmed. For instance, the adversary may want to target a specific account and prevent the account holder from issuing an outgoing transaction. Such an attack is not feasible under the conditions of Theorem 5.2. Indeed, liveness ensures that, provided the transaction is attempted to be inserted for a sufficient number of slots by the network, it will be eventually confirmed.

Desynchronization attacks In a desynchronization attack, a shareholder behaves honestly but is nevertheless incapable of synchronizing correctly with the rest of the network. This leads to ill-timed issuing of blocks and being offline during periods when the shareholder is supposed to participate. Such an attack can be mounted by preventing the party's access to a time server or any other mechanism that allows synchronization between parties. Moreover, a desynchronization may also occur due to exceedingly long delays in message delivery. Our model allows parties to become desynchronized by incorporating them into the adversary. No guarantees of liveness and persistence are provided for desynchronized parties.

Eclipse attacks In an eclipse attack, message delivery to a shareholder is violated due to a subversion in the peer-to-peer message delivery mechanism. As in the case of desynchronization attacks, our model allows parties to be eclipse attacked by incorporating them into the adversary. No guarantees of liveness or persistence are provided for such parties.

51% attacks A 51% attack occurs whenever the adversary controls more than the majority of the stake in the system. It is easy to see that any sequence of slots in such a case is with very high probability forkable and thus once the system finds itself in such setting the honest stakeholders may be placed in different forks for long periods of time. Both persistence and liveness can be violated.

Nothing at stake and past majority attacks As stake moves our assumption is that only the *current* majority of stakeholders is honest. This means that past account keys (which potentially do not hold any stake at present) may be compromised. This leads to a potential vulnerability for any PoS system since a set of malicious shareholders from the past can build an alternative blockchain exploiting such old accounts and the fact that it is effortless to build such a blockchain. In light of Theorem 5.2 such attack can only occur against shareholders who are not frequently online to observe the evolution of the system or in case the stake shifts are higher than what is anticipated by the preconditions of the theorem. This is a special instance of the "nothing at stake" problem which refers in general to attacks against PoS blockchain systems that are facilitated by shareholders continuing simultaneously multiple blockchains exploiting the fact that little computational effort is needed to build a PoS blockchain. Provided that stakeholders are frequently online, nothing at stake is taken care of by our analysis of forkable strings: even if the adversary brute-forces all possible strategies to fork the evolving blockchain in the near future, there is none that is viable. It is also worth noting that, contrary to PoW-based blockchains, in our protocol it is infeasible to have a fork generated in earnest by two shareholders. This is because slots are uniquely assigned and thus at any given moment there is a single uniquely identified shareholder that is elected to advance the blockchain. Players following the longest chain rule will adopt the newly minted block (unless the adversary presents at that moment an alternative blockchain using older blocks).

Selfish-mining In this type of attack, an attacker withholds blocks and releases them strategically attempting to drop honestly generated blocks from the main chain. In this way the attacker reduces chain growth and increases the relative ratio of adversarially generated blocks. In conventional reward schemes, as that of bitcoin, this has serious implications as it enables the attacker to obtain a higher rate of rewards compared to the rewards it would be receiving in case it was following the honest strategy. Using our reward mechanism however, selfish mining attacks are neutralized. The intuition behind this, is that input endorsers, who are the entities that receive rewards proportionally to their contributions, cannot be stifled because of block withholding: any input endorser can have its contribution accepted for a sufficiently long period of time after its endorsement took place, thus ensuring it will be incorporated into the blockchain (due to sufficient chain quality and chain growth). Given that input endorsers' contributions are (approximately) proportional to their stake this ensures that reward distribution cannot be affected substantially by block withholding.

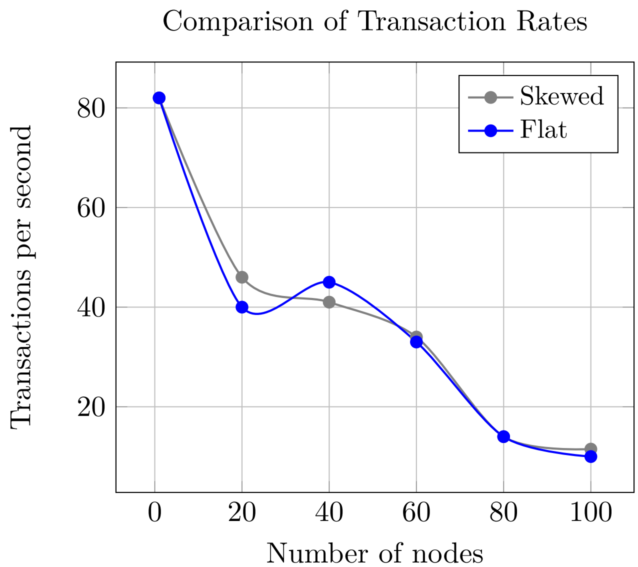


Figure 13: Graph of transaction rates for an Ouroboros prototype run over a wide area network with flat and skewed stake distributions.

9 Experimental Results

We have implemented a prototype instantiation of Ouroboros in Haskell in order to evaluate its concrete performance. More specifically, we have implemented Protocol π_{DPoS} using Protocol π_{DLS} to generate leader selection parameters (basically generating fresh randomness for the weighed stake sampling procedure). For this instantiation, we use the PVSS scheme of [24] implemented over the elliptic curve secp256r1. This PVSS scheme's share verification information includes a commitment to the secret, which is also used as the commitment specified in protocol π_{DLS}; this eliminates the need for a separate commitment to be generated and stored in the blockchain. In order to obtain better efficiency, the final output ρ of Protocol π_{DLS} is a uniformly random binary string of 32 bytes. This string is then used as a seed for a PRG (ChaCha in our implementation) and stretched into R random labels of $\log \tau$ bits corresponding to each slot in an epoch. The weighing by stake leader selection process is then implemented by using the random binary string associated to each epoch to perform the sequence of coin-flips for selecting a stakeholder. The signature scheme used for signing blocks is ECDSA, also implemented over curve secp256r1.

Experimental Setup. In our experiments we consider a total number of stakeholders that varies from 1 (for the sake of a centralized baseline) to 100 communicating through a peer-to-peer network. Although these numbers of stakeholders might seem low, notice that delegation can be used to limit the size of the committee of stakeholders actively participating in the protocol while still supporting a much larger number of stakeholders. We consider both situations where the stake is equally distributed among all stakeholders and situations with skewed stake distributions where a small number of stakeholders have most of the stake (resembling the distribution of computational power across mining pools in Bitcoin). The prototype implementation for each stakeholder is run on t2.large Amazon EC2 nodes equally spread among Asia, Europe, North America and South America. The slot duration is set to 20 seconds and the epoch length is $R = 6k$ slots, where $k = 6$.

Experiments. In our experiments we measure the maximum Transaction Per Second (TPS) rates that our protocol sustains in each situation described above. Apart from the nodes that implement stakeholders running the protocol, our experiments employ an extra node that acts as a Transaction Generator (henceforth referred to as TxGen). The stakeholders participating in the experiment are registered in the genesis block and TxGen starts by generating a number of initial transactions from each of the stakeholders distributing their initial stake in number of $maxTPS$ outputs to themselves, which form an output pool for each stakeholder. In the first phase, called *warm-up*, TxGen uses the outputs of each transaction pool as inputs to generate a number of $currentTPS$ transactions from each stakeholder to itself in regular intervals. These transactions (as well as future transactions in this experiments) are sent into the peer-to-peer network connecting the stakeholders. Once a transaction is confirmed (meaning that it appears in a block of depth k in the current blockchain), TxGen uses the output of this transaction as input to generate a new transaction with an output to the same stakeholder that owns the input. After the warm-up phase, the system is populated with a number of transactions that are both generated from the output pool of each stakeholder and from the outputs of confirmed transactions. In the next phase, called *interim*, the same transaction creation procedure continues but now TxGen increments a variable *counter* (initially set to 0) for each transaction (from/to any stakeholder) that is confirmed in the blockchain. In a final phase, called *cool-down*, no new transactions are generated at all but *counter* continues to be incremented for each confirmed transaction. By the end of the cool-down phase, a variable $realTPS$ is calculated by dividing *counter* by the total time elapsed from the beginning of the interim phase to the end of the cool-down phase. If $realTPS$ is bigger than $currentTPS$, the experiment is rerun with an incremented value of $currentTPS$ in order to measure the maximum TPS rate the system can sustain. Otherwise, the past $currentTPS$ value is considered as the highest TPS rate the system can achieve. Our results are summarized in Figures 13 and 14.

Known Prototype Caveats. Our implementation is a first rendering of a working prototype and we anticipate substantial improvements can be still attained. When analysing our current experimental results, one should keep in mind that our prototype implementation currently incurs several overheads that are not intrinsic to our protocol. One of the main issues is the naive peer-to-peer layer, which is not optimized and causes a heavy communication overhead as the number of nodes grows. These overheads explain why the TPS rates fall rapidly as the number of nodes increases. In an ideal network, the TPS of our system would approximate the average local TPS across all participating nodes, where local TPS corresponds to the performance of a node when is ran as a central transaction processor. Notice that the PVSS scheme from [24] requires $O(n^2)$ exponentiations for share generation and validation (where n is the size of the committee), which results in a concrete execution time that increases quadratically with committee size, limiting the size of the committees that our prototype implementation supports given the chosen slot duration. This limitation can be easily mitigated by using our delegation scheme with a threshold or devising a PVSS scheme with better computational complexity, which is an interesting open question.

10 Acknowledgements

We thank Ioannis Konstantinou who contributed in a preliminary version of our protocol. We thank Lars Brünjes for comments on previous drafts of the article. We thank George Agapov for the prototype implementation of our protocol.

Nodes	TPS; flat	TPS; skewed
1	82	82
20	40	46
40	45	41
60	33	34
80	14	14
100	10	11.5

Figure 14: Transaction rates for an Ouroboros prototype run over a wide area network with flat and skewed stake distributions.

References

[1] Noga Alon and Joel Spencer. *The Probabilistic Method*. Wiley, 3rd edition, 2008.

[2] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, volume 8642 of *Lecture Notes in Computer Science*, pages 538–557. Springer, 2014.

[3] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. *CoRR*, abs/1406.5694, 2014.

[4] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]y. *SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.

[5] Manuel Blum. Coin flipping by telephone. In Allen Gersho, editor, *Advances in Cryptology: A Report on CRYPTO 81, CRYPTO 81, IEEE Workshop on Communications Security, Santa Barbara, California, USA, August 24-26, 1981.*, pages 11–15. U. C. Santa Barbara, Dept. of Elec. and Computer Eng., ECE Report No 82-04, 1981.

[6] Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure proxy signature schemes for delegation of signing rights. *J. Cryptology*, 25(1):57–115, 2012.

[7] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. In *23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.

[8] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 585–605. Springer, 2015.

[9] Ittay Eyal and Emin Gun Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Angelos D. Keromytis, editor, *Financial Cryptography*, volume 7397 of *Lecture Notes in Computer Science*. Springer, 2014.

[10] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 427–437. IEEE Computer Society, 1987.

[11] Bryan Ford. Delegative democracy. Available online, http://www.brynosaurus.com/deleg/deleg.pdf, (Last Retrieved November 11th, 2016), 2002.

[12] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

[13] Charles M Grinstead and J Laurie Snell. *Introduction to Probability*. American Mathematical Society, 2nd edition, 1997.

[14] Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. Blockchain mining games. In Vincent Conitzer, Dirk Bergemann, and Yiling Chen, editors, *Proceedings of the 2016 ACM Conference on Economics and Computation, EC '16, Maastricht, The Netherlands, July 24-28, 2016*, pages 365–382. ACM, 2016.

[15] Aggelos Kiayias and Giorgos Panagiotakos. Speed-security tradeoffs in blockchain protocols. Cryptology ePrint Archive, Report 2015/1019, 2015. `http://eprint.iacr.org/2015/1019`.

[16] Tal Moran and Ilan Orlov. Proofs of space-time and rational proofs of storage. Cryptology ePrint Archive, Report 2016/035, 2016. `http://eprint.iacr.org/2016/035`.

[17] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf, 2008.

[18] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

[19] Karl J. O'Dwyer and David Malone. Bitcoin mining and its energy footprint. *ISSC 2014 / CIICT 2014, Limerick, June 26–27*, 2014.

[20] Sunoo Park, Krzysztof Pietrzak, Albert Kwon, Joël Alwen, Georg Fuchsbauer, and Peter Gazi. Spacemint: A cryptocurrency based on proofs of space. *IACR Cryptology ePrint Archive*, 2015:528, 2015.

[21] Rafael Pass. Cryptography and game theory. Securty and Cryptography for Networks, 2016, invited talk., 2016.

[22] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR Cryptology ePrint Archive*, 2016:454, 2016.

[23] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *CoRR*, abs/1507.06183, 2015.

[24] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August*

15-19, 1999, Proceedings, volume 1666 of *Lecture Notes in Computer Science*, pages 148–164. Springer, 1999.

A The Proof of Theorem 4.11

We return now to prove Theorem 4.11 in full generality (with forks of arbitrary structure). We recall the statement of the theorem:

Theorem 4.11, restated. *Let $\epsilon \in (0,1)$ and let w be a string drawn from $\{0,1\}^n$ by independently assigning each $w_i = 0$ with probability $(1+\epsilon)/2$. Then $\Pr[w \text{ is forkable}] = 2^{-\Omega(\sqrt{n})}$.*

As in Section 4.2, we place a probability distribution on $\{0,1\}^n$ by independently selecting each $w_i \in \{0,1\}$ so that

$$\Pr[w_i = 0] = \frac{1+\epsilon}{2} = 1 - \Pr[w_i = 1]\,.$$

As w is forkable if and only if $\text{margin}(w) \geq 0$, our goal is to show that

$$\Pr[\text{margin}(w) \geq 0] = 2^{-\Omega(\sqrt{n})}$$

when w is drawn with this distribution.

In preparation for the proof, we introduce some further terminology.

Definition A.1 (Component)**.** *Let $w \in \{0,1\}^*$ and consider a tine t of a fork $F \vdash w$; the* component *of t is the union of all tines t' so that $t \sim t'$. We write $[t]$ to denote the component associated with a tine t and treat components as subgraphs of F. For convenience, we let ϵ denote the empty tine consisting of the path of length zero containing only the vertex r; while $\epsilon \not\sim \epsilon$, we define $[\epsilon]$ to be a component containing only this trivial path; this component we call the* trivial *component.*

Note that all tines in a nontrivial component $[t]$ share the same initial edge from r (the root) and, in general, a fork has a unique description as a union of nontrivial components. We extend the notion of *margin* to components in the obvious way:

$$\text{margin}([t]) = \max_{t' \text{ a tine of } [t]} (\text{reserve}(t') - \text{gap}(t'))\,.$$

Note that gap() still refers to the longest tine $\hat{t}$ of the (entire) fork and that

$$\text{margin}(F) = \max_{t \not\sim \hat{t}} \text{margin}([t])\,.$$

For a fork F, we let $\mathbf{cm}_F$ denote the multiset of component margins:

$$\mathbf{cm}_F = \{\text{margin}([t])| \mid [t] \text{ a component of } F\}\,,$$

where the multiplicity of $k \in \mathbf{cm}(F)$ is given by the number of components with margin k. Thus $\mathbf{cm}(F)$ has cardinality equal to the number of components of F, including the trivial component. We remark that $[\hat{t}]$, the component of the longest tine of F, always has non-negative margin. Note finally that

$$\text{margin}(F) \geq 0 \Longleftrightarrow \mathbf{cm}_F = \{0\} \text{ or } \mathbf{cm}(F) \text{ has multiple non-negative elements}\,.$$

Components and fork extension. As in Section 4.2, we consider the relationship between forks for "neighboring" strings; specifically, consider two closed forks $F' \sqsubseteq F$ where $F' \vdash w$ and $F \vdash wa$, where $w \in \{0,1\}^*$ and a is a single symbol in $\{0,1\}$. In the case where $a = 1$, we have $F = F'$ (as the forks are assumed to be closed), and it is easy to see that the margin of each component $[t']$ of F' has increased by exactly one when viewed as a component in F (for the string $w1$). We write $\text{margin}_F([t']) = \text{margin}_{F'}([t']) + 1$, where we introduce the notation $\text{margin}_{\Box}()$ to denote the margin in a particular fork. Thus

$$\mathbf{cm}(F) = \{m + 1 \mid m \in \mathbf{cm}(F')\}\,,$$

where we understand this notation to mean that the multiplicity of $m+1$ in $\mathbf{cm}(F)$ is the same as that of m in $\mathbf{cm}(F')$.

The case when $a = 0$ is more delicate. Here F is obtained from F' by adding a path labeled with a string of the form 1^s0 to the end of a tine t' of F'. (In fact, it is easy to see that we may always assume that this is appended to an honest tine.) In order for this to be possible, $\text{gap}(t') \leq \text{reserve}(t')$ and, in particular, $\text{gap}(t') \leq s \leq \text{reserve}(t')$: for the first inequality, note that the depth of the new honest vertex must exceed that of the deepest current (honest) vertex in F so $s \geq \text{gap}(t')$; as for the second inequality, there are only $\text{reserve}(t')$ possible adversarial indices that may be added to t' so $s \leq \text{reserve}(t')$. We write $s = \text{gap}(t') + r$ and let t denote the tine (of F) resulting by extending t' in this way. The typical resulting $\text{margin}_F([t])$ is exactly $\text{margin}_{F'}([t']) - (r+1)$, as the length of the longest tine has increased by $r+1$; however, there is a special case arising when $\text{margin}_{F'}([t']) = r$; then $\text{margin}_F([t]) = 0$ (rather than -1), as the new honest tine is counted among those in $[t]$. The effect on other components is straightforward: if $t_0 \not\sim t'$, $\text{margin}_F([t_0]) = \text{margin}_{F'}([t_0]) - (r+1)$. (Note that the special case above accounts for the fact that the margin of $[\hat{t}]$, where $\hat{t}$ is the longest tine of a fork, must be non-negative.) A final remark: in case t' is the empty tine, this process actually introduces a new component in F. To summarize these cases,

$$\mathbf{cm}(F) = \begin{cases} \{m - (r+1) \mid m \in \mathbf{cm}(F')\} & \text{if margin}([t']) > r \text{ and } t' \neq \epsilon, \\ \{m - (r+1) \mid m \in \mathbf{cm}(F') \setminus \{r\}\} \cup \{0\} & \text{if margin}([t']) = r \text{ and } t' \neq \epsilon, \\ \{m - (r+1) \mid m \in \mathbf{cm}(F')\} \cup \{\text{margin}([\epsilon]) - (r+1)\} & \text{if margin}([t']) > r \text{ and } t' = \epsilon, \\ \{m - (r+1) \mid m \in \mathbf{cm}(F')\} \cup \{0\} & \text{if margin}([t']) = r \text{ and } t' = \epsilon. \end{cases}$$

Above, we adopt the convention that canonical set operations such as $\cup$ and $\setminus$ change the multiplicity of the relevant elements by no more than 1 (so that, for example, $\{1\} \cup \{0\} = \{0,0,1\} \setminus \{0\} = \{0,1\} \setminus \{2\} = \{0,1\}$).

While the full rule is complicated by the exotic cases when $t' = \epsilon$ or when r saturates $\text{margin}([t'])$, we single out the simple features that are directly relevant for the proof. For a multiset of integers M of cardinality at least 2, we let $\max_0(M)$ denote the maximum element of the multiset $M \setminus \max(M)$. Consider now circumstances where F is nontrivial (so that $|\mathbf{cm}(F)| > 1$), r is always chosen to be 1 (in which case the adversary chooses the most conservative possible extension of the tine that can still feasibly place the new honest vertex at an appropriate height), and $t' \neq \epsilon$ (which will occur with high probability except with a small number of exceptions at the beginning of a string), we note that when $\max_0(\mathbf{cm}(F')) > 0$ the resulting $\max_0(\mathbf{cm}(F))$ satisfies

$$\max\nolimits_0(\mathbf{cm}(F)) = \begin{cases} \max_0(\mathbf{cm}(F')) + 1 & \text{when } a = 1, \\ \max_0(\mathbf{cm}(F')) - 1 & \text{when } a = 0\,. \end{cases}$$

In fact, the very same rule applies when $\max_0(\mathbf{cm}(F)) < 0$, in which case there is a unique tine of F with non-negative margin. Note that any honest index *must* be added to this unique tine and, in this case, the remaining tines (which determine $\max_0()$) obey the simple ± 1 rule given above.

For the string $w_1, \ldots, w_n$ chosen with the probability distribution above, define the random variables

$$X_t = \begin{cases} \perp & \text{if } F \text{ trivial,} \\ \max\limits_{F \vdash w_1 \ldots w_t} \max_0 \mathbf{cm}(F) & \text{otherwise.} \end{cases}$$

Preparing to study the X_t, we recall two basic facts about the standard biased walk: Let $Z_i \in \{\pm 1\}$ (for $i = 1, 2, \ldots$) denote a family of independent random variables for which $\Pr[Z_i = 1] = (1 - \epsilon)/2$. Then the biased walk given by the variables $Y_t = \sum_i^t Z_i$ has the following two properties:

Constant escape probability. As $\epsilon > 0$, the probability that an infinite walk beginning at state 0 ever visits the state 0 again is a constant less than 1 (depending only on ϵ). Specifically, with constant probability $Y_t \neq 0$ for all $t > 0$. (See, e.g., [13, Chapter 12].)

Concentration (the Chernoff bound). Consider s steps of the biased walk beginning at state 0; then the resulting value is tightly concentrated around $-\epsilon s$. Specifically, $\mathbb{E}[Y_s] = -\epsilon S$ and

$$\Pr\left[Y_s > -\frac{\epsilon S}{2}\right] = 2^{-\Omega(S)}\,. \tag{5}$$

(The constant hidden in the $\Omega()$ notation depends only on ϵ. See, e.g., [1, Cor. A.1.14].)

Write $w = w^{(1)} \cdots w^{(\sqrt{n})}$ where $\lfloor \sqrt{n} \rfloor \leq |w^{(i)}| \leq \lceil \sqrt{n} \rceil$ for each i. Let $X^{(0)} = 0$ and $X^{(t)} = X_\ell$ where $\ell = |w^{(1)} \ldots w^{(t)}|$. Fix $\delta < \epsilon$ to be a small constant.

Let L denote the event that for each $t \geq \sqrt{n}$, $\mathrm{wt}(w_1 \ldots w_t) < t/2$. By the Chernoff bound above and a union bound over the $n - \sqrt{n}$ various values of t, we conclude that $\Pr[L] \geq 1 - n2^{-\Omega(\sqrt{n})}$, and we will condition on this event throughout the rest of the proof. Let $F_{\sqrt{n}} \sqsubseteq F_{\sqrt{n}+1} \sqsubseteq \ldots \sqsubseteq F_n$ denote any sequence of forks so that $F_t \vdash w_1 \cdots w_t$. Observe that when L occurs, no honest index after position $\sqrt{n}$ in w can possibly have been added to the trivial tine ϵ: it follows that the numbers of components in the forks $F_{\sqrt{n}}, \ldots$ are all the same. (Furthermore, the number of components in these forks is no more than $\sqrt{n}$, though we will not exploit this particular fact.)

We define three events based on the random variables $X^{(t)}$:

Hot We let Hot_t denote the event that $X^{(t)} \geq 2\delta\sqrt{n}$.

Volatile We let Vol_t denote the event that $-2\delta\sqrt{n} \leq X^{(t)} < 2\delta\sqrt{n}$.

Cold We let Cold_t denote the event that $X^{(t)} < -2\delta\sqrt{n}$.

Note that for each t, exactly one of these events occurs—they partition the probability space. Then we will establish that

$$\Pr[\mathsf{Cold}_{t+1} \mid \mathsf{Cold}_t] \geq 1 - 2^{-\Omega(\sqrt{n})}\,, \tag{6}$$

$$\Pr[\mathsf{Cold}_{t+1} \mid \mathsf{Vol}_t] \geq \Omega(\epsilon)\,, \tag{7}$$

$$\Pr[\mathsf{Hot}_{t+1} \mid \mathsf{Vol}_t] \leq 2^{-\Omega(\sqrt{n})}\,. \tag{8}$$

Note that the event Vol_0 occurs by definition. Assuming these inequalities, we observe that the system is very likely to eventually become cold, and stay that way. Note that the probability that the system ever transitions from volatile to hot is no more than $2^{-\Omega(\sqrt{n})}$ (as transition from Vol to Hot is bounded above by $2^{-\Omega(\sqrt{n})}$, and there are no more than $\sqrt{n}$ possible transition opportunities). Note, also, that while the system is volatile, it transitions to cold with constant probability during each period. In particular, the probability that the system is volatile for the entire process is no

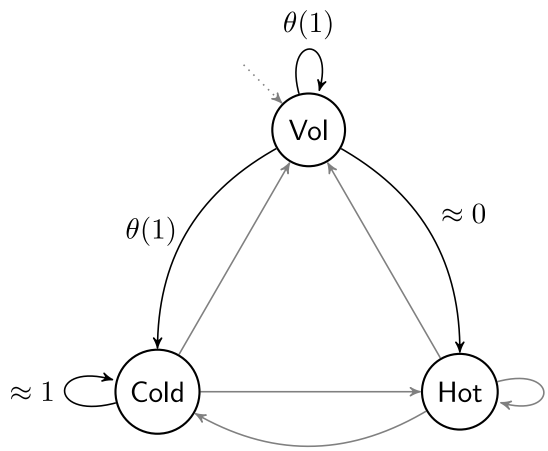


Figure 15: An illustration of the transitions between Cold, Vol, and Hot.

more that $2^{-\Omega(\sqrt{n})}$. Finally, note that the probability that the system ever transitions out of the cold state is no more than $2^{-\Omega(\sqrt{n})}$ (again, there are at most $\sqrt{n}$ possible times when this could happen, and any individual transition occurs with probability $2^{-\Omega(\sqrt{n})}$). It follows that the system is cold at the end of the process with probability $1 - 2^{-\Omega(\sqrt{n})}$.

Observe that when $\mathsf{Cold}_{\sqrt{n}}$ occurs, it follows that w is not forkable, as desired. The statements above now follow from reasoning analogous to that of Section 4.2.