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

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

July 5, 2019

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 also present a novel reward mechanism for incentivizing proof of stake protocols and we prove that, given this mechanism, honest behavior is an approximate Nash equilibrium, thus neutralizing attacks such as selfish mining. We also present initial evidence of the practicality of our protocol in real world settings by providing experimental results on transaction confirmation and processing.

1 Introduction

A primary consideration regarding the operation of blockchain protocols based on proof of work (PoW)—such as bitcoin [34]—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 indicated that the energy requirements of the protocol were comparable to that of a small country [36].

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. Fundamentally, the proof of work mechanism of bitcoin facilitates a type of robust 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 [25, 43].)

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 and IOHK. akiayias@inf.ed.ac.uk. Work partly performed while at the National and Kapodistrian University of Athens, supported by ERC project CODAMODA #259152. Work partly supported by H2020 Project #653497, PANORAMIX.

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

‡Aarhus University and IOHK, bernardo@bmdavid.com. 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 protocol 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 [7] as well as the sole mechanism for a blockchain protocol [6]. Although Bentov et al. showed that their protocols are secure against some classes of attacks, they do not provide a formal model for analyzing 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 [6] 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., [21]). 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 as well as the fact that the complexity of system maintenance should be sublinear in the total number of users/stakeholders.

Another alternative to PoW is the concept of *proof of space* [2, 24], which has been specifically investigated in the context of blockchain protocols [37]. 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 demand less energy. A related concept is *proof of space-time* (PoST) [32]. In all these cases, however, the protocol relies on an expensive physical resource (either storage or computational power).

The PoS Design challenge. A fundamental problem for PoS-based blockchain protocols is to simulate the leader election process. In order to achieve a fair randomized election among stakeholders, entropy must be introduced into the system, and mechanisms to introduce entropy may be prone to manipulation by the adversary. For instance, an adversary controlling a set of stakeholders may attempt to simulate the protocol execution trying different sequences of stakeholder participants so that it finds a protocol continuation that favors the adversarial stakeholders. This leads to a so called "grinding" vulnerability, where adversarial parties may use computational resources to bias the leader election.

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 [28], 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 adapted to reflect 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 (simple) secure multiparty implementation of a coin-flipping protocol to produce the randomness for the leader election process. This distinguishes our approach (and prevents so called "grinding attacks") 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 [6]. 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 elected stakeholders that will execute the protocol in the subsequent 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 synchronous in the sense that an upper bound can be determined during which any honest stakeholder is able to communicate with any other stakeholder, (2) a number of stakeholders drawn from the honest majority is available as needed to participate in each epoch, (3) the stakeholders do not remain offline for long periods of time, (4) the adaptivity of corruptions is subject to a small delay that is measured in rounds linear in the security parameter (or alternatively, the players have access to a sender-anonymous broadcast channel). At the core of our security arguments is a probabilistic argument regarding a combinatorial notion of "forkable strings" which we formulate, prove and also investigate experimentally. In our analysis we also distinguish *covert attacks*, a special class of general forking attacks. "Covertness" here is interpreted in the spirit of covert adversaries against secure multiparty computation protocols, cf. [3], where the adversary wishes to break the protocol but prefers not to be caught doing so. We show that covertly forkable strings are a subclass of the forkable strings with much smaller density; this permits us to provide two distinct security arguments that achieve different trade-offs in terms of efficiency and security guarantees. Our forkable string technique is a natural and general tool that may have applications beyond analysis of our specific PoS protocol.

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-mining* [25, 43] are mitigated by our design. The core idea behind the reward mechanism is to provide positive payoff for those protocol actions that cannot be stifled by a coalition of parties that diverges from the protocol. In this way, it is possible to show that under plausible assumptions—namely that certain protocol execution costs are small—following the protocol faithfully is an equilibrium when all players are rational.

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 stakeholders to delegate their "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 [27]), stakeholders have the ability to revoke their delegative appointment when they wish, independently of each other.

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 present evidence for the practical efficiency of our design. First we consider double spending attacks. For illustrative purposes, we perform a comparison with Nakamoto's analysis for bitcoin regarding transaction confirmation time with assurance 99.9%. Against covert adversaries, the transaction confirmation time is from 10 to 16 times faster than that of bitcoin, depending on the adversarial hashing power; for general adversaries confirmation time is from 5 to 10 times faster. Moreover, our concrete analysis of double-spending attacks relies on our combinatorial analysis of forkable and covertly forkable strings and applies to a much broader class of adversarial behavior than Nakamoto's more simplified analysis.[3] We then 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.

Related and Follow-up Work. In parallel to the development of Ouroboros, a number of other protocols were developed targeting various positions in the design space of distributed ledgers based on PoS. Sleepy consensus [8] considers a fixed stakeholder distribution (i.e., stake does not evolve over time) and targets a "mixed" corruption setting, where the adversary is allowed to be adaptive as well as perform fail-stop and recover corruptions in addition to Byzantine faults. It is straightforward to extend our analysis in this mixed corruption setting, cf. Remark 2; nevertheless, the resulting security can be argued only in the "corruptions with delay" setting, and thus is not fully adaptive; in follow up work it is shown how our analysis can be extended to incorporate both adaptive corruptions [22] as well as a more refined model of availability [4] that covers fail-stop and recover/sleepy corruptions. Snow White [9] addresses an evolving stakeholder distribution and uses a corruption delay mechanism similar to ours for arguing security. Contrary to our approach here the leader election of [9] enables a degree of "grinding", i.e., making it feasible to significantly bias any adversarially chosen high probability event but affords a much more efficient randomness generation process. In follow-up work [22] it is shown how it is possible to obtain full adaptive security and similarly efficient randomness generation extending the forkable strings analysis introduced in the present paper. Algorand [31] provides a distributed ledger following a Byzantine agreement *per block* approach that can withstand adaptive corruptions. Given that agreement needs to be reached for each block, such protocols will produce blocks at a rate substantially slower than a PoS blockchain (where the slow down matches the expected length of the execution of the Byzantine agreement protocol); on the other hand, Algorand can be parameterised to be free of forks and then blocks will "settle" immediately after the conclusion of each consensus sub-protocol. In particular full settlement will happen at expected constant number of rounds, which, asymptotically, is superior to any blockchain protocol (it is worth noting that blockchain protocols somewhat mitigate this inherent slow-down by letting ledger users choose a different level of settlement security depending on the value of transactions that are at risk or by using an overlay protocol such as lightning [41]). This benefit comes at the expense of imposing specific participation bounds for

[3] Nakamoto's simplifications are pointed out in [28]: the analysis considers only the setting where a block withholding attacker acts without interaction as opposed to a more general attacker that, for instance, tries strategically to split the honest parties in more than one chains during the course of the double spending attack.

each sub-protocol instance which are unnecessary in the case of a blockchain protocol, see the follow up work on Ouroboros with dynamic availability [4] for further discussion on this topic. The Algorand approach also does not require an agreed concept of time between the participants and merely relies on the assumption that time passes with roughly with the same speed. While the protocol presented herein requires access to global time, in follow-up work, [5], it is shown how it is possible for the Ouroboros protocol to be enhanced with a time synchronization sub-protocol while retaining the core security argument we present here. Fruitchain [40] provides a reward mechanism and an approximate Nash equilibrium proof for a PoW-based blockchain. We use a similar reward mechanism at the blockchain level, nevertheless our underlying mechanics are different since we operate in a PoS setting. The core of the idea is to provide a PoS analogue of "endorsing" inputs in a *fair proportion* using the same logic as the PoW-based byzantine agreement protocol for honest majority from [28]. Finally, in the present paper we prove that the error bound for the common-prefix property (and thus persistence of transactions) has an error rate that drops sub-exponentially with the security parameter, while experimentally we demonstrate an exponential drop. Follow up work [42] shows how this gap can be closed by proving an exponentially decreasing error bound.

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 outlined in Sec. 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. We then present the protocol enhancement with anonymous channels in Sec. 7. Our incentive mechanism and the equilibrium argument are presented in Sec. 8. Following this, we discuss delegation in Sec. 9 and in Sec. 10, the resilience of the protocol under various particular attacks of interest. Finally, in Sec. 11 we discuss transaction confirmation times as well as general 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 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, with parameter $k \in \mathbb{N}$.** Once a node of the system proclaims a certain transaction tx as *stable*, the remaining nodes, if queried, will report tx at the same position of the ledger and agree on the entire prefix of the ledger (prior to tx). Stability is defined in terms of the blockchain: 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, with parameter $u \in \mathbb{N}$.** 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.

Persistence and liveness can be derived from the following three elementary properties [28, 30, 39] provided that protocol Π derives the ledger from a data structure in the form of a blockchain (or simply chain) that each party maintains locally and updates at the onset of each slot. The properties are as follows.

- **Common Prefix (CP); with parameter $k \in \mathbb{N}$.** The chains $\mathcal{C}_1, \mathcal{C}_2$ adopted by two honest parties at the onset of the slots $sl_1 \leq sl_2$ 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.
- **Honest Chain Growth (HCG); with parameters $\tau \in (0, 1]$ and $s \in \mathbb{N}$.** Consider the chain $\mathcal{C}$ adopted by an honest party. Let sl_2 be the slot associated with the last block of $\mathcal{C}$ and let sl_1 be a prior slot in which $\mathcal{C}$ has an honestly-generated block. If $sl_2 \geq sl_1 + s$, then the number of blocks appearing in $\mathcal{C}$ after sl_1 is at least τs. The parameter τ is called the speed coefficient.
- **Existential Chain Quality ($\exists$CQ); with parameter $s \in \mathbb{N}$.** Consider the chain $\mathcal{C}$ adopted by an honest party at the onset of a slot and any portion of $\mathcal{C}$ spanning s prior slots; then at least one honestly-generated block appears in this portion.

Some remarks are in place. This definition of common prefix reflects a strong variant of that appearing in [30]. As for chain quality, we work with this simple "existential" formulation for convenience. Previous work focused on a more general notion that bounds the *density* of honestly-generated blocks in a sufficiently long portion of a chain. In fact, one can establish such a density bound directly from existential chain quality along the lines of the proof of chain growth in Section 4.5.

As for chain growth, we focus on a version pertaining to the suffix of the chain following an honest block. In fact, as we note in Section 4.5, combining $\exists$CQ and HCG one can directly infer the following stronger version.

- **Chain Growth (CG); with parameters $\tau \in (0, 1]$ and $s \in \mathbb{N}$.** Consider the chain $\mathcal{C}$ adopted by an honest party at the onset of a slot and any portion of $\mathcal{C}$ spanning s prior slots; then the number of blocks appearing in this portion of the chain is at least τs.

As in the case of bitcoin, the *longest* chain plays a preferred role in our protocol; this provides a straightforward guarantee of honest chain growth.

Security Model. We adopt the model introduced by [28] for analyzing security of blockchain protocols enhanced with an ideal functionality $\mathcal{F}$. 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 $\mathcal{F}$. Similarly we denote by $\mathsf{EXEC}^{\mathcal{F}}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa)$ the output of $\mathcal{Z}$.

We note that multiple different "functionalities" will be encompassed by $\mathcal{F}$. Contrary to [28], our analysis is in the "standard model," and without a random oracle functionality. The first interfaces we incorporate in the ideal functionality used in the protocol are the "diffuse" and "key and transaction" functionality, denoted $\mathcal{F}_{\mathsf{D+KT}}$ and described below. Note that the diffuse functionality is also the mechanism via which we will obtain protocol synchronization.

Diffuse functionality. The diffuse functionality maintains an *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; one may think of this as a mailbox. Additionally, parties can instruct the functionality to *diffuse* a message, in which case the message will be appended to each party's incoming string. The functionality maintains rounds (slots) and all parties activated in a round are allowed to diffuse once. Rounds do not advance unless all activated parties have diffused a message. The adversary, when activated, may also interact with the functionality; it may 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). The current slot index may be requested at any time by any party. If a party does not fetch in a certain slot the messages written to its incoming string, they are flushed.

Key and Transaction functionality. The key registration functionality is initialized with n users, $U_1, \ldots, U_n$, and their respective stakes, $s_1, \ldots, s_n$; given such initialization, the functionality will consult with the adversary and will accept a (possibly empty) sequence of $(\mathsf{Corrupt}, U)$ messages and mark the corresponding users U as corrupt. For the corrupt users without a registered public-key the functionality will allow the adversary to set their public-keys. For honest users the functionality will sample public/secret-key pairs and record them based on a digital signature algorithm; we will consider two modes for key generation. In the ideal mode, key generation will be outsourced to an instance of an ideal signature functionality; in the real mode, an actual digital signature algorithm will be utilized; (cf. Figure 1 where we make these two variants explicit). Public-keys of corrupt users will be marked as such. Subsequently, any sequence of the following actions may take place: (i) A user may request to sign or verify a message whereupon the functionality will respond accordingly, restricting signing requests to only the owners of the respective keys. (ii) The entire directory of public-keys may be requested, whereupon the functionality will return it to the requesting user. (iii) A new user may be requested to be created by a message $(\mathsf{Create}, U, \mathcal{C})$ from the environment, in which case the functionality will follow the same procedure as above: 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; additionally, it will store $\mathcal{C}$ as the suggested initial state. The functionality will return the public-key to the environment upon successful completion of this interaction. (iv) An existing user may be requested to be corrupted by the adversary via a message $(\mathsf{Corrupt}, U)$. A user can only be corrupted after a delay of D slots; specifically, after a corruption request is registered, a timer is maintained and the corruption will take place after D slots have passed according to the round counter maintained in the Diffuse component of the functionality. Note that when running in real mode, a corruption

will have the effect of revealing the secret-key to the adversary. In any case, the adversary will be able to access the signing interface of any corrupted party.

We assume throughout that the execution of the protocol is with respect to a functionality $\mathcal{F}$ that incorporates, at least, the above two functionalities. Further functionalities are 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. The environment can give arbitrary permissions, however with respect to an initial stakeholder set and respective stake, the adversary will be restricted to controlling only a percentage of that stake, say α. In such case we refer to this adversary as an *α-initially-bounded* adversary. 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, at least, as the last entity in each sl_j (as well as during all adversarial party activations).

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 regarding the activation of the honest parties running the protocol.

- In each slot there will be at least one honest activated party.
- There will be a parameter $k \in \mathbb{Z}$ that will signify the maximum number of slots that an honest shareholder can be offline. In case an honest stakeholder is spawned after the beginning of the protocol via $(\mathsf{Create}, U, \mathcal{C})$ its initialization chain $\mathcal{C}$ provided by the environment should match an honest parties' chain which was active in the previous slot.
- The environment will distribute the transaction data $d \in \{0,1\}^*$ to all parties, emulating the effect of diffusing transactions in a peer-to-peer network. This transaction data, including the required signatures by each stakeholder, is obtained by the environment as specified in the protocol.
- In each slot sl_r, and for each active stakeholder U_j, the view of the stakeholder will include a set $\mathbb{S}_j(r)$ of public-keys and stake pairs of the form $(\mathsf{vk}_i, s_i) \in \{0,1\}^* \times \mathbb{N}$, for $j = 1, \ldots, n_r$; here n_r is the number of users introduced up to that slot according to the view of U_j. Public-keys will be marked as "corrupted" if the corresponding stakeholder has been corrupted (or marked for corrupted by a delayed adversary). We will say the adversary is $(1/2 - \delta)$-*bounded* in a particular execution for a parameter $\delta > 0$ if it holds that the total stake of the corrupted keys divided by the total stake $\sum_i s_i$ is less than $1/2 - \delta$ in all possible $\mathbb{S}_j(r)$. (Note the distinction between this notion—a property of the execution—and that of a $(1/2 - \delta)$-initially-bounded

adversary, which refers to the initial distribution appearing in the genesis block.) If the above is violated, we say the event $\mathsf{Bad}^{1/2-\delta}$ occurs for the given execution.

- Stake shift is bounded over short periods. Specifically, for any chain adopted by an honest party, the observed shift in stake distribution between "nearby" prefixes of the chain are bounded above. See Definition 5.2 for a precise definition.

We note that the availability assumption (restricting honest parties from long periods of disconnection) stated above is very conservative and our protocol can tolerate much longer offline times depending on the course of the execution; nevertheless, for the sake of simplicity, we use the above restriction. Finally, we note that in all of our proofs, whenever we say that a property Q holds with high probability over all executions, we will in fact argue that $Q \vee \mathsf{Bad}^{1/2-\delta}$ holds with high probability over all executions. This captures the fact that we exclude environments and adversaries that trigger $\mathsf{Bad}^{1/2-\delta}$ with non-negligible probability.

3 The Ouroboros Protocol: Overview of Design and Analysis

We first provide a general overview of the approach to design and analyze our protocol. The protocol's analysis will depend on a number of parameters: (i.) k is the number of blocks a certain message should have "on top of it" in order to be treated as part of the immutable history of the ledger, (ii.) ϵ, σ are parameters that bound adversarial stake and stake shift in the sense that the adversary is $(1/2-\epsilon)$-initially bounded as well as $(1/2-\epsilon-\sigma)$-bounded throughout the execution and moreover stake shifts no more than σ occur over short enough time periods, (iii.) D is the corruption delay that is imposed on the adversary, i.e., an honest stakeholder will be corrupted D slots after the corresponding "corrupt" message is generated 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 (see below for a discussion).

We present our protocol description in four *stages*, successively improving the adversarial model it can withstand. In all stages an "ideal functionality" $\mathcal{F}_{\mathsf{LS}}$ 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 $\mathcal{F}_{\mathsf{LS}}$).

Stage 1: Static stake; $D = L = R$. In the first stage, the trust assumption is static and remains with the initial set of stakeholders; the execution is not further divided into epochs. 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 on the environment, adversarial stake of no more than $1/2-\epsilon$ is assumed among those initial stakeholders. Specifically, the environment initially will allow the corruption of a number of stakeholders whose relative stake represents $1/2-\epsilon$. 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 effectively "static." $\mathcal{F}_{\mathsf{LS}}$ 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 $\mathsf{vk}_{i_1}, \ldots, \mathsf{vk}_{i_R}$ to form the committee that will possess honest majority with overwhelming probability in R, (this uses the fact that the relative stake possessed by malicious parties is $1/2-\epsilon$; a linear dependency of R to ϵ^{-2} will be imposed at this stage to guarantee applicability of strong concentration bounds). In more detail, the committee will be selected implicitly by appointing a stakeholder with probability proportional to its stake to each one of the $L = R$ 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 novel combinatorial argument that the probability of a k-common prefix violation drops exponentially in $\sqrt{k}$, cf. Theorem 4.24. Similar reasoning will yield favorable guarantees of chain growth and chain quality. An even more favorable analysis can be made against *covert* adversaries, i.e., adversaries that prefer to remain "under the radar" (cf. Section 6).

Stage 2: Dynamic stake with a beacon, adversarial look-ahead E, epoch period of R slots, and delay $D \approx 2R \ll L$. The central idea for the extension of the lifetime of the above protocol is to consider the sequential composition of several invocations. We first describe a protocol which relies on a trusted beacon that emits a uniformly random string at regular intervals. Specifically, the beacon, during slots $\{(j-1) \cdot R + 1, \ldots, jR\}$, reveals the j-th random string that seeds the leader election function for the following epoch. To simplify the relationship between this stage and the next, which adopts a secure multiparty computation for randomness generation, we expose the randomness beacon to the adversary E slots prior to exposure to honest parties; this is the "adversarial look-ahead" parameter. 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 the stake distribution adopted during the j-th epoch (with $j \geq 2$) is determined by the most recent block with time stamp less than $j \cdot R - k'$ for an appropriate parameter k'. (The generic parameter k' appearing here is given a precise formula in the description of the protocol.)

Regarding the evolving stake distribution, transactions will be continuously generated and transferred between stakeholders via the environment; players will incorporate posted transactions in the blockchain-based ledgers that they maintain. In order to accommodate the new accounts that are being created, the $\mathcal{F}_{\mathsf{LS}}$ 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 $\mathcal{F}_{\mathsf{LS}}$ 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 stakeholder accounts and 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 places $(1/2 + \epsilon + \sigma)$ stake majority with the honest players (note that different honest players might perceive a different stakeholder distribution in a certain slot). Furthermore, the stake distribution is guaranteed to shift by at most σ statistical distance over a certain number of slots—this permits honest players to obtain reliable estimates on past stake distributions by examining portions of their blockchains which may have adversarial blocks. 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 $(1/2 - \epsilon - \sigma)$ 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 $\ln(L/R)$ given that the lifetime of the system includes such a number of successive epochs). The corruption delay is set to $D \approx 2R$ (in fact, it can be set more precisely as a function of E and k'), thus enabling the adversary to perform adaptive corruptions as long as these are not instantaneous.

Stage 3: Dynamic state without a beacon, epoch period of R slots, and delay $D \approx 2R \ll L$. In the third stage, we remove dependence on the beacon by introducing a secure multiparty protocol with "guaranteed output delivery" that effectively simulates it. In this way, we can obtain

the long-livedness of the protocol as described in the stage 2 design but under the weaker assumptions of the stage 1 design, i.e., the mere availability of an initial random string and stakeholder distributions with honest majority. The core idea is the following: given that we guarantee that an honest majority among elected stakeholders will hold with very high probability, we can further use this elected set as participants in an instance of a (simple) secure multiparty computation (MPC) protocol. Naturally, 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 challenge is to demonstrate that the MPC suitably simulates a beacon with the relaxation that the output may become known to the adversary before it is known to the honest parties. A feature of this stage—from a cryptographic design perspective—is the use of the ledger itself for the simulation of a reliable broadcast that supports the MPC protocol.

Stage 4: Input endorsers, stakeholder delegates, anonymous communication. In the final stage of our design, we augment the protocol with two new roles for the entities that are maintaining the ledger and consider the benefits of anonymous communication. Input-endorsers create a second layer of transaction endorsing prior to block inclusion. This mechanism enables the protocol to withstand deviations such as selfish mining and enables us to show that honest behaviour is an approximate Nash equilibrium under reasonable assumptions regarding the costs of running the protocol. Note that input-endorsers are assigned to slots in the same way that slot leaders are, and inputs included in blocks are only acceptable if they are endorsed by an eligible input-endorser. Second, 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, we observe that by including an anonymous communication layer we can remove the corruption delay requirement that is imposed in our analysis. This is done at the expense of increasing the online time requirements for the honest parties.[4]

4 Analysis of the Static Stake Protocol

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 which will be the effective stake distribution throughout the execution. 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. Additionally, an adversary in control of a particular slot may freely generate multiple

[4] In follow-up work we show how the same can be achieved efficiently, see [22].

blocks associated with the slot, perhaps committing to distinct prior blockchains, and strategically advertise them to honest players. In contrast, adversaries in typical PoW settings are constrained to make decisions in an online fashion and cannot freely generate multiple blocks. 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 prescribed signature scheme is generated; we assume without loss of generality that the verification keys $\mathsf{vk}_1, \ldots$ are known by all stakeholders. Before describing the protocol, we establish basic definitions following the notation of [28].

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

Definition 4.3 (Block)**.** *A* block B *generated at a slot* $sl \in \{1, \ldots, R\}$ *contains the current state* $st \in \{0,1\}^\lambda$*, data* $d \in \{0,1\}^*$*, the slot number* sl *and a signature* $\sigma = \mathsf{Sign}_{\mathsf{sk}}(st, d, sl)$ *computed under* sk *corresponding to the stakeholder* U *generating the block at that slot.*

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* $\mathrm{len}(\mathcal{C}) = n$ *is its number of blocks. The block* B_n *is the* head *of the chain, denoted* $\mathrm{head}(\mathcal{C})$*. We treat the empty string* ε *as a legal chain and by convention set* $\mathrm{head}(\varepsilon) = \varepsilon$*.*

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 $k \geq \mathrm{len}(\mathcal{C})$ 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 $\mathcal{C}_2$. We let $\mathcal{C}[k] = B_k$ and adopt interval notation to reflect contiguous portions of a chain: specifically, $\mathcal{C}[k:\ell] = B_k, \ldots, B_\ell$ and parentheses are used to indicate removal of endpoint so that, for example, $\mathcal{C}[k:\ell) = B_k, \ldots, B_{\ell-1}$.

Definition 4.5 (Epoch)**.** *An* epoch *is a set of* R *adjacent slots* $S = \{1, \ldots, R\}$*.*

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

Definition 4.6 (Adversarial Stake Ratio)**.** *Let* $U_{\mathcal{A}}$ *be the set of stakeholders controlled by an adversary* $\mathcal{A}$*. Then the* adversarial stake ratio *is defined as*

$$\alpha = \frac{\sum_{j \in U_{\mathcal{A}}} s_j}{\sum_{i=1}^{n} s_i},$$

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

Reflecting corruption delay. *In general, we consider adversaries subject to a* corruption delay*, which imparts a delay between the time when the adversary selects a party for corruption and the time when the party actually passes into adversarial control. With such an adversary, parties which have been identified for corruption but are not yet under adversarial control are counted in the adversarial stake ratio.*

Unambiguous stake distributions. *In principle, it is possible for various honest parties to disagree on the current (or a previous) stake distribution. We avoid these ambiguities by explicitly identifying a stake distribution when considering adversarial stake ratio.*

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 $\mathcal{F}_{\mathsf{LS}}$, defined in Figure 1. This functionality is parameterized by the set of initial stakeholders $\{(U_1, s_1), \ldots, (U_n, s_n)\}$ 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 *with respect to stakeholder distribution* $\mathbb{S} = \{(\mathsf{vk}_1, s_1), \ldots, (\mathsf{vk}_n, s_n)\}$, $(\mathcal{D}, \mathsf{F})$ *is a pair consisting of a distribution and a deterministic function such that, when* $\rho \leftarrow \mathcal{D}$ *it holds that for all* $sl_j \in \{sl_1, \ldots, sl_R\}$, $\mathsf{F}(\mathbb{S}, \rho, sl_j)$ *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 "weighting by stake"); furthermore the family of random variables* $\{\mathsf{F}(\mathbb{S}, \rho, sl_j)\}_{j=1}^{R}$ *are independent.*

We note that sampling proportional to stake can be implemented in a straightforward manner. For instance, a simple process operates as follows. Let $\tilde{p}_i = s_i / \sum_{j=i}^{n} s_j$. For each $i = 1, \ldots, n-1$, provided that no stakeholder has yet been selected, the process flips a $\tilde{p}_i$-biased coin; if the result of the coin is 1, the party U_i is selected for the slot and the process is complete. (Note that $\tilde{p}_n = 1$, so the process is certain to complete with a unique leader.) When we implement this process as a function $F(\cdot)$, sufficient randomness must 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 $\mathcal{F}_{\mathsf{LS}}[\mathsf{mode}]$

$\mathcal{F}_{\mathsf{LS}}[\mathsf{mode}]$ incorporates the diffuse and key/transaction functionality $\mathcal{F}_{\mathsf{D+KT}}$ from Section 2 and is parameterized by the respective stakes of the initial stakeholders $\mathbb{S}_0 = \{(U_1, s_1), \ldots, (U_n, s_n)\}$, a distribution $\mathcal{D}$ and a function F so that $(\mathcal{D}, \mathsf{F})$ is a leader selection process. In addition, $\mathcal{F}_{\mathsf{LS}}[\mathsf{mode}]$ is parameterized by mode, which determines how signature verification keys are generated. When $\mathcal{F}_{\mathsf{LS}}[\mathsf{mode}]$ is instantiated with $\mathsf{mode} = \mathsf{SIG}$ (resp. $\mathsf{mode} = \mathcal{F}_{\mathsf{DSIG}}$) it is denoted $\mathcal{F}_{\mathsf{LS}}[\mathsf{SIG}]$ (resp. $\mathcal{F}_{\mathsf{LS}}[\mathcal{F}_{\mathsf{DSIG}}]$). $\mathcal{F}_{\mathsf{LS}}$ interacts with stakeholders as follows:

- **Signature Key Pair Generation:** $\mathcal{F}_{\mathsf{LS}}[\mathsf{SIG}]$ generates signing and verification keys $\mathsf{sk}_i, \mathsf{vk}_i$ for stakeholder U_i by executing $\mathsf{KG}(1^\kappa)$ for $i = 1, \ldots, n$. $\mathcal{F}_{\mathsf{LS}}[\mathcal{F}_{\mathsf{DSIG}}]$ generates $(\mathsf{sk}_i, \mathsf{vk}_i)$ by querying $\mathcal{F}_{\mathsf{DSIG}}$ (Figure 3) with (KeyGen, sid_i) on behalf of U_i (with a unique session identifier sid_i related to U_i) and setting $(\mathsf{sk}_i = sid_i, \mathsf{vk}_i = v_i)$ (received from $\mathcal{F}_{\mathsf{DSIG}}$ as response) for $i = 1, \ldots, n$. In either case, $\mathcal{F}_{\mathsf{LS}}[\mathsf{mode}]$ sets $\mathbb{S}_0' = \{(\mathsf{vk}_1, s_1), \ldots, (\mathsf{vk}_n, s_n)\}$.
- **Genesis Block Generation** Upon receiving $(\mathsf{genblock\_req}, U_i)$ from stakeholder U_i, $\mathcal{F}_{\mathsf{LS}}$ proceeds as follows. If ρ has not been set, $\mathcal{F}_{\mathsf{LS}}$ samples $\rho \leftarrow \mathcal{D}$. In any case, $\mathcal{F}_{\mathsf{LS}}$ sends $(\mathsf{genblock}, \mathbb{S}_0', \rho, \mathsf{F})$ to U_i.
- **Signatures and Verification**. For signing and verification requests on behalf of user U_i, $\mathcal{F}_{\mathsf{LS}}[\mathcal{F}_{\mathsf{DSIG}}]$ provides access to the corresponding $\mathcal{F}_{\mathsf{DSIG}}$ interface, while $\mathcal{F}_{\mathsf{LS}}[\mathsf{SIG}]$ utilizes $(\mathsf{sk}_i, \mathsf{vk}_i)$ to respond restricting access of signing to the respective secret-key owners.

Figure 1: Functionality $\mathcal{F}_{\mathsf{LS}}[\mathsf{mode}]$.

A Protocol in the $\mathcal{F}_{\mathsf{LS}}[\mathsf{mode}]$-hybrid model. We start by describing a simple PoS based blockchain protocol considering static stake in the $\mathcal{F}_{\mathsf{LS}}[\mathsf{SIG}]$-hybrid model, i.e., where the genesis block B_0 (and consequently the slot leaders) are determined by the ideal functionality $\mathcal{F}_{\mathsf{LS}}[\mathsf{SIG}]$. $\mathcal{F}_{\mathsf{LS}}[\mathsf{SIG}]$ provides the stakeholders with a genesis block containing a stake distribution indexed by signature verification keys generated by a $\mathsf{EUF\text{-}CMA}$ signature scheme, while $\mathcal{F}_{\mathsf{LS}}[\mathcal{F}_{\mathsf{DSIG}}]$ obtains such keys from a signature ideal functionality $\mathcal{F}_{\mathsf{DSIG}}$. This subtle difference comes into play when describing an ideal version of π_{SPoS} used in an intermediate hybrid argument of the security proof, which will be discussed in Section 4.2. The stakeholders $U_1, \ldots, U_n$ interact among themselves and with $\mathcal{F}_{\mathsf{LS}}$ through Protocol π_{SPoS} described in Figure 2.

We are interested in applications where transactions are inserted in the ledger. In our analysis, we will consider simple coin transfer transactions of the format "stakeholder vk_1 transfers to stakeholders vk_2 an amount of x coins." A transaction will consist of a transaction template tx of this format accompanied by a signature of tx under the signing key corresponding to vk_1. We define a valid transaction as follows:

Definition 4.8 (Valid Transaction)**.** *A pair (tx, σ) is considered a valid transaction by a verifier V if the following holds:*

- *The transaction template tx is of the format "stakeholder vk_1 transfers to stakeholder vk_2 an amount of x coins with transaction serial number sn" where vk_1 is a verification key contained in the current stake distribution $\mathbb{S}$ and $x \in \mathbb{Z}$.*
- $\mathsf{Vrf}_{\mathsf{vk}_1}(\sigma, tx) = 1$.
- *The ledger cannot contain two transactions issued from the same stakeholder with the same serial number sn; thus a transaction is only valid with respect to a blockchain if no previous transaction (from the same stakeholder) has the same serial number.*

For simplicity we assume that all properly signed transactions are valid and are included in the ledger; in particular this means that there is a way to parse the ledger and disambiguate any recorded double/overspents (e.g., following the order that is imposed by the ledger).

Given Definitions 4.4 and 4.8, we define a *valid chain* as a blockchain (according to Definition 4.4) where all transactions contained in every block are valid (according to Definition 4.8). 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 $\mathsf{maxvalid}(\mathcal{C}, \mathbb{C})$: 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.

4.2 Transition to the Ideal Protocol

As a first step of the security analysis of π_{SPoS}, we will introduce an *idealized protocol* π_{iSPoS} and present an intermediate hybrid argument that shows that it is computationally indistinguishable from π_{SPoS}. Instead of relying on $\mathcal{F}_{\mathsf{LS}}[\mathsf{SIG}]$ and an $\mathsf{EUF\text{-}CMA}$ signature scheme, π_{iSPoS} operates with an ideal signature scheme. To that end, π_{iSPoS} interacts with $\mathcal{F}_{\mathsf{LS}}[\mathcal{F}_{\mathsf{DSIG}}]$ for obtaining signing

Protocol π_{SPoS}

π_{SPoS} is a protocol run by stakeholders $U_1, \ldots, U_n$ interacting with $\mathcal{F}_{\mathsf{LS}}[\mathsf{SIG}]$ over a sequence of slots $S = \{1, \ldots, R\}$. π_{SPoS} proceeds as follows:

1. **Initialization** Stakeholder $U_i \in \{U_1, \ldots, U_n\}$, receives from the key registration interface its public and secret key. Then it receives the current slot from the diffuse interface and in case it is sl_1 it sends $(\mathsf{genblock\_req}, U_i)$ to $\mathcal{F}_{\mathsf{LS}}[\mathsf{SIG}]$, receiving $(\mathsf{genblock}, \mathbb{S}_0, \rho, \mathsf{F})$ as answer. U_i sets the local blockchain $\mathcal{C} = B_0 = (\mathbb{S}_0, \rho)$ and the initial internal state $st = H(B_0)$. Otherwise, it receives from the key registration interface the initial chain $\mathcal{C}$, sets the local blockchain to $\mathcal{C}$ and the initial internal state $st = H(\mathrm{head}(\mathcal{C}))$.
2. **Chain Extension** For every slot $sl_j \in S$, every stakeholder U_i performs the following steps:
   (a) U_i receives from the environment the transaction data $d \in \{0,1\}^*$ to be inserted into the blockchain.
   (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 $U' = \mathsf{F}(\mathbb{S}_0, \rho, sl')$. U_i computes $\mathcal{C}' = \mathsf{maxvalid}(\mathcal{C}, \mathbb{C})$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\mathrm{head}(\mathcal{C}'))$.
   (c) If U_i is the slot leader determined by $\mathsf{F}(\mathbb{S}_0, \rho, sl_j)$, 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 computes $\mathcal{C}' = \mathcal{C}|B$, broadcasts $\mathcal{C}'$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\mathrm{head}(\mathcal{C}'))$.
3. **Transaction generation** Upon receiving a transaction template tx from the environment, U_i computes $\sigma = \mathsf{Sign}_{\mathsf{sk}_i}(tx)$ provided that tx is consistent with the state of the ledger in the view of U_i) and sends (tx, σ) to the environment.

Figure 2: Protocol π_{SPoS}.

and verification keys for the ideal signature scheme employed in the protocol. In the next sections, we will prove that π_{iSPoS} is secure through a series of combinatorial arguments. This hybrid approach insulates these combinatorial arguments from the specific details of the underlying signature schemes used to instantiate π_{SPoS} and the biases that these schemes might introduce in the distributions of π_{SPoS}.

First, in Figure 3, we present Functionality $\mathcal{F}_{\mathsf{DSIG}}$ as defined in [16], where it is also shown that EUF-CMA signature schemes realize $\mathcal{F}_{\mathsf{DSIG}}$. Notice that this fact will be used to show that our idealized protocol can actually be realized based on practical digital signature schemes (such as DSA and ECDSA) and ultimately that π_{iSPoS} is indistinguishable from π_{SPoS}.

The idealized protocol π_{iSPoS} is run by the stakeholders interacting with $\mathcal{F}_{\mathsf{LS}}[\mathcal{F}_{\mathsf{DSIG}}]$ and $\mathcal{F}_{\mathsf{DSIG}}$. Basically, π_{iSPoS} behaves exactly as π_{SPoS} except for calls to $\mathsf{Vrf}_{\mathsf{vk}}(\sigma)$ and $\mathsf{Sign}_{\mathsf{sk}}(m)$. Namely, instead of locally computing $\mathsf{Sign}_{\mathsf{sk}_i}(m)$, U_i sends (Sign, sid, m) to $\mathcal{F}_{\mathsf{DSIG}}$, receiving $(\mathsf{Signature}, sid, m, \sigma)$ and outputting σ as the signature. Moreover, instead locally computing $\mathsf{Vrf}_{\mathsf{vk}'}(\sigma, m)$, U_i sends $(\mathsf{Verify}, sid_i, m, \sigma, v')$ to $\mathcal{F}_{\mathsf{DSIG}}$ (where v' corresponds to verification key vk'), outputting the value f received in message $(\mathsf{Verified}, sid_i, m, f)$. Protocol π_{iSPoS} is described in Figure 4. This idealized description will be further developed when arguing about the dynamic stake case, where additional building blocks must be considered in the idealized protocol.

The following proposition is an immediate corollary of the results in [16] showing that EUF-CMA signature schemes realize $\mathcal{F}_{\mathsf{DSIG}}$.

Functionality $\mathcal{F}_{\mathsf{DSIG}}$

$\mathcal{F}_{\mathsf{DSIG}}$ interacts with stakeholders as follows:

- **Key Generation** Upon receiving a message (KeyGen, sid) from a stakeholder U_i, verify that $sid = (U_i, sid')$ for some sid'. If not, then ignore the request. Else, hand (KeyGen, sid) to the adversary. Upon receiving $(\mathsf{VerificationKey}, sid, v)$ from the adversary, output $(\mathsf{VerificationKey}, sid, v)$ to U_i, and record the pair (U_i, v).
- **Signature Generation** Upon receiving a message (Sign, sid, m) from U_i, verify that $sid = (U_i, sid')$ for some sid'. If not, then ignore the request. Else, send (Sign, sid, m) to the adversary. Upon receiving $(\mathsf{Signature}, sid, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to U_i and halt. Else, output $(\mathsf{Signature}, sid, m, \sigma)$ to U_i, and record the entry $(m, \sigma, v, 1)$.
- **Signature Verification** Upon receiving a message $(\mathsf{Verify}, sid, m, \sigma, v')$ from some stakeholder $U_{i'}$, hand $(\mathsf{Verify}, sid, m, \sigma, v')$ to the adversary. Upon receiving $(\mathsf{Verified}, sid, m, \phi)$ from the adversary do:
  1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds.)
  2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, v, 0)$. (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
  3. Else, if there is an entry (m, σ, v', f') recorded, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
  4. Else, let $f = \phi$ and record the entry (m, σ, v', ϕ).

  Output $(\mathsf{Verified}, sid, m, f)$ to $U_{i'}$.

Figure 3: Functionality $\mathcal{F}_{\mathsf{DSIG}}$.

Proposition 4.9. *For each PPT $\mathcal{A}, \mathcal{Z}$ it holds that there is a PPT $\mathcal{S}$ so that*

$$\mathsf{EXEC}^{\mathcal{F}_{\mathsf{LS}}[\mathsf{SIG}]}_{\pi_{\mathrm{SPoS}}, \mathcal{A}, \mathcal{Z}}(\lambda) \quad \textit{and} \quad \mathsf{EXEC}^{\mathcal{F}_{\mathsf{LS}}[\mathcal{F}_{\mathsf{DSIG}}]}_{\pi_{\mathrm{iSPoS}}, \mathcal{S}, \mathcal{Z}}(\lambda)$$

are computationally indistinguishable.

In light of the above proposition in the remaining of the analysis we will focus on the properties of the protocol π_{iSPoS} (note that this implication does not apply to any[5] possible property one might consider in an execution for π_{iSPoS}; nevertheless the properties we will prove for π_{iSPoS} are all verifiable by the environment $\mathcal{Z}$ and as a result they can be inherited by π_{SPoS} due to proposition 4.9).

4.3 The Fork Abstraction

In our security arguments we routinely use elements of $\{0,1\}^n$ to indicate which slots—among a particular window of slots of length n—have been assigned to adversarial stakeholders. When strings have this interpretation we refer to them as *characteristic strings.*

Definition 4.10 (Characteristic String)**.** *Fix an execution $\mathcal{E}$ with genesis block B_0, adversary $\mathcal{A}$, and environment $\mathcal{Z}$. Let $S = \{i+1, \ldots, i+n\}$ denote a sequence of slots of length $|S| = n$. The*

[5] An example of such a property would be a property testing a non-trivial fact about the parties' private states.

Protocol π_{iSPoS}

π_{iSPoS} is a protocol run by stakeholders $U_1, \ldots, U_n$ interacting with $\mathcal{F}_{\mathsf{LS}}[\mathcal{F}_{\mathsf{DSIG}}]$ over a sequence of slots $S = \{1, \ldots, R\}$. π_{iSPoS} proceeds as follows:

1. **Initialization** Stakeholder $U_i \in \{U_1, \ldots, U_n\}$, receives from the key registration interface its public and secret key. Then it receives the current slot from the diffuse interface and in case it is sl_1 it sends $(\mathsf{genblock\_req}, U_i)$ to $\mathcal{F}_{\mathsf{LS}}[\mathcal{F}_{\mathsf{DSIG}}]$, receiving $(\mathsf{genblock}, \mathbb{S}_0, \rho, \mathsf{F})$ as answer. U_i sets the local blockchain $\mathcal{C} = B_0 = (\mathbb{S}_0, \rho)$ and the initial internal state $st = H(B_0)$. Otherwise, it receives from the key registration interface the initial chain $\mathcal{C}$, sets the local blockchain to $\mathcal{C}$ and the initial internal state $st = H(\text{head}(\mathcal{C}))$.
2. **Chain Extension** For every slot $sl_j \in S$, every 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 $\mathcal{F}_{\mathsf{DSIG}}$ answers with $(\mathsf{Verified}, sid, (st', d', sl'), 1)$ upon being queried with $(\mathsf{Verify}, sid, (st', d', sl'), \sigma', \mathsf{vk}')$, where vk' is the verification key of the stakeholder $U' = \mathsf{F}(\mathbb{S}_0, \rho, sl')$. U_i computes $\mathcal{C}' = \mathsf{maxvalid}(\mathcal{C}, \mathbb{C})$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\text{head}(\mathcal{C}'))$.
   (b) If U_i is the slot leader determined by $\mathsf{F}(\mathbb{S}_0, \rho, sl_j)$, 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 σ is obtained from $\mathcal{F}_{\mathsf{DSIG}}$'s answer $(\mathsf{Signature}, sid, (st, d, sl_j), \sigma)$ upon being queried with $(\mathsf{Sign}, sid_i, (st, d, sl_j))$. U_i computes $\mathcal{C}' = \mathcal{C}|B$, broadcasts $\mathcal{C}'$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\text{head}(\mathcal{C}'))$.
3. **Transaction generation** Given a transaction template tx, U_i returns σ obtained from $\mathcal{F}_{\mathsf{DSIG}}$'s answer $(\mathsf{Signature}, sid_i, tx, \sigma)$ upon being queried with $(\mathsf{Sign}, sid_i, tx)$, provided that tx is consistent with the state of the ledger in the view of U_i.

Figure 4: Protocol π_{iSPoS}.

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

We start with some intuition for our approach to analyze the protocol. Let $w \in \{0,1\}^n$ be a characteristic string for a sequence of slots S. Among the fundamental properties we wish to ensure of our protocol (CP, $\exists\mathsf{CQ}$, and HCG), common prefix (CP) will require the most technical effort. In this section, we develop a graph-theoretic abstraction to facilitate reasoning about these properties, principally motivated by the task of establishing CP.

To motivate this, consider two observers that (i.) go offline immediately prior to the commencement of a sequence of slots S, (ii.) have adopted the same current chain $\mathcal{C}_0$ 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 exactly $\mathcal{C}_0$. 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 $\mathcal{C}_0$. The bulk of the proof that the Ouroboros protocol achieves CP relies on the fact that characteristic strings permitting 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 the protocol at a more abstract level, we define below a formal notion of "fork" that captures the relationship between the chains adopted by *honest* slot leaders during an execution of the protocol π_{iSPoS}. 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 B previously broadcast by an honest player, the prefix of $\mathcal{C}$ prior to B must entirely agree with the chain (terminating at B) broadcast by this previous honest player. This "confluence" property follows immediately from the fact that the state of any honest block effectively commits to a unique chain beginning at the genesis block. To conclude, any chain $\mathcal{C}$ diffused by an honest player must begin with a chain produced by a previously honest player (or, alternatively, the genesis block), continue with a possibly empty sequence of adversarial blocks and, finally, terminate with an honest block. 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 the various honest blocks added by honest players during the protocol must all be distinct.

Of course, the actual chains induced by an execution of π_{iSPoS} are comprised of blocks containing a variety of data that are immaterial for reasoning about forking or the other elementary chain properties. For this reason the formal notion of fork below merely reflects the directed tree formed by the relevant chains and the *identities of the players*—expressed as indices in the string w—responsible for generating the blocks in these chains.

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.

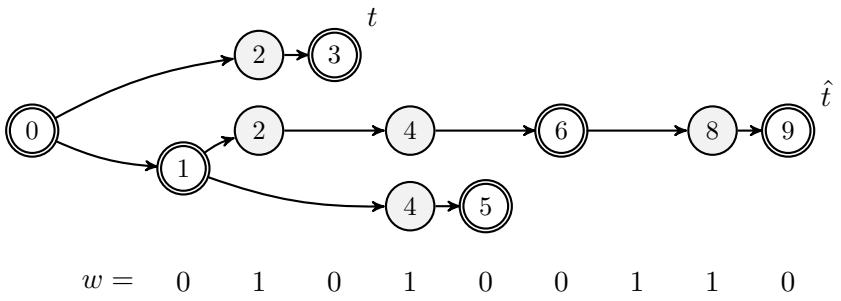


Figure 5: A fork F for the string $w = 010100110$; vertices appear with their labels and honest vertices are highlighted with double borders. Note that the depths of the (honest) vertices associated with the honest indices of w are strictly increasing. Two tines are distinguished in the figure: one, 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 $\text{gap}(t)$ indicates the difference in length between t and $\hat{t}$; in this case $\text{gap}(t) = 4$. The quantity $\text{reserve}(t) = |\{i \mid \ell(v) < i \leq |w| \text{ and } w_i = 1\}|$ indicates the number of adversarial indices appearing after the label of the last honest vertex v of the tine; in this case $\text{reserve}(t) = 3$. As each leaf of F is honest, F is closed.

Definition 4.11 (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;*
- *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. We say that a fork is trivial *if it contains a single vertex, the root.*

Definition 4.12 (Tines, depth, and height)**.** *A path in a fork F originating at the root is called a* tine*. For a tine t we let* length(t) *denote its* length*, equal to the number of edges on the path. For a vertex v, we let* depth(v) *denote the length of the (unique) tine terminating at v. The* height *of a fork (as usual for a tree) is defined to be the length of the longest tine.*

We overload the notation $\ell()$ so that it applies to tines, by defining $\ell(t) \triangleq \ell(v)$, where v is the terminal vertex on the tine t. We borrow the "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 length$(t) \leq k$*, we define $t^{\lceil k}$ to consist solely of the root.*

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*. For convenience, we declare the root vertex to be honest. We extend this terminology to tines: a tine is *honest* if it terminates with an honest vertex and *adversarial* otherwise. By this convention the empty tine t_ϵ is honest.

The fork of honestly constructed chains. As discussed above, with an execution $\mathcal{E}$ of π_{iSPoS} one can naturally associate a characteristic string w and a fork $F_D \vdash w$ corresponding to the chains *constructed and diffused by honest participants* during the protocol. See Figure 5 for an example, which also demonstrates some of the quantities defined above and in the remainder of this section. The fork shown in the figure reflects an execution in which *(i.)* the honest player associated with the first slot builds directly on the genesis block (as it must), *(ii.)* the honest player associated with the third slot is shown a chain of length 1 produced by the adversarial player of slot 2 (in addition to the honestly generated chain of step *(i.)*), which it elects to extend, *(iii.)* the honest player associated with slot 5 is shown a chain of length 2 building on the chain of step *(i.)* augmented with a further adversarial block produced by the player of slot 4, etc.

We remark that the tight correspondence described above between forks and executions requires that a slot is marked as adversarial if the owner of that slot *ever fell under adversarial control* during the protocol; this is one direct indication of the challenge of "long-range attacks" which involve corruption of slot leaders long after they have been awarded leadership. In our long-lived protocols, such attacks are mitigated by a bounded-depth longest-chain rule which permits analysis of forks whose characteristic strings are determined by the corruption schedule of an adversary over a bounded period of time.

We begin by defining a natural notion of inclusion for two forks:

Definition 4.13 (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 4.14 (Closed forks)**.** *A fork is* closed *if each leaf is honest. By convention the trivial fork, consisting solely of a root vertex, is closed.*

Note that the fork F_D discussed above of corresponding to honestly created chains is closed: Every chain constructed by an honest player naturally terminates with block broadcast in an honest slot. We remark 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 $\check{w}$ is a prefix of w and $F \vdash w$, then there is a unique closed fork $\check{F} \vdash \check{w}$ for which $\check{F} \sqsubseteq F$. In particular, taking $\check{w} = w$, we note that for any fork $F \vdash w$, there is a unique closed fork $\overline{F} \vdash w$ for which $\overline{F} \sqsubseteq F$; in this case we say that $\overline{F}$ is the *closure* of F.

The forks of adopted chains. Consider now the valid chains *adopted* by the honest participants of the protocol. This set clearly includes all chains constructed (and diffused) by honest participants; on the other hand, it may contain additional valid chains delivered by the adversary to honest participants. In particular, we may associate a fork $F_A \vdash w$ with these adopted chains and note that $F_D \sqsubseteq F_A$. The fork F_A is not, in general, closed. Note, however, that maximal tines in this fork—as they have been adopted by an honest player according to the longest chain rule—must have length no less than any chain previously diffused by an honest player. We begin by precisely defining this notion.

Definition 4.15 (Viability)**.** *Let $F \vdash w$ be a fork for a string $w \in \{0,1\}^n$ and let t be a tine of F. We say that t is* viable *if, for all honest indices $h \leq \ell(t)$, we have*

$$\mathbf{d}(h) \leq \operatorname{length}(t)\,.$$

(Recall that $\ell(t)$ is the label of the terminal vertex of t.)

If t is viable, an honest participant (or observer) witnessing the execution at time $\ell(t)$—if provided the tine t along with all honest tines generated up to time $\ell(t)$—could conceivably select t via the $\mathsf{maxvalid}()$ rule. Observe that any honest tine is viable: by definition, the depth of the terminal vertex of an honest tine exceeds that of all prior honest vertices. As remarked above, all maximal tines appearing in F_A are viable.

4.3.1 The Abstract Chain Properties

Let $w \in \{0,1\}^n$ be a characteristic string. We define the following properties of w which are direct analogues of the general protocol properties defined above. Given a tine t and a sequence of slots S such that $\ell(t) \geq \max S$, we refer to a "portion of t spanning S" as the maximum subgraph t' of t such that $\ell(v) \in S \wedge v \in t \implies v \in t'$. We use the notation $t(S)$ to denote this subgraph. We continue to use interval notation for sequences of slots: that is, $[sl_1 : sl_2] = \{sl_1, \ldots, sl_2\}$ and parentheses in place of brackets indicate that the endpoint is left out. Thus $[sl_1 : sl_2) = \{sl_1, \ldots, sl_2 - 1\}$. As a final matter of notation, we often elide the parentheses in expressions such as $t([sl_1, sl_2])$, simply writing $t[sl_1, sl_2]$. We will refer to a "portion of t spanning s slots" when the particular sequence of slots is not specified.

- **Common Prefix (cp); with parameter $k \in \mathbb{N}$.** The string possesses k-cp if, for every fork $F \vdash w$ and every pair of viable tines t_1 and t_2 of F for which $\ell(t_1) \leq \ell(t_2)$, the tine $t_1^{\lceil k}$ is a prefix of t_2. (Equivalently, $\text{length}(t_1) - \text{length}(t_1 \cap t_2) \leq k$, where $t_1 \cap t_2$ denotes the common prefix of the two tines.)

- **Honest-Bounded Chain Growth (hcg); with parameters $\tau \in (0,1]$ and $s \in \mathbb{N}$.** A characteristic string w possesses hcg with parameters τ and s if, for every fork $F \vdash w$, every viable tine t of F, and every honest vertex v on t for which $\ell(v) + s \leq \ell(t)$, the path $t(\ell(v), \ell(t)]$ contains at least τs vertices.

- **Chain Growth (cg); with parameters $\tau \in (0,1]$ and $s \in \mathbb{N}$.** The string possesses (τ, s)-cg if, for every fork $F \vdash w$ and every viable tine t of F, any portion of t spanning s slots contains at least τs vertices.

- **Existential Chain Quality ($\exists$cq); with parameter $s \in \mathbb{N}$.** The string possesses s-$\exists$cq if, for every fork $F \vdash w$ and every viable tine t of F, any portion of t spanning s slots contains at least one honest vertex.

4.3.2 Probabilistic Preliminaries

Anticipating the proofs in the next few sections, we record a Chernoff–Hoeffding bound and an elementary stochastic dominance argument.

Theorem 4.16 (Chernoff–Hoeffding bound; see, e.g., [33])**.** *Let $X_1, \ldots, X_T$ be independent random variables with $X_i \in [0,1]$. Let $X = \sum_{i=1}^{T} X_i$ and $\mu = \mathbb{E}[X]$. Then, for all $\delta \geq 0$,*

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu} \qquad \text{and} \qquad \Pr[X \leq (1-\delta)\mu] \leq e^{-\frac{\delta^2}{2}\mu}\,.$$

Additionally, for any $\Lambda > 0$,

$$\Pr[X \geq \mu + \Lambda] \leq e^{-2\Lambda^2/T} \qquad \text{and} \qquad \Pr[X \leq \mu - \Lambda] \leq e^{-2\Lambda^2/T}\,.$$

Definition 4.17. *For two elements $x, y \in \{0,1\}^n$, we write $x \leq y$ if, for each i, $x_i \leq y_i$. With this partial order, we define a notion of monotone subsets of $\{0,1\}^n$: A subset $E \subseteq \{0,1\}^n$ is* monotone *if, for each pair $x, y \in \{0,1\}^n$, $x \in E$ and $x \leq y$ implies that $y \in E$. Let X and Y be two random variables taking values in $\{0,1\}^n$. We write $X \prec Y$ if $\Pr[X \in E] \leq \Pr[Y \in E]$ for any monotone set E. In this case, we say that Y stochastically dominates X.*

Lemma 4.18. *Let $X = (X_1, \ldots, X_n)$ be a family of random variables taking values in $\{0,1\}$ with the property that, for each $i > 0$, $\mathbb{E}[X_i \mid X_1, \ldots, X_{i-1}] \leq p$. Let $B = (B_1, \ldots, B_n)$ be a family of independent random variables, taking values in $\{0,1\}$, for which $\mathbb{E}[B_i = 1] = p$. Then $X \prec B$.*

Proof. We proceed by induction. The statement is clear for $n = 1$. In general, consider a random variable X satisfying the conditions of the theorem and taking values in $\{0,1\}^{n+1}$; let $E \subset \{0,1\}^{n+1}$ be a monotone event. We wish to prove that $\Pr[X \in E] \leq \Pr[B \in E]$.

We write $X = (Y, Z)$, where Y takes values in $\{0,1\}^n$ and Z in $\{0,1\}$. By induction, we may assume that $Y \prec (B_1, \ldots, B_n)$. Consider the events

$$E_0 = \{(y_1, \ldots, y_n) \mid (y_1, \ldots, y_n, 0) \in E\} \qquad \text{and} \qquad E_1 = \{(y_1, \ldots, y_n) \mid (y_1, \ldots, y_n, 1) \in E\}\,;$$

observe that the monotonicity of E implies that $E_0 \subseteq E_1$ and that E_0, E_1 are monotone. To study $\Pr[X \in E]$, for an element $y = (y_1, \ldots, y_n) \in \{0,1\}^n$ define

$$q(y) = \Pr[X \in E \mid Y = y]\,.$$

Observe that $\Pr[X \in E] = \mathbb{E}[q(Y)]$ and, recalling that $E_0 \subset E_1$, that

$$\begin{aligned} y \in E_0 &\Rightarrow q(y) = 1, \\ y \in E_1 \setminus E_0 &\Rightarrow q(y) \leq p \quad \text{by assumption, and} \\ y \notin E_1 &\Rightarrow q(y) = 0\,. \end{aligned}$$

We conclude that

$$\begin{aligned} \Pr[X \in E] &\leq \Pr[Y \in E_0] + p \cdot \Pr[Y \in E_1 \setminus E_0] = p \Pr[Y \in E_1] + (1-p)\Pr[Y \in E_0] \\ &\leq p \Pr[(B_1, \ldots, B_n) \in E_1] + (1-p)\Pr[(B_1, \ldots, B_n) \in E_0] \qquad (1) \\ &= \Pr[(B_1, \ldots, B_n) \in E_0] + p \Pr[(B_1, \ldots, B_n) \in E_1 \setminus E_0] = \Pr[B \in E]\,, \end{aligned}$$

as desired. The inequality of line (1) follows by the induction hypothesis. □

4.4 Chain Quality

Lemma 4.19 (Abstract Existential Chain Quality)**.** *Consider a characteristic string $w = w_1 \ldots w_L$ drawn in $\{0,1\}^L$ so that each w_i is independently assigned the value 1 with probability $1/2 - \epsilon$ for some $\epsilon \in (0, 1/2)$. Then, for $s > 0$,*

$$\Pr[w \textit{ does not satisfy } s\text{-}\exists\mathsf{cq}] \leq \epsilon_{\exists\mathsf{cq}}(s; L, \epsilon) \triangleq \left((\epsilon^{-2} + 3)/2\right) L \exp(-2\epsilon^2 s)\,.$$

Proof. An s-$\exists$cq violation for w consists of a fork $F \vdash w$ and a viable tine t for which there is a pair of indices sl_1 and sl_2 so that $sl_1 + s \leq sl_2 \leq \ell(t)$ and $t(sl_1 : sl_2]$ contains no honest vertices. For such a violation, let sl_1^* denote the largest index of an honest vertex in $t[0 : sl_1]$—note that the root of F is honest by fiat so that sl_1^* is well-defined. Similarly, let sl_2^* denote the smallest index for which $sl_2 \leq sl_2^* \leq \ell(t)$ and $t[0 : sl_2^*]$ is viable. This is also well defined since t is viable. Observe that no $sl_2 \leq sl^* \leq sl_2^*$ can index an honest vertex of t; otherwise, $sl^* > sl_2$ (by assumption) but the tine $t[0 : sl^* - 1]$ would be viable—as it supports extension by an honest vertex—which contradicts minimality of sl_2^*. As we assume that $s > 0$,

$$sl_1^* \leq sl_1 < sl_1 + s \leq sl_2 \leq sl_2^*$$

and observe that $t(sl_1^*, sl_2^*]$ contains no honest vertices. We define the *rank* of this violation to be the quantity $\ell = sl_2^* - sl_1^*$ and refer to sl_2^* as the *target* of the violation; we say that the pair $(sl_2^*; \ell)$ is the *signature* of such a violation. Note that the rank ℓ of a s-$\exists$cq violation is always at least s.

Consider now the sequence

$$sl_1^* = hsl_0 < \cdots < hsl_g \leq sl_2^*\,,$$

where $hsl_0, \ldots, hsl_g$ denote the honest indices in the region $[sl_1^*, sl_2^*]$, and let $t_0, \ldots, t_g$ denote the tines for which $\ell(t_i) = hsl_i$. Note that $t[0 : sl_1^*] = t_0$ and that $\text{length}(t_{i-1}) < \text{length}(t_i)$ for each $0 < i \leq g$. As $t[0 : sl_2^*]$ is viable and hsl_g is honest, we note that $\text{length}(t_g) \leq \text{length}(t[0 : sl_2^*])$. To conclude,

$$\text{length}(t[0 : sl_2^*]) \geq \text{length}(t[0 : sl_1^*]) + g\,.$$

We remark that $g = \#_0(\hat{w})$, where $\hat{w} = w_{sl_1^*+1}, \ldots, w_{sl_2^*}$, and note that $\ell = |\hat{w}|$. On the other hand, as $t(sl_1^*, sl_2^*]$ contains no honest vertex we have the immediate upper bound

$$\text{length}(t[0 : sl_2^*]) \leq \text{length}(t[0 : sl_1^*]) + \#_1(\hat{w})\,,$$

where $\#_1(\hat{w})$ denotes the number of adversarial indices in $\hat{w}$. Thus $\#_1(\hat{w}) \geq \#_0(\hat{w})$.

Fixing a particular signature (sl, ℓ) it follows that

$$\begin{aligned}
&\Pr[w \text{ admits an } s\text{-}\exists\mathsf{CQ} \text{ violation with signature } (sl;\ell)]\\
&\leq \Pr_w[\#_0(\hat{w}) - \#_1(\hat{w}) \leq 0]\\
&= \Pr_w[\#_1(\hat{w}) \geq \ell/2]\\
&\leq \Pr_w[\#_1(\hat{w}) \geq \ell(1/2 - \epsilon) + \epsilon\ell] \leq \exp(-2\epsilon^2\ell)
\end{aligned}$$

where $\hat{w}$, as above, denotes $w_{sl-\ell+1}, \ldots, w_{sl}$. It follows that for any $sl \leq L$

$$\begin{aligned}
&\Pr[w \text{ admits an } s\text{-}\exists\mathsf{CQ} \text{ violation with signature } (sl;\ell) \text{ for any } \ell \geq s]\\
&\leq \sum_{\ell=s}^{\infty} \exp(-2\epsilon^2\ell) \leq \int_{s-1}^{\infty} \exp(-2\epsilon^2\ell)\, d\ell = \frac{1}{2\epsilon^2}\exp(-2\epsilon^2(s-1))\\
&= \frac{\exp(2\epsilon^2)}{2\epsilon^2}\exp(-2\epsilon^2 s)\,.
\end{aligned}$$

The union bound, applied over all indices, yields

$$\Pr[w \text{ admits an } s\text{-}\exists\mathsf{CQ} \text{ violation over } y] \leq \frac{\exp(2\epsilon^2)}{2\epsilon^2} L \exp(-2\epsilon^2 s) \leq \left[(\epsilon^{-2}+3)/2\right] L \exp(-2\epsilon^2 s)\,,$$

where the final inequality follows from $\exp(x) \leq 1 + (3/2)x$ for $x \in (0, 1/2)$. □

4.5 Chain Growth

We begin with the lemma below demonstrating that chain growth (cg) follows from existential chain quality (∃cq) and honest-bounded chain growth (hcg).

Lemma 4.20. *Consider a characteristic string w that satisfies ∃cq with parameter $s_{\exists cq}$ and hcg with parameters τ_{hcg} and s_{hcg}; then it satisfies cg with parameters*

$$s = 2s_{\exists cq} + s_{hcg} \qquad \text{and} \qquad \tau = \tau_{hcg} \cdot \left(\frac{s_{hcg}}{s_{hcg} + 2s_{\exists cq}}\right).$$

In particular, assuming $s_{hcg} \geq 2s_{\exists cq}$, the execution satisfies cg with parameter $\tau \geq \tau_{hcg}/2$.

Proof. Let $F \vdash w$ be a fork and let t be a viable tine. Consider a portion of t spanning $\hat{s} \geq s = 2s_{\exists\mathsf{cq}} + s_{\mathsf{hcg}}$ slots. By ∃cq, there must be an honest vertex of t associated with the first $s_{\exists\mathsf{cq}}$ and last $s_{\exists\mathsf{cq}}$ slots. As these two honest blocks are separated by at least s_{hcg} slots, applying hcg to the tine that terminates at the later honest block (which is necessarily viable) guarantees that at least

$$\tau_{\mathsf{hcg}} \cdot (\hat{s} - 2s_{\exists\mathsf{CQ}}) = \tau_{\mathsf{hcg}} \cdot \underbrace{\left(\frac{\hat{s} - 2s_{\exists\mathsf{cq}}}{\hat{s}}\right)}_{(\dagger)} \hat{s} \geq \tau_{\mathsf{hcg}} \cdot \left(\frac{s_{\mathsf{hcg}}}{s_{\mathsf{hcg}} + 2s_{\exists\mathsf{cq}}}\right)\hat{s}$$

vertices appear in the region. (The last inequality follows because the function $f_\lambda(x) = (x - \lambda)/x$, for any $\lambda > 0$, is strictly increasing for $x > 0$—thus (†) is minimized when $\hat{s} = s_{\mathsf{HCG}} + 2s_{\exists\mathsf{CQ}}$.) The statement of the lemma follows. □

We now establish concrete bounds on hcg; coupled with the conclusions about ∃cq from Section 4.4, this will yield our desired bounds on cg.

Lemma 4.21 (Abstract Honest Chain Growth)**.** *Consider $w = w_1 \dots w_L$ drawn in $\{0,1\}^L$ so that each w_i independently takes the value 1 with probability $1/2 - \epsilon$. Then for any $s > 0$ and $\delta > 0$,*

$$\Pr[w \text{ admits a } (\tau, s)\text{-hcg violation}] \leq \epsilon_{hcg}(\tau, s; L, \epsilon) \triangleq \left((\delta^{-2} + 3)/2\right) L \exp(-2\delta^2 s)\,,$$

where $\tau = 1/2 + \epsilon - \delta$. In particular, taking $\delta = \epsilon$, we remark that

$$\Pr[w \text{ admits a } (1/2, s)\text{-hcg violation}] \leq (\epsilon^2/2 + 2) L \exp(-2\epsilon^2 s)\,.$$

Proof. In the event that w admits a (τ, s)-hcg violation there is a fork $F \vdash w$, a viable tine t of F, and two indices sl_1 and sl_2 for which (i.) t has an honest vertex v_1 associated with sl_1, (ii.) $sl_2 = \ell(t)$, (iii.) $sl_1 + s \leq sl_2$, and (iv.) $t(sl_1, sl_2]$ contains fewer than τs vertices. Let

$$sl_1 = hsl_0 < hsl_1 < \cdots < hsl_g \leq sl_2$$

denote the increasing sequence of all honest indices in the interval $\{sl_1, \dots, sl_2\}$ and, for each $0 \leq i \leq g$, let t_i denote the tine for which $\ell(t_i) = hsl_i$. Observe that $t[0 : sl_1] = t_0$ and, in case $g > 0$, $\text{length}(t_i) < \text{length}(t_{i+1})$ for $i \in \{0, \dots, g-1\}$. Observe that,

$$\text{length}(t) = \text{length}(t[0 : sl_2]) \overset{(*)}{\geq} \text{length}(t_g) \geq \text{length}(t_0) + g \geq \text{length}(t[0 : sl_1]) + g\,.$$

where $\overset{(*)}{\geq}$ follows from viability of t. Thus $\text{length}(t) \geq \text{length}(t[0 : sl_1]) + g$.

Note that g is the number of zeros appearing in the string $\check{w} = w_{sl_1+1}, \dots, w_{sl_2}$ and, in light of the discussion above $g < \tau s$. We say sl_2 is the *target* of this violation, and that $sl_2 - sl_1 = |\check{w}|$ is the *rank*.

Observe now that

$$\begin{aligned}
&\Pr[w \text{ admits a } (\tau, s)\text{-hcg violation with target } sl \text{ and rank } \ell] \\
&\leq \Pr[\#_0(\check{w}) < \tau s] = \Pr[\#_0(\check{w}) < [(1-\alpha) - \delta]\ell] \\
&\leq \Pr\left[\text{wt}(\check{w}) < \mathbb{E}[\text{wt}(\check{w})] - \ell\delta\right] \leq \exp(-2\delta^2 \ell)\,,
\end{aligned}$$

where the last inequality follows from the Chernoff-Hoeffding bounds (Theorem 4.16).

By the union bound, applied over ranks ℓ, we conclude that

$$\begin{aligned}
&\Pr[W \text{ admits a } (\tau, s)\text{-HCG violation with target } sl] \\
&\leq \sum_{\ell \geq s} \exp(-2\delta^2 \ell) \leq \int_{\ell = s-1}^{\infty} \exp(-2\delta^2 \ell)\, d\ell \\
&= \frac{1}{2\delta^2} \exp(-2\delta^2 (s-1)) \leq \frac{\exp(2\delta^2)}{2\delta^2} \exp(-2\delta^2 s)\,.
\end{aligned}$$

To complete the argument, a union bound over all targets yields the bound:

$$\Pr[w \text{ does not possess } (s, \tau)\text{-hcg}] \leq L \frac{\exp(2\delta^2)}{2\delta^2} \exp(-2\delta^2 s) \leq \left((\delta^{-2} + 3)/2\right) L \exp(-2\delta^2 s)\,,$$

where the final inequality follows from $\exp(x) \leq 1 + (3/2)x$ for $x \in (0, 1/2)$. □

Lemma 4.22 (Abstract Chain Growth)**.** *Consider $w = w_1, \ldots, w_L$ drawn in $\{0,1\}^L$ so that each w_i independently takes the value 1 with probability $1/2 - \epsilon$. Then for $s > 0$, we have*

$$\Pr[w \text{ does not satisfy } (1/4, s)\text{-}\mathsf{cg}] \leq \epsilon_{\mathsf{cg}}(1/4, s; L, \epsilon) \triangleq (\epsilon^{-2} + 4)L \exp(-\epsilon^2 s/2)\,.$$

Proof. The corollary follows directly by combining Lemmas 4.19, 4.20, and 4.21 with the parameters

$$s_{\exists\mathsf{cq}} = s/4, \qquad \text{and} \qquad s_{\mathsf{hcg}} = s/2, \quad \delta_{\mathsf{hcg}} = \epsilon\,.$$

Specifically, applying Lemma 4.19 with $s_{\exists\mathsf{cq}} = s/4$,

$$\Pr[w \text{ does not satisfy } s/4\text{-}\exists\mathsf{cq}] \leq (\epsilon^{-2}/2 + 2)L \exp(-\epsilon^2 s/2)\,.$$

Likewise, applying Lemma 4.21 with $s_{\mathsf{hcg}} = s/2$, $\delta_{\mathsf{hcg}} = \epsilon$, and $\tau = 1/2 + \epsilon - \delta_{\mathsf{hcg}} = 1/2$,

$$\Pr[w \text{ does not satisfy } (1/2, s/2)\text{-}\mathsf{hcg}] \leq (\epsilon^{-2}/2 + 2)L \exp(-\epsilon^2 s)\,.$$

In light of Lemma 4.20, and considering that $s_{\exists\mathsf{cq}} \leq s_{\mathsf{hcg}}/2$,

$$\begin{aligned}\Pr[w \text{ does not satisfy } (1/4, s)\text{-}\mathsf{cg}] &\leq (\epsilon^{-2}/2 + 2)L \exp(-\epsilon^2 s/2) + (\epsilon^{-2}/2 + 2)L \exp(-\epsilon^2 s) \\ &\leq (\epsilon^{-2} + 4)L \exp(-\epsilon^2 s/2)\,.\end{aligned}$$

□

4.6 Common Prefix

Definition 4.23 (Flat forks; the $\sim$ relation)**.** *For two tines t_1 and t_2 of a fork F, we write $t_1 \sim t_2$ if they share an edge. Note that $\sim$ is an equivalence relation on the set of nontrivial tines; on the other hand, if t_ϵ denotes the "empty" tine consisting solely of the root vertex then $t_\epsilon \not\sim t$ for any tine t. We say that a fork is* flat *if it has two tines $t_1 \not\sim t_2$ 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 $F \vdash w$.*

Note that in order for an execution of π_{iSPoS} to yield two entirely disjoint chains of maximum length (in the view of an observer), the characteristic string associated with the execution must be forkable. Our goal is to establish the following upper bound on the number of forkable strings. We then apply this to control the probability of a common prefix violation.

Theorem 4.24. *Let $\epsilon \in (0,1)$ and let w be a characteristic string drawn from $\{0,1\}^n$ by independently assigning each $w_i = 1$ with probability $(1-\epsilon)/2$. Then $\Pr[w \text{ is forkable}] = 2^{-\Omega(\sqrt{n})}$.*

Here the $\Omega()$ notation hides a constant that depends on ϵ. Note that in subsequent work, Russell et al. [42] improved this bound to $2^{-\Omega(n)}$.

Structural features of forks: Reach, gap, reserve, and margin.

Definition 4.25 (Gap, reserve and reach)**.** *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 $\mathrm{gap}(t)$, 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 last 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. Finally, for a tine t we define

$$\mathrm{reach}(t) = \mathrm{reserve}(t) - \mathrm{gap}(t)\,.$$

Definition 4.26 (Margin)**.** *For a closed fork $F \vdash w$ we define $\rho(F)$ to be the maximum reach taken over all tines in F:*

$$\rho(F) = \max_t \operatorname{reach}(t)\,.$$

Likewise, we define the margin *of F, denoted $\mu(F)$, to be the "penultimate" reach taken over edge-disjoint tines of F: specifically,*

$$\operatorname{margin}(F) = \mu(F) = \max_{t_1 \not\sim t_2} \Big(\min\{\operatorname{reach}(t_1), \operatorname{reach}(t_2)\} \Big)\,. \tag{2}$$

We remark that the maxima above can always be obtained by honest tines. Specifically, if t is an adversarial tine of a fork $F \vdash w$, $\operatorname{reach}(t) \leq \operatorname{reach}(\bar{t})$, where $\bar{t}$ is the longest honest prefix of t.

As $\sim$ is an equivalence relation on the nonempty tines, it follows that there is always a pair of (edge-disjoint) tines t_1 and t_2 achieving the maximum in the defining equation (2) which satisfy $\operatorname{reach}(t_1) = \rho(F) \geq \operatorname{reach}(t_2) = \mu(F)$.

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

Proposition 4.27. *A string w is forkable if and only if there is a closed fork $F \vdash w$ for which* $\operatorname{margin}(F) \geq 0$*.*

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 $\hat{i}$ denote the largest honest index of w. Let F be a flat fork for w and let $\overline{F} \vdash w$ be the closure of F (obtained from F by removing any adversarial vertices from the ends of the tines of F). Note that the tine $\hat{t}$ containing $\hat{i}$ 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 $\hat{t}$. The prefixes of these two tines in $\overline{F}$ must clearly have reserve no less than gap (and hence non-negative reach); thus $\operatorname{margin}(\overline{F}) \geq 0$ as desired.

On the other hand, suppose w has a closed fork with $\operatorname{margin}(F) \geq 0$, in which case there are two edge-disjoint tines of F, t_1 and t_2, for which $\operatorname{reach}(t_i) \geq 0$. Then we can produce a flat fork by simply adding to each t_i a path of $\operatorname{gap}(t_i)$ vertices labeled with the subsequent adversarial indices promised by the definition of reserve(). □

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

$$\rho(w) = \max_{\substack{F \vdash w, \\ F \text{ closed}}} \rho(F)\,, \qquad \mu(w) = \max_{\substack{F \vdash w, \\ F \text{ closed}}} \mu(F)\,,$$

and, for convenience,

$$\mathbf{m}(w) = (\rho(w), \mu(w))\,.$$

Note that this overloads the notation $\rho(\cdot)$ and $\mu(\cdot)$ so that they apply to both forks and strings, but the setting will be clear from context. We remark that the definitions do not guarantee a priori that $\rho(w)$ and $\mu(w)$ can be achieved by the same fork, though this will be established in the lemma below. In any case, it is clear that $\rho(w) \geq 0$ and $\rho(w) \geq \mu(w)$ for all strings w; furthermore, by Proposition 4.27 a string w is forkable if and only if $\mu(w) \geq 0$. We refer to $\mu(w)$ as the *margin* of the string w.

In preparation for the proof of Theorem 4.24, we establish a recursive description for these quantities.

Lemma 4.28. $\mathbf{m}(\epsilon) = (0, 0)$ *and, for all nonempty strings* $w \in \{0,1\}^*$,

$$\mathbf{m}(w1) = (\rho(w) + 1, \mu(w) + 1)\,, \textit{ and}$$
$$\mathbf{m}(w0) = \begin{cases} (\rho(w) - 1, 0) & \textit{if } \rho(w) > \mu(w) = 0, \\ (0, \mu(w) - 1) & \textit{if } \rho(w) = 0, \\ (\rho(w) - 1, \mu(w) - 1) & \textit{otherwise.} \end{cases}$$

Furthermore, for every string w*, there is a closed fork* $F_w \vdash w$ *for which* $\mathbf{m}(w) = (\rho(F_w), \mu(F_w))$.

Proof. The proof proceeds by induction. If $w = \epsilon$, define F_ϵ to be the trivial fork; $F_\epsilon \vdash w$ is the unique closed fork for this string and $\mathbf{m}(\epsilon) = (0, 0) = (\rho(F_\epsilon), \mu(F_\epsilon))$, as desired.

In general, we consider $\mathbf{m}(w')$ for a string $w' = wx$—where $w \in \{0,1\}^*$ and $x \in \{0,1\}$; the argument recursively expands $\mathbf{m}(w')$ in terms of $\mathbf{m}(w)$ and the value of the last symbol x. In each case, we consider the relationship between two closed forks $F \sqsubseteq F'$ where $F \vdash w$ and $F' \vdash w' = wx$.

In the case where $x = 1$, we must have $F = F'$ as graphs, because the forks are assumed to be closed; it is easy to see that the reach of any tine t of $F \vdash w$ has increased by exactly one when viewed as a tine of $F' \vdash w'$. We write $\operatorname{reach}_{F'}(t) = \operatorname{reach}_F(t) + 1$, where we introduce the notation $\operatorname{reach}_\square()$ to denote the reach in a particular fork. It follows that $\rho(F') = \rho(F) + 1$ and $\mu(F') = \mu(F) + 1$. If $F^* \vdash w'$ is a closed fork for which $\rho(F^*) = \rho(w')$, note that F^* may be treated as a fork for w and, applying the argument above, we find that $\rho(w') \leq \rho(w) + 1$. A similar argument implies that $\mu(w') \leq \mu(w) + 1$. On the other hand, by induction there is a fork F_w for which $\mathbf{m}(w) = (\rho(F_w), \mu(F_w))$ and hence $\mathbf{m}(w') \geq (\rho(w) + 1, \mu(w) + 1)$. We conclude that

$$\mathbf{m}(w') = (\rho(w) + 1, \mu(w) + 1)\,. \tag{3}$$

Moreover, $\mathbf{m}(w') = (\rho(F_w), \mu(F_w))$, where F_w is treated as a fork for $w' = w1$.

The case when $x = 0$ is more delicate. As above, we consider the relationship between two closed forks $F \vdash w$ and $F' \vdash w' = w0$ for which $F \sqsubseteq F'$. Here F' is necessarily obtained from F by appending a path labeled with a string of the form $1^a 0$ 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, $\operatorname{gap}(t) \leq \operatorname{reserve}(t)$ (which is to say that $\operatorname{reach}(t) \geq 0$) and, in particular, $\operatorname{gap}(t) \leq a \leq \operatorname{reserve}(t)$: for the first inequality, note that the depth of the new honest vertex must exceed that of the deepest (honest) vertex in F and hence $a \geq \operatorname{gap}(t)$; as for the second inequality, there are only $\operatorname{reserve}(t)$ possible adversarial indices that may be added to t and hence $a \leq \operatorname{reserve}(t)$. We define the quantity $\tilde{a} \geq 0$ by the equation $a = \operatorname{gap}(t) + \tilde{a}$ and let t' denote the tine (of F') resulting by extending t in this way. We say that $\tilde{a}$ is the *parameter* for this pair of forks $F \sqsubseteq F'$.

Of course, every honest tine t of F is an honest tine of F' and it is clear that $\operatorname{reach}_{F'}(t) = \operatorname{reach}_F(t) - (\tilde{a} + 1)$, as the length of the longest tine t' in F' exceeds the length of the longest tine of F by exactly $\tilde{a} + 1$. Note that the reach of the new honest tine t' (in F') is always 0, as both $\operatorname{gap}(t')$ and $\operatorname{reserve}(t')$ are zero. It remains to describe how $\mu(w)$ and $\rho(w)$ are determined by this process.

The case $\rho(w) > \mu(w) = 0$**.** By induction, there is a fork F_w for which $\mathbf{m}(w) = (\rho(F_w), \mu(F_w))$. Let t_1 and t_2 be edge-disjoint tines of F_w for which $\rho(F_w) = \operatorname{reach}(t_1)$ and $\mu(F_w) = \operatorname{reach}(t_2)$. Define $F' \vdash w'$ to be the fork obtained by extending the tine t_2 of F_w with parameter $\tilde{a} = 0$ to yield a new tine t_2' in F'. Then $\operatorname{reach}_{F'}(t_1) = \rho(w) - 1$ and $\operatorname{reach}_{F'}(t_2') = 0$. It follows that

$\rho(w0) \geq \rho(w) - 1$ and $\mu(w0) \geq 0$. We will show that $\rho(w0) \leq \rho(w) - 1$ and that $\mu(w0) \leq 0$, in which case we can conclude that

$$\rho(w0) = \rho(w) - 1 \qquad \text{and} \qquad \mu(w0) = 0\,.$$

Moreover, the fork $F_{w'} = F'$ achieves these statistics, as desired.

We return to establish that $\rho(w0) \leq \rho(w) - 1$ and that $\mu(w0) \leq 0$. Let $F^* \vdash w0$ be a closed fork for which $\rho(w0) = \rho(F^*)$ and let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$; as above, let $\tilde{a}$ denote the parameter for this extension. Let t^* be an honest tine of F^* so that $\operatorname{reach}_{F^*}(t^*) = \rho(w0)$. If t^* is a tine of F, $\operatorname{reach}_{F^*}(t^*) = \operatorname{reach}_F(t^*) - (\tilde{a} + 1) \leq \rho(w) - 1$. Otherwise t^* was obtained by extension and $\operatorname{reach}_{F^*}(t^*) = 0 \leq \rho(w) - 1$ by assumption. In either case $\rho(w0) \leq \rho(w) - 1$, as desired. It remains to show that $\mu(w0) \leq 0$. Now consider $F^* \vdash w0$ to be a closed fork for which $\mu(F^*) = \mu(w0)$. Let t_1^* and t_2^* be two edge-disjoint honest tines of F^* so that $\operatorname{reach}_{F^*}(t_1^*) = \rho(F^*)$ and $\operatorname{reach}_{F^*}(t_2^*) = \mu(F^*) = \mu(w0)$. Let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$ and let $\tilde{a}$ be the parameter for this extension. If both t_1^* and t_2^* are tines of F, $\operatorname{reach}_{F^*}(t_i^*) = \operatorname{reach}_F(t_i^*) - (\tilde{a} + 1)$ and, in particular, $\operatorname{reach}_F(t_1^*) \geq \operatorname{reach}_F(t_2^*)$. It follows that $\operatorname{reach}_F(t_2^*) \leq \mu(F) \leq \mu(w) = 0$ and hence that $\mu(w0) < 0$. Otherwise, one of the two tines was the result of extension and has zero reach() in F^*. As $\operatorname{reach}_{F^*}(t_1^*) \geq \operatorname{reach}_{F^*}(t_2^*)$, in either case it follows that $\mu(F^*) = \operatorname{reach}_{F^*}(t_2^*) \leq 0$, as desired.

The case $\rho(w) = 0$. By induction, there is a fork F_w for which $\mathbf{m}(w) = (\rho(F_w), \mu(F_w))$. Let t_1 and t_2 be edge-disjoint tines of F_w for which $\rho(F_w) = \operatorname{reach}(t_1)$ and $\mu(F_w) = \operatorname{reach}(t_2)$. Define $F' \vdash w'$ to be the fork obtained by extending the tine t_1 of F_w with parameter $\tilde{a} = 0$ to yield a new tine t_1' in F'. Then $\operatorname{reach}_{F'}(t_1') = 0$ and $\operatorname{reach}_{F'}(t_2) = \operatorname{reach}_F(t_2) - 1$. It follows that $\rho(w0) \geq 0$ and $\mu(w0) \geq \mu(w) - 1$. We will show that $\rho(w0) \leq 0$ and that $\mu(w0) \leq \mu(w) - 1$, in which case we can conclude that

$$\rho(w0) = 0 \qquad \text{and} \qquad \mu(w0) = \mu(w) - 1\,.$$

Moreover, the fork $F_{w'} = F'$ achieves these statistics, as desired.

We return to establish that $\rho(w0) \leq 0$ and that $\mu(w0) \leq \mu(w) - 1$. Let $F^* \vdash w0$ be a closed fork for which $\rho(w0) = \rho(F^*)$ and let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$; as above, let $\tilde{a}$ denote the parameter for this extension. Let t^* be an honest tine of F^* so that $\operatorname{reach}_{F^*}(t^*) = \rho(w0)$. Note that t^* cannot be a tine of F; if it were then $\operatorname{reach}_{F^*}(t^*) = \operatorname{reach}_F(t^*) - (\tilde{a} + 1) \leq \rho(w) - 1 < 0$ which contradicts $\rho(w0) \geq 0$. Thus t^* was obtained by extension and $\operatorname{reach}_{F^*}(t^*) = 0$. It remains to show that $\mu(w0) \leq 0$. Now let $F^* \vdash w0$ be a closed fork for which $\mu(F^*) = \mu(w0)$. Let t_1^* and t_2^* be two edge-disjoint honest tines of F^* so that $\operatorname{reach}_{F^*}(t_1^*) = \rho(F^*)$ and $\operatorname{reach}_{F^*}(t_2^*) = \mu(F^*) = \mu(w0)$. Let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$ and let $\tilde{a}$ be the parameter for this extension. Similarly, t_1^* cannot be a tine of F; if it were, $\rho(F^*) = \operatorname{reach}_{F^*}(t_1^*) = \operatorname{reach}_F(t_1^*) - (\tilde{a} + 1) \leq \rho(F) - 1 \leq \rho(w) - 1 < 0$ which contradicts $\rho(F) \geq 0$. It follows that t_1^* must extend a tine t_1 of F for which $\operatorname{reach}_F(t_1) = 0$, because extension can only occur for tines of non-negative reach and $\rho(F) = 0 = \rho(w)$. Thus t_2^* is a tine of F and $t_1 \not\sim t_2^*$ so that $\operatorname{reach}_F(t_2^*) \leq \mu(F) \leq \mu(w)$ and we conclude that $\mu(w0) = \operatorname{reach}_{F^*}(t_2^*) \leq \operatorname{reach}_F(t_2^*) - 1 \leq \mu(w) - 1$, as desired.

The case $\rho(w) > 0, \mu(w) \neq 0$. By induction, there is a fork F_w for which $\mathbf{m}(w) = (\rho(F_w), \mu(F_w))$. Let t_1 and t_2 be edge-disjoint tines of F_w for which $\rho(F_w) = \operatorname{reach}(t_1)$ and $\mu(F_w) = \operatorname{reach}(t_2)$. In fact, any extension of F_w will suffice for the construction; for concreteness, define $F' \vdash$

w' to be the fork obtained by extending the tine t_1 of F_w with parameter $\tilde{a} = 0$. Then $\operatorname{reach}_{F'}(t_i) = \operatorname{reach}_{F_w}(t_i) - 1$. It follows that $\rho(w0) \geq \rho(w) - 1$ and $\mu(w0) \geq \mu(w) - 1$. We will show that $\rho(w0) \leq \rho(w) - 1$ and that $\mu(w0) \leq \mu(w) - 1$, in which case we can conclude that

$$\rho(w0) = \rho(w) - 1 \qquad \text{and} \qquad \mu(w0) = \mu(w) - 1\,.$$

Moreover, the fork $F_{w'} = F'$ achieves these statistics, as desired.

We return to establish that $\rho(w0) \leq \rho(w) - 1$ and that $\mu(w0) \leq \mu(w) - 1$. Let $F^* \vdash w0$ be a closed fork for which $\rho(w0) = \rho(F^*)$ and let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$; as above, let $\tilde{a}$ denote the parameter for this extension. Let t^* be an honest tine of F^* so that $\operatorname{reach}_{F^*}(t^*) = \rho(w0)$. Note that if t^* is a tine of F then $\operatorname{reach}_{F^*}(t^*) = \operatorname{reach}_F(t^*) - (\tilde{a} + 1) \leq \rho(w) - 1$; otherwise t^* is obtained by extension and $\operatorname{reach}_{F^*}(t^*) = 0 \leq \rho(w) - 1$, as desired. (Recall that $\rho(w) > 0$.) It remains to show that $\mu(w0) \leq \mu(w) - 1$. Now let $F^* \vdash w0$ be a closed fork for which $\mu(F^*) = \mu(w0)$. Let t_1^* and t_2^* be two edge-disjoint honest tines of F^* so that $\operatorname{reach}_{F^*}(t_1^*) = \rho(F^*)$ and $\operatorname{reach}_{F^*}(t_2^*) = \mu(F^*) = \mu(w0)$. Let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$ and let $\tilde{a}$ be the parameter for this extension. If both t_1^* and t_2^* are tines of F then $\operatorname{reach}_{F^*}(t_i^*) = \operatorname{reach}_F(t_i^*) - (\tilde{a} + 1)$ and, in particular, $\operatorname{reach}_F(t_1^*) \geq \operatorname{reach}_F(t_2^*)$ so that $\operatorname{reach}_F(t_2^*) \leq \mu(w)$ and $\operatorname{reach}_{F*}(t_2^*) \leq \mu(w) - 1$, as desired.

To complete the argument, we consider the case that one of the tines t_i^* arises by extension. Note that in this case $\operatorname{reach}_{F^*}(t_2^*) \leq 0$, as either t_2^* is obtained by extension so that it has zero reach, or t_1^* is obtained by extension so that $\operatorname{reach}_{F^*}(t_2^*) \leq \operatorname{reach}_{F^*}(t_1^*) = 0$. Here we further separate the analysis into two cases depending on the sign of $\mu(w)$:

- If $\mu(w) > 0$, then $\operatorname{reach}_{F^*}(t_2^*) \leq 0 \leq \mu(w) - 1$, as desired.
- If $\mu(w) < 0$ then t_2^* cannot be the extension of a tine in F. To see this, suppose to the contrary that t_2^* extends a tine t_2 of F; then $\operatorname{reach}_F(t_2) \geq 0$. Additionally, t_1^* must be a tine of F, edge-disjoint from t_2, and $\operatorname{reach}_F(t_1^*) = \operatorname{reach}_{F^*}(t_1^*) + (\tilde{a} + 1) > 0$. It follows that $\mu(w) \geq \mu(F) \geq 0$, a contradiction.

  The other possibility is that t_1^* is an extension of a tine t_1 of F in which case $\operatorname{reach}_F(t_1) \geq 0$. Note that t_2^* is a tine of F and edge-disjoint from t_1; thus $\min(\operatorname{reach}_F(t_2^*), \operatorname{reach}_F(t_1)) \leq \mu(F) < 0$ and $\operatorname{reach}_F(t_2^*) \leq \mu(F)$. We conclude that $\operatorname{reach}_{F^*}(t_2^*) = \operatorname{reach}_F(t_2^*) - (\tilde{a}+1) \leq \mu(w) - 1$, as desired. □

With this recursive description in place, we return to the proof of Theorem 4.24, which we restate below for convenience.

Theorem 4.24, restated. *Let $\epsilon \in (0, 1)$ and let w be a string drawn from $\{0, 1\}^n$ by independently assigning each $w_i = 1$ with probability $(1 - \epsilon)/2$. Then*

$$\Pr[w \textit{ is forkable}] = 2^{-\Omega(\sqrt{n})}\,.$$

Proof of Theorem 4.24. The theorem concerns the probability distribution on $\{0, 1\}^n$ given by independently selecting each $w_i \in \{0, 1\}$ so that

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

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

$$R_t = \rho(w_1 \dots w_t) \qquad \text{and} \qquad M_t = \mu(w_1 \dots w_t)\,.$$

Our goal is to establish that

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

We extract from the statement of Lemma 4.28 some facts about these random variables.

$$R_t > 0 \Longrightarrow \begin{cases} R_{t+1} = R_t + 1 & \text{if } w_{t+1} = 1, \\ R_{t+1} = R_t - 1 & \text{if } w_{t+1} = 0; \end{cases} \tag{4}$$

$$M_t < 0 \Longrightarrow \begin{cases} M_{t+1} = M_t + 1 & \text{if } w_{t+1} = 1, \\ M_{t+1} = M_t - 1 & \text{if } w_{t+1} = 0; \end{cases} \tag{5}$$

$$R_t = 0 \Longrightarrow \begin{cases} R_{t+1} = 1 & \text{if } w_{t+1} = 1, \\ R_{t+1} = 0 & \text{if } w_{t+1} = 0, \\ M_{t+1} < 0 & \text{if } w_t = 0. \end{cases} \tag{6}$$

In light of the properties (4) above, the random variables R_t are quite well-behaved when positive—in particular, considering the distribution placed on each w_i, they simply follow the familiar biased random walk of Figure 6. Likewise, considering the properties (5), the random variables M_t follow a biased random walk when negative. The remainder of the proof combines these probability laws with (6) and the fact that $M_t() \leq R_t()$ to establish that $M_n < 0$ with high probability.

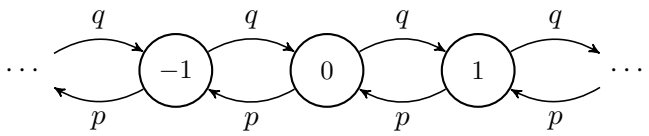


Figure 6: The simple biased walk where $p = (1+\epsilon)/2$ and $q = 1 - p$.

We recall two basic facts about the standard biased walk associated with the Markov chain of Figure 6. 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 properties.

Constant escape probability; gambler's ruin. With constant probability, depending only on ϵ, $Y_t \neq 1$ for all $t > 0$. In general, for each $k > 0$,

$$\Pr[\exists t, Y_t = k] = \alpha^k\,, \tag{7}$$

for a constant $\alpha < 1$ depending only on ϵ. (In fact, the constant α is $(1-\epsilon)/(1+\epsilon)$; see, e.g., [29, Chapter 12] for a complete development.)

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

$$\Pr\left[Y_T > -\frac{\epsilon T}{2}\right] = 2^{-\Omega(T)}\,. \tag{8}$$

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

Partitioning the string w, we write $w = w^{(1)} \cdots w^{(\sqrt{n})}$ where $w^{(t)} = w_{1+a_{t-1}} \dots w_{a_t}$ and $a_t = \lceil t\sqrt{n} \rceil$, for $t = 0, 1, \dots$. Let $R_{(0)} = 0$ and $R_{(t)} = R_{a_t}$; similarly define $M_{(0)} = 0$ and $M_{(t)} = M_{a_t}$. Fix $\delta \ll \epsilon$ to be a small constant.

We define three events based on the random variables $R_{(t)}$ and $M_{(t)}$:

Hot We let Hot_t denote the event that $R_{(t)} \geq \delta\sqrt{n}$ and $M_{(t)} \geq -\delta\sqrt{n}$.

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

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

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

$$\Pr[\mathsf{Cold}_{t+1} \mid \mathsf{Cold}_t] \geq 1 - 2^{-\Omega(\sqrt{n})}\,, \tag{9}$$

$$\Pr[\mathsf{Cold}_{t+1} \mid \mathsf{Vol}_t] \geq \Omega(\epsilon)\,, \tag{10}$$

$$\Pr[\mathsf{Hot}_{t+1} \mid \mathsf{Vol}_t] \leq 2^{-\Omega(\sqrt{n})}\,. \tag{11}$$

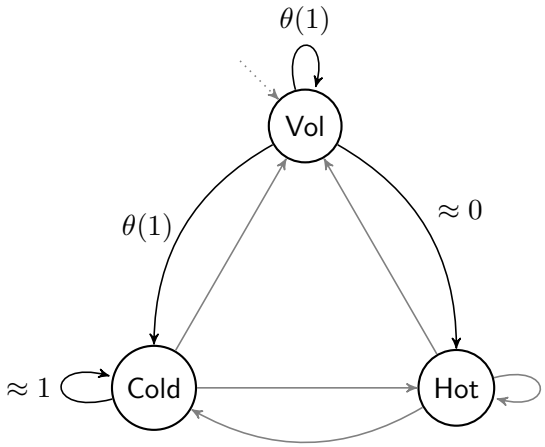


Figure 7: An illustration of the transitions between Cold, Vol, and Hot.

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. In this case, $\mathsf{Cold}_{\sqrt{n}}$ occurs, $M_n^\circ < \delta\sqrt{n} < 0$, and w is not forkable. Specifically, 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 $1 - 2^{-\Omega(\sqrt{n})}$.

It remains to establish the three inequalities (9), (10), and (11).

Inequality (9)**:** This follows directly from (4) and (7). Specifically, in light of (5) the random variables M_i follow the probability law of the simple biased walk when they are negative.

Conditioned on $M_{(t)} = M_{a_t} < -\delta\sqrt{n}$, the probability that any future M_s ever climbs to the value -1 is no more than $\alpha^{-\delta\sqrt{n}} = 2^{-\Omega(n)}$, as desired. (Here $\alpha < 1$ is a fixed constant that depends only on ϵ.)

Inequality (10)**:** This follows from (4), (6), (7), and (8). Specifically, conditioned on Vol_t, $R_{(t)} \leq \delta\sqrt{n}$. Recall from (4) that the random variables R_i follow the probability law of the simple biased walk when they are positive. Let D be the event that $R_i > 0$ for all $a_t \leq i < a_t + 2\delta\sqrt{n}$. According to (8), then, where we take $T = 2\delta\sqrt{n}$, $\Pr[D] \leq 2^{-\Omega(\sqrt{n})}$. With near certainty, then, the random variables R_i visit the value 0 during this period. Observe that if $R_i = 0$ then, by (6), $M_{i+1} \leq -1$ with constant probability and (conditioned on this), by (7), with constant probability the subsequent random variables M_j do not return to the value 0. Additionally, in light of (8), the probability that there is a sequence $w_i \dots w_j$ of length at least $2(\delta/\epsilon)\sqrt{n}$ for which

$$\left(\sum_{k=i}^{j} \begin{cases} 1 & \text{if } w_k = 1, \\ -1 & \text{if } w_k = 0. \end{cases} \right) \geq -\delta\sqrt{n}$$

is no more than $(\sqrt{n})^2 2^{-\Omega(\sqrt{n})}$. It follows that with constant probability, the walk (of R_i) hits 0, as described above, and then M_i terminates at a value less than $-\delta\sqrt{n}$.

Inequality (11)**:** This follows from (4), (6), (7), and (8). Specifically, conditioned on Vol_t, $R_{(t)} \leq \delta\sqrt{n}$. Recall from (4) that the random variables R_i follow the probability law of the simple biased walk when they are positive. Let D be the event that $R_i > 0$ for all $a_t \leq i < a_t + 2\delta\sqrt{n}$. According to (8), then, where we take $T = 2\delta\sqrt{n}$, $\Pr[D] \leq 2^{-\Omega(\sqrt{n})}$. With near certainty, then, the random variables R_i visit the value 0 during this period. Conditioned on $\overline{D}$, in order for $R_{a_{t+1}} \geq \delta\sqrt{n}$ there must be a sequence of these random variables $0 = R_i, R_{i+1}, \dots, R_j = \lfloor \delta\sqrt{n} \rfloor$ so that none of these take the value 0 except the first. (Such a sequence arises by taking i to be the last time the variables $R_{a_t}, \dots$ visit 0 and j the first subsequent time that the sequence is larger than $\delta\sqrt{n}$.) In light of (7), the probability of such a subsequence appearing at a particular value for i is no more than $\alpha^{-\delta\sqrt{n}}$. It follows that the probability that $R_{a_{t+1}} \geq \delta\sqrt{n}$ is less than $\sqrt{n}\alpha^{-\delta\sqrt{n}} = 2^{-\Omega(\sqrt{n})}$, as desired. □

Exact probabilities of forkability for explicit values of n. In order to gain further insight regarding the density of forkable strings, we exactly computed the probability that a string w—drawn so that each w_i is independently assigned the value 1 with probability $p \in \{.40, .41, \dots, .50\}$—is forkable for several different lengths. These results are presented in Figure 8.

Reducing common prefix to forkability. Returning now to the challenge of common prefix, we note that the random assignment of slots to stakeholders given by $\mathcal{F}_{\mathsf{LS}}$ guarantees that the coordinates of the associated characteristic string w are independent Bernoulli random variables taking the value 1 with probability equal to the adversarial stake. Thus Theorem 4.24 establishes that no execution of the protocol π_{iSPoS} can induce two tines (chains) of maximal length with no common prefix.

In the context of π_{iSPoS}, however, we wish to establish a much stronger *common prefix* property: any pair of chains which could, in principle, be presented by the adversary to an honest party 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 a further definition regarding tines and forks.

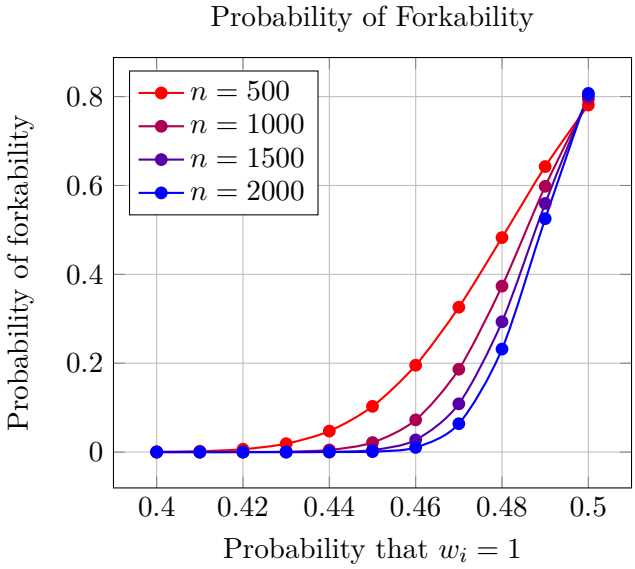


Figure 8: Graphs of the probability that a string drawn so that each bit is independently assigned the value 1 with a certain probability is forkable. Graphs for string lengths $n = 500, 1000, 1500, 2000$ are shown for probabilities $.40, .41, \ldots, .49, .50$.

Definition 4.29 (Divergence)**.** *Let F be a fork for a string $w \in \{0,1\}^*$. For two viable tines t and t' of F we define the notation t/t' by the rule*

$$t/t' = \text{length}(t) - \text{length}(t \cap t')\,,$$

where $t \cap t'$ denotes the common prefix of t and t'. Then define the divergence *of two viable tines t_1 and t_2 to be the quantity*

$$\text{div}(t_1, t_2) = \begin{cases} t_1/t_2 & \text{if } \ell(t_1) < \ell(t_2), \\ t_2/t_1 & \text{if } \ell(t_2) < \ell(t_1), \\ \max(t_1/t_2, t_2/t_1) & \text{if } \ell(t_1) = \ell(t_2). \end{cases}$$

We overload this notation by defining divergence for F as the maximum over all pairs of viable tines:

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

Finally, define the divergence of w *to be the maximum such divergence over all possible forks for w:*

$$\text{div}(w) = \max_{F \vdash w} \text{div}(F)\,.$$

Observe that if $\text{div}(t_1, t_2) \leq k$ and, say, $\ell(t_1) \leq \ell(t_2)$, the tine $t_1^{\lceil k}$ is a prefix of t_2.

Divergence and common prefix violations. Divergence directly reflects the possibility of a common prefix violation. In particular, the characteristic string w satisfies k-cp if and only if $\text{div}(w) \leq k$. We first establish that a string with large divergence must have a large forkable substring. We then apply this in Theorem 4.31 below to conclude that characteristic strings arising from π_{iSPoS} (that is, for which each bit is a Bernoulli random variable) are unlikely to have large

divergence and, hence, possess the common prefix property. The specific fork of relevance, as described above, is F_A arising from the protocol; however, the analysis below will show that no fork realizes large divergence.

Theorem 4.30. *Let $w \in \{0,1\}^*$. Then there is forkable substring $\check{w}$ of w with $|\check{w}| \geq \operatorname{div}(w)$.*

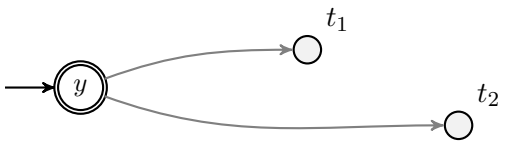
Proof. Consider a string $w \in \{0,1\}^n$, a fork $F \vdash w$, and a pair of viable tines (t_1, t_2) for which

$$\ell(t_1) \leq \ell(t_2) \quad \text{and} \quad t_1/t_2 = \operatorname{div}(w)\,. \tag{12}$$

We may further assume that

$$|\ell(t_2) - \ell(t_1)| \quad \text{is minimum among all pairs of tines for which (12) holds.} \tag{13}$$

We begin by identifying the substring $\check{w}$; the remainder of the proof is devoted to constructing a flat fork for $\check{w}$ to establish forkability. Let y denote the last vertex on the tine $t_1 \cap t_2$, as in the diagram below, and let $\alpha \triangleq \ell(y) = \ell(t_1 \cap t_2)$.



Let β denote the smallest honest index of w for which $\beta \geq \ell(t_2)$, with the convention that if there is no such index we define $\beta = n+1$. It follows that $\ell(t_1) < \beta$. Specifically, if $\ell(t_1) = \ell(t_2) = i$ then this label i is adversarial and hence $\ell(t_1) < \beta$; otherwise, either $\ell(t_1) < \ell(t_2) \leq \beta$ or $\beta = n+1$ and thus $\ell(t_1) < \beta$.

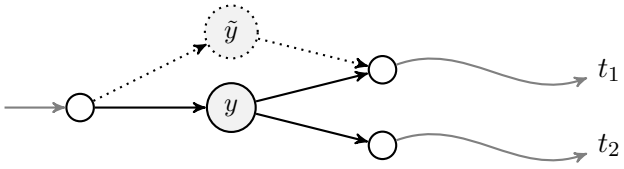
These indices, α and β, distinguish the substring $\check{w} = w_{\alpha+1} \dots w_{\beta-1}$ which will be the subject of the remainder of the proof. As the function $\ell(\cdot)$ is strictly increasing along any tine, observe that

$$|\check{w}| = \beta - \alpha - 1 \geq \ell(t_1) - \ell(y) \geq \operatorname{length}(t_1) - \operatorname{length}(t_1 \cap t_2) = \operatorname{div}(w)\,,$$

so $\check{w}$ has the desired length and it suffices to establish that it is forkable.

We briefly summarize the proof before presenting the details. We begin by establishing several structural properties of the tines t_1 and t_2 that follow from the assumptions (12) and (13) above. To establish that $\check{w}$ is forkable we then extract from F a flat fork (for $\check{w}$) in two steps: (i.) the fork F is subjected to some minor restructuring to ensure that all "long" tines pass through y; (ii.) a flat fork is constructed by treating the vertex y as the root of a portion of the subtree of F labeled with indices of $\check{w}$. At the conclusion of the construction, segments of the two tines t_1 and t_2 will yield the required "long, disjoint" tines satisfying the definition of forkable.

We observe, first of all, that the vertex y cannot be adversarial: otherwise it is easy to construct an alternative fork $\tilde{F} \vdash w$ and a pair of tines in $\tilde{F}$ 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 t_1/t_2—and hence 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 those vertices of F with labels less than or equal to $\alpha = \ell(y)$ has a unique vertex of depth $\operatorname{depth}(y)$ (namely, y itself). In the presence of another vertex $\tilde{y}$ (of F_0) with depth $\operatorname{depth}(y)$, "redirecting" t_1 through $\tilde{y}$ (as in the argument above) would likewise result in a fork with larger divergence. Note that $\ell(\cdot)$ would indeed be increasing along this new tine (resulting from redirecting t_1) because $\ell(\tilde{y}) \leq \ell(y)$ according to the definition of F_0. As α is the last index of the string, this additionally implies that F_0 has no vertices of depth exceeding $\operatorname{depth}(y)$.

We remark that the minimality assumption (13) implies that any honest index h for which $h < \beta$ has depth no more than $\min(\operatorname{length}(t_1), \operatorname{length}(t_2))$: specifically,

$$h < \beta \quad \implies \quad \mathbf{d}(h) \leq \min(\operatorname{length}(t_1), \operatorname{length}(t_2)) \,. \tag{14}$$

To see this, consider an honest index $h < \beta$ and the tine t_h for which $\ell(t_h) = h$. Recall that t_1 and t_2 are viable; as $h < \ell(t_2)$ it follows immediately that $\mathbf{d}(h) \leq \operatorname{length}(t_2)$. Similarly, if $h \leq \ell(t_1)$ then $\mathbf{d}(h) \leq \operatorname{length}(t_1)$, so it remains to settle the case when $\ell(t_1) < h < \ell(t_2)$: in particular, in this regime we wish to likewise guarantee that $\mathbf{d}(h) \leq \operatorname{length}(t_1)$. For the sake of contradiction, assume that $\operatorname{length}(t_h) = \mathbf{d}(h) > \operatorname{length}(t_1)$. Considering the tine t_h, we separately investigate two cases depending on whether t_h shares an edge with t_1 after the vertex y. If, indeed, t_h and t_1 share an edge after the vertex y then t_h and t_2 do not share such an edge, and we observe that $\operatorname{div}(w) \geq t_h/t_2 \geq t_1/t_2 = \operatorname{div}(w)$ (and hence $\operatorname{div}(w) = \operatorname{div}(t_h, t_2)$) while $|\ell(t_2) - h| < |\ell(t_2) - \ell(t_1)|$ which contradicts (13). If, on the other hand, t_h shares no edge with t_1 after y, we similarly observe that $\operatorname{div}(w) \geq t_1/t_h \geq t_1/t_2 = \operatorname{div}(w)$ while $|h - \ell(t_1)| < |\ell(t_2) - \ell(t_1)|$, which contradicts (13).

In light of the remarks above, we observe that the fork F may be "pinched" at y to yield an essentially identical fork $F^{\triangleright y \triangleleft} \vdash w$ with the exception that all tines of length exceeding $\operatorname{depth}(y)$ pass through the vertex y. Specifically, the fork $F^{\triangleright y \triangleleft} \vdash w$ is defined to be the graph obtained from F by changing every edge of F directed towards a vertex of depth $\operatorname{depth}(y) + 1$ so that it originates from y. To see that the resulting tree is a well-defined fork, it suffices to check that $\ell(\cdot)$ is still increasing along all tines of $F^{\triangleright y \triangleleft}$. For this purpose, consider the effect of this pinching on an individual tine t terminating at a particular vertex v—it is replaced with a tine $t^{\triangleright y \triangleleft}$ defined so that:

- If $\operatorname{length}(t) \leq \operatorname{depth}(y)$, the tine t is unchanged: $t^{\triangleright y \triangleleft} = t$.
- Otherwise, $\operatorname{length}(t) > \operatorname{depth}(y)$ and t has a vertex z of depth $\operatorname{depth}(y) + 1$; note that $\ell(z) > \ell(y)$ because F_0 contains no vertices of depth exceeding $\operatorname{depth}(y)$. Then $t^{\triangleright y \triangleleft}$ is defined to be the path given by the tine terminating at y, a (new) edge from y to z, and the suffix of t beginning at z. (As $\ell(z) > \ell(y)$ this has the increasing label property.)

Thus the tree $F^{\triangleright y \triangleleft}$ is a legal fork on the same vertex set; note that depths of vertices in F and $F^{\triangleright y \triangleleft}$ are identical.

By excising the tree rooted at y from this pinched fork $F^{\triangleright y \triangleleft}$ we may extract a fork for the string $w_{\alpha+1} \dots w_n$. Specifically, consider the induced subgraph $F^{y \triangleleft}$ of $F^{\triangleright y \triangleleft}$ given by the vertices $\{y\} \cup \{z \mid \operatorname{depth}(z) > \operatorname{depth}(y)\}$. By treating y as a root vertex and suitably defining the labels $\ell^{y \triangleleft}$ of $F^{y \triangleleft}$ so that $\ell^{y \triangleleft}(z) = \ell(z) - \ell(y)$, this subgraph has the defining properties of a fork for $w_{\alpha+1} \dots w_n$. In particular, considering that α is honest it follows that each honest index $h > \alpha$ has depth $\mathbf{d}(h) > \operatorname{length}(y)$ and hence labels a vertex in $F^{y \triangleleft}$. For a tine t of $F^{\triangleright y \triangleleft}$, we let $t^{y \triangleleft}$ denote the suffix of this tine beginning at y, which forms a tine in $F^{y \triangleleft}$. (If $\operatorname{length}(t) \leq \operatorname{depth}(y)$, we define $t^{y \triangleleft}$ to consist solely of the vertex y.) Note that $t_1^{y \triangleleft}$ and $t_2^{y \triangleleft}$ share no edges in the fork $F^{y \triangleleft}$.

Finally, let $\check{F}$ denote the tree obtained from $F^{y\triangleleft}$ as the union of all tines t of $F^{y\triangleleft}$ so that all labels of t are drawn from $\check{w}$ (as it appears as a prefix of $w_{\alpha+1}\dots w_n$), and

$$\operatorname{length}(t) \leq \max_{\substack{h\leq|\check{w}|\\ h \text{ honest}}} \mathbf{d}(h)\,.$$

It is immediate that $\check{F}\vdash\check{w}$. To conclude the proof, we show that $\check{F}$ is flat. For this purpose, we consider the tines $t_1^{y\triangleleft}$ and $t_2^{y\triangleleft}$. As mentioned above, they share no edges in $F^{y\triangleleft}$, and hence the prefixes $\check{t_1}$ and $\check{t_2}$ (of $t_1^{y\triangleleft}$ and $t_2^{y\triangleleft}$) appearing in $\check{F}$ share no edges. We wish to see that these prefixes have maximum length in $\check{F}$, in which case $\check{F}$ is flat, as desired. This is immediate for the tine $\check{t_1}$ because all labels of $t_1^{y\triangleleft}$ are drawn from $\check{w}$ and, considering (14), its depth is at least that of all relevant honest vertices. As for $\check{t_2}$, observe that if $\ell(t_2)$ is not honest then $\beta>\ell(t_2)$ so that, as with $\check{t_1}$, the tine $\check{t_2}$ is labeled by $\check{w}$ so that the same argument, relying on (14), ensures that $\check{t_2}$ has length at least that of all relevant honest vertices. If $\ell(t_2)$ is honest, $\beta=\ell(t_2)$, and the terminal vertex of $t_2^{y\triangleleft}$ does not appear in $\check{F}$ (as it does not index $\check{w}$). In this case, however, $\operatorname{length}(t_2^{y\triangleleft})>\mathbf{d}(h)$ for any honest index of $\check{w}$, and it follows that $\operatorname{length}(\check{t_2})=\operatorname{length}(t_2^{y\triangleleft})-1$ is at least the depth of any honest index of $\check{w}$, as desired. □

Theorem 4.31. *Let $k,L\in\mathbb{N}$ and $\epsilon\in(0,1/2)$. Let $w=w_1\dots w_L$ of length L, where each w_i is independently distributed in $\{0,1\}$ so that $\Pr[w_i=1]=1/2-\epsilon$. Then*

$$\Pr[w \textit{ does not possess } k\text{-}\mathsf{cp}]=\epsilon_{\mathsf{cp}}(k;L,\epsilon)=L\exp(-\Omega(\sqrt{k}))\,.$$

The constant hidden by the $\Omega()$ notation depends only on ϵ.

Proof. Recall that w violates k-cp precisely when there is a fork $F\vdash w$ for which $\operatorname{div}(F)\geq k$. Thus we wish to show that the probability that $\operatorname{div}(w)\geq k$ is no more than $\exp(-\Omega(\sqrt{k})+\log L)$.

It follows from Theorem 4.30 that if $\operatorname{div}(w)\geq k$, there is a forkable substring $\check{w}$ of length at least k. Thus

$$\begin{aligned}\Pr[w \text{ does not possess } k\text{-}\mathsf{cp}] &\leq \Pr\begin{bmatrix}\exists\alpha,\beta\in\{1,\dots,L\} \text{ so that } \alpha+k-1\leq\beta \text{ and}\\ w_\alpha\dots w_\beta \text{ is forkable}\end{bmatrix}\\ &\leq \sum_{1\leq\alpha\leq L}\underbrace{\sum_{\alpha+k-1\leq\beta\leq L}\Pr[w_\alpha\dots w_\beta \text{ is forkable}]}_{(*)}\,.\end{aligned}$$

According to Theorem 4.24 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 $\alpha\geq 1$,

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

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

$$\Pr[w \text{ does not possess } k\text{-}\mathsf{cp}]\leq L\cdot\exp(-\Omega(\sqrt{k}))\leq\exp(\ln L-\Omega(\sqrt{k}))\,,$$

as desired. □

4.7 Security Analysis of the Idealized Protocol

Theorem 4.32. *Consider the execution $\mathcal{E}$ of π_{iSPoS} over a lifetime of L slots with a $(1/2-\epsilon)$-initially-bounded adversary $\mathcal{A}$ and environment $\mathcal{Z}$. Then persistence with parameter k fails to hold with probability no more than*

$$\epsilon_{\mathit{CP}}(k; L, \epsilon) + \epsilon_{\mathit{H}} = L \cdot \exp(-\Omega(\sqrt{k})) + \epsilon_{\mathit{H}}$$

and liveness, with parameter $u = 2(k+\ell)$, fails to hold with probability no more than

$$\epsilon_{\mathit{HCG}}(1/2, 2k; L, \epsilon) + \epsilon_{\exists \mathit{CQ}}(\ell; L, \epsilon) + \epsilon_{\mathit{H}} = O(L \cdot [\exp(-4\epsilon^2 k) + \exp(-2\epsilon^2 \ell)]) + \epsilon_{\mathit{H}}\,,$$

where ϵ_{H} denotes the probability of a collision across all queries to H by the participants in the protocol (including $\mathcal{A}$). All probabilities are conditioned on $\neg\mathsf{Bad}^{1/2-\epsilon}$ (which is to say that we assume executions in which the adversary is $(1/2-\epsilon)$-bounded).

Proof. Recall that the $\mathcal{F}_{\mathsf{LS}}[\mathcal{F}_{\mathsf{DSIG}}]$ functionality directly provides a leader selection schedule and—for any adversarial selection of corrupted players with total stake $(1/2-\epsilon)$—yields a characteristic string $w = w_1, \ldots, w_L$ where each w_i is selected independently and $\mathbb{E}[w_i] = 1/2 - \epsilon$. Recall that

- we have the luxury of an ideal signature scheme and, furthermore,
- the probability—taken across all evaluations of the hash function by all participants of the protocol including the adversary—that a collision is observed is no more than $\epsilon_{\mathsf{H}}(q) \triangleq q^2/|\mathit{Range}|$, where q is the total number of queries to the hash function and $|\mathit{Range}|$ is the size of the range of the function.

In light of these properties, unless there is an (unlikely) collision among the queries to the hash function, the adopted chains form a well-defined fork $F_A \vdash w$. We observe that if w satisfies cp, hcg, and $\exists\mathsf{cq}$, the execution $\mathcal{E}$ satisfies CP, HCG, and $\exists\mathsf{CQ}$ with the same parameters. (It is possible, of course, that the execution $\mathcal{E}$ actually satisfies these properties with stronger parameters, as the execution yields a specific fork (F_A) while the abstract quantities cp, cg, and $\exists\mathsf{cq}$ for w are worst case estimates taken over all forks.)

We observe that a violation of persistence would imply directly a violation of k-CP. As a result

$$\begin{aligned}\Pr[\mathcal{E} \text{ violates persistence}] &\leq \Pr[F_A \text{ violates } k\text{-}\mathsf{cp}] + \epsilon_{\mathsf{H}} \leq \Pr[w \text{ violates } k\text{-}\mathsf{cp}] + \epsilon_{\mathsf{H}} \\ &\leq \epsilon_{\mathsf{CP}}(k; L, \epsilon) + \epsilon_{\mathsf{H}} \leq L \cdot \exp(-\Omega(\sqrt{k})) + \epsilon_{\mathsf{H}}\,,\end{aligned}$$

where the last inequality follows directly from Theorem 4.31.

Now regarding liveness with parameter $u = 2(k+\ell)$, we observe that it is implied by $(1/2, 2k)$-HCG and ℓ-$\exists\mathsf{CQ}$. Specifically, consider a transaction tx being provided as part of input data for a period S of u slots. Let t be any viable tine that spans S. In light of ℓ-$\exists\mathsf{CQ}$, t contains an honestly generated block associated with a slot sl_1 among the first ℓ slots of S and—as tx was available to the honest party generating this block—it must appear in $t[0, sl_1]$. Similarly, an honest block must be associated with a slot sl_2 among the last ℓ slots of S. As $u = 2(k+\ell)$, $sl_2 \geq sl_1 + 2k$ and, as a consequence of $(1/2, 2k)$-HCG, the tine t contains at least k blocks after sl_1 (but indexed by slots in S); specifically, $\text{length}(t(sl_1, sl_2]) \geq k$. This guarantees u-liveness. Thus

$$\begin{aligned}\Pr[\mathcal{E} \text{ violates liveness}] &\leq \Pr[F_A \text{ violates } \ell\text{-}\exists\mathsf{cq}, \text{ or } (1/2, 2k)\text{-}\mathsf{hcg}] + \epsilon_{\mathsf{H}} \\ &\leq \Pr[w \text{ violates } \ell\text{-}\exists\mathsf{cq}, \text{ or } (1/2, 2k)\text{-}\mathsf{hcg}] + \epsilon_{\mathsf{H}} \\ &\leq \epsilon_{\exists\mathsf{CQ}}(\ell; L, \epsilon) + \epsilon_{\mathsf{HCG}}(1/2, 2k; L, \epsilon) + \epsilon_{\mathsf{H}} \\ &\leq L[O(\exp(-2\epsilon^2\ell) + O(\exp(-4\epsilon^2 k)] + \epsilon_{\mathsf{H}}\,,\end{aligned}$$

which completes the proof. □

5 Analysis of the Dynamic Stake Protocol

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 $\mathcal{F}_{\mathsf{LS}}$ ideal functionality to now provide randomness and auxiliary information for the leader election process throughout the epochs (the enhanced functionality will be called $\mathcal{F}_{\mathsf{DLS}}$). We then discuss how to implement $\mathcal{F}_{\mathsf{DLS}}$ using only $\mathcal{F}_{\mathsf{LS}}$ 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 $\mathcal{F}_{\mathsf{LS}}$ so that multiple epochs are considered. The resulting functionality, $\mathcal{F}_{\mathsf{DLS}}$, allows stakeholders to query it for the leader selection data specific to each epoch. $\mathcal{F}_{\mathsf{DLS}}$ 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 in the latest block of the previous epoch's first $R-k'$ slots (k' is parameter we set below). Given that there is no predetermined view of the stakeholder distribution, the functionality $\mathcal{F}_{\mathsf{DLS}}$ will provide only a random string and will leave the interpretation according to the stakeholder distribution to the party that is calling it. The effective stakeholder distribution is the sequence $\mathbb{S}_1, \mathbb{S}_2, \ldots$ defined as follows: $\mathbb{S}_1$ is the initial stakeholder distribution; for slots $\{(j-1)R+1, \ldots, jR\}$ for $j \geq 2$ the effective stakeholder $\mathbb{S}_j$ is determined by the stake allocation that is found in the latest block with timestamp less than $(j-1)R-k'$, provided all honest parties agree on it, or is undefined if the honest parties disagree on it. The functionality $\mathcal{F}_{\mathsf{DLS}}$ is defined in Figure 9. For maximum flexibility, and to anticipate the needs of the next phase of our analysis, we allow the adversary to perform a lookahead on the beacon value for $E \geq 0$ slots prior to the onset of the epoch. To account for this lookahead the value of k' must be suitably adjusted; see below.

Functionality $\mathcal{F}_{\mathsf{DLS}}$

$\mathcal{F}_{\mathsf{DLS}}$ incorporates the diffuse and key/transaction functionality from Section 2 and is parameterized by the public keys and respective stakes of the initial (before epoch e_1 starts) stakeholders $\mathbb{S}_0 = \{(\mathsf{vk}_1, s_1^0), \ldots, (\mathsf{vk}_n, s_n^0)\}$ a distribution $\mathcal{D}$ and a leader selection function F. In addition, $\mathcal{F}_{\mathsf{DLS}}$ operates as follows:

- **Genesis Block Generation** Upon receiving $(\mathsf{genblock\_req}, U_i)$ from stakeholder U_i it operates as functionality $\mathcal{F}_{\mathsf{LS}}$ on that message.
- **Signature Key Pair Generation** It operates as functionality $\mathcal{F}_{\mathsf{LS}}$.
- **Epoch Randomness Update** Upon receiving $(\mathsf{epochrnd\_req}, U_i, e_j)$ from stakeholder U_i, if $j \geq 2$ is the current epoch, $\mathcal{F}_{\mathsf{DLS}}$ proceeds as follows. If ρ^j has not been set, $\mathcal{F}_{\mathsf{DLS}}$ samples $\rho^j \leftarrow \mathcal{D}$. Then, $\mathcal{F}_{\mathsf{DLS}}$ sends $(\mathsf{epochrnd}, \rho^j)$ to U_i. The adversary is allowed a lookahead of $E \geq 0$ slots to this interface where E is a parameter of the functionality.

Figure 9: Functionality $\mathcal{F}_{\mathsf{DLS}}$ with beacon lookahead parameter $E \geq 0$.

We now describe protocol π_{DPoS}, which is a modified version of π_{SPoS} that updates its genesis

block data (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:

> Function $\mathsf{maxvalid}_k(\mathcal{C}, \mathbb{C})$. 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 $\mathbb{C}$.

As a matter of notation, we simply refer to this rule as $\mathsf{maxvalid}()$ when the parameter can be inferred from context.

Protocol π_{DPoS} is described in Figure 10 and functions in the $\mathcal{F}_{\mathsf{DLS}}$-hybrid model. We also define an idealised version of this protocol π_{iDPoS} for which it holds that access to the digital signature is performed via the interface of $\mathcal{F}_{\mathsf{DSIG}}$.

Protocol π_{DPoS} ; with parameters k, ℓ, E, R, L satisfying $2(k+\ell)+E < R$.

π_{DPoS} is a protocol run by a set of stakeholders, initially equal to $U_1, \ldots, U_n$, interacting with $\mathcal{F}_{\mathsf{DLS}}$ over a sequence of L slots $S = \{1, \ldots, L\}$ divided in epochs of length $R < L$. π_{DPoS} proceeds as follows:

1. **Initialization** Stakeholder $U_i \in \{U_1, \ldots, U_n\}$, receives from the key registration interface its public and secret key. Then it receives the current slot from the diffuse interface and in case it is sl_1 it sends $(\mathsf{genblock\_req}, U_i)$ to $\mathcal{F}_{\mathsf{LS}}$, receiving $(\mathsf{genblock}, \mathbb{S}_0, \rho, \mathsf{F})$ as the answer. U_i sets the local blockchain $\mathcal{C} = B_0 = (\mathbb{S}_0, \rho)$ and the initial internal state $st = H(B_0)$. Otherwise, it receives from the key registration interface the initial chain $\mathcal{C}$, sets the local blockchain as $\mathcal{C}$ and the initial internal state $st = H(\mathrm{head}(\mathcal{C}))$.
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 drawn from the most recent block with timestamp less than $jR - 2(k+\ell) - E$ where $E \geq 0$ is the lookahead the adversary is allowed on the beacon and sends $(\mathsf{epochrnd\_req}, U_i, e_j)$ to $\mathcal{F}_{\mathsf{LS}}$, receiving $(\mathsf{epochrnd}, \rho^j)$ as answer.
   (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 $U' = \mathsf{F}(\mathbb{S}_{j'}, \rho^{j'}, sl')$ with $e_{j'}$ being the epoch in which the slot B' belongs (as determined by sl'). U_i computes $\mathcal{C}' = \mathsf{maxvalid}(\mathcal{C}, \mathbb{C})$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\mathrm{head}(\mathcal{C}'))$.
   (c) If U_i is the slot leader determined by $\mathsf{F}(\mathbb{S}_j, \rho^j, sl)$ 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 the data and $\sigma = \mathsf{Sign}_{\mathsf{sk}_i}(st, d, sl)$ is a signature on (st, d, sl). U_i computes $\mathcal{C}' = \mathcal{C}|B$, broadcasts $\mathcal{C}'$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\mathrm{head}(\mathcal{C}'))$.
3. **Transaction generation** as in protocol π_{SPoS}.

Figure 10: Protocol π_{DPoS}

With a similar argument as in Proposition 4.9 we have the following statement.

Proposition 5.1. *For each PPT $\mathcal{A}, \mathcal{Z}$ it holds that there is a PPT $\mathcal{S}$ so that*

$$\mathsf{EXEC}^{\mathcal{F}_{\mathsf{DLS}}[\mathsf{SIG}]}_{\pi_{\mathrm{DPoS}}, \mathcal{A}, \mathcal{Z}}(\lambda) \quad \textit{and} \quad \mathsf{EXEC}^{\mathcal{F}_{\mathsf{DLS}}[\mathcal{F}_{\mathsf{DSIG}}]}_{\pi_{\mathrm{iDPoS}}, \mathcal{S}, \mathcal{Z}}(\lambda)$$

are computationally indistinguishable.

In the light of this proposition in the remaining of this section we will focus on the analysis of iDPoS.

Remark 1. *The modification to* $\mathsf{maxvalid}(\cdot)$ *to not diverge more than* k *blocks from the last chain possessed will require stakeholders to be online at least every* k *slots. The relevance of the rule comes from the fact that as stake shifts over time, it will be feasible for the adversary to corrupt stakeholders that used to possess a stake majority at some point without triggering* $\mathsf{Bad}^{1/2}$ *and thus any adversarial chains produced due to such an event should be rejected.*

We are now ready to state the main result of the section that establishes that the π_{DPOS} protocol is a robust transaction ledger under the environmental conditions that we have assumed. Recall that in the dynamic stake case we have to ensure that the adversary cannot exploit the way stake changes over time and corrupt a set of stakeholders that will enable the control of the majority of an elected committee of stakeholders in an epoch. In order to capture this dependency on stake "shifts," we introduce the following property.

Definition 5.2 (Stake shift)**.** *Consider two slots* sl_1, sl_2 *and an execution* $\mathcal{E}$*. The* stake shift *between* sl_1 *and* sl_2 *is the maximum statistical distance, taken over all chains* $\mathcal{C}$ *adopted by an honest party, of the two weighted-by-stake distributions that are defined by* $\mathcal{C}[0 : sl_1]$ *and* $\mathcal{C}[0 : sl_2]$*.*

With this definition in place we are ready state the main theorem.

Theorem 5.3. *Fix parameters* k*,* ℓ*,* ϵ, σ*,* E*,* R*, and* L*, where* $R > 2(k + \ell) + E$ *and* L *is an integer multiple of* R*. Consider an execution* $\mathcal{E}$ *of* π_{iDPoS} *with lifetime* L *coupled with a* $(1/2 - \epsilon)$*-initially-bounded adversary* $\mathcal{A}$ *with corruption delay* $D = R + E + 2(k + \ell)$ *and beacon lookahead* E *and environment* $\mathcal{Z}$ *exhibiting a stake-shift of no more than* σ *over any period of* ℓ *slots. Then persistence, with parameter* k*, and liveness, with parameter* $u = 2(k+\ell)$*, are violated with probability no more than*

$$\epsilon_{\mathsf{CP}}(k; L, \epsilon) + \epsilon_{\mathsf{HCG}}(1/2, 2k; L, \epsilon) + \epsilon_{\exists\mathsf{CQ}}(\ell; L, \epsilon) + \epsilon_{\mathsf{H}} \leq L \cdot \left[\exp(-\Omega(\sqrt{k})) + O(\exp(-2\epsilon^2\ell))\right] + \epsilon_{\mathsf{H}}\,.$$

Here ϵ_{H} *denotes the probability of a collision occurring among the queries to* H *(including those of* $\mathcal{A}$*). The above probabilities are in the conditional space* $\neg\mathsf{Bad}^{1/2-\epsilon-\sigma}$*.*

Proof. Consider an execution $\mathcal{E}$ generated by π_{iDPoS} with a $(1/2-\epsilon)$-initially-bounded adversary $\mathcal{A}$ with corruption delay $D = R+E+2(k+\ell)$ and environment $\mathcal{Z}$. We implicitly condition throughout on $\neg\mathsf{Bad}^{1/2-\epsilon-\sigma}$ by simply working in this conditioned probability space; we furthermore condition on the event that there are no collisions among the queries made to the hash function by all participants and the adversary: rather than adjusting the ambient probability space for this second conditioning, we introduce an error term ϵ_{H} into our security guarantees. In terms of the random variable $\mathcal{E}$, we begin by defining several other random variables. The most important family of random variables capture the high-level chain properties of the protocol:

Chain guarantees. For each t, Chain_t is the event that, over the course of epochs numbered 1 through t,

- CP is satisfied with parameter k,
- $\exists\mathsf{CQ}$ is satisfied with parameter ℓ, and
- HCG is satisfied with parameters $(1/2, 2k)$.

We remark, by reasoning parallel to that of Theorem 4.32, that these properties directly imply CG with parameters $(k/(2k+2\ell), 2(k+\ell))$. (In particular, this guarantees growth of k blocks over any period of $2(k+\ell)$ slots.)

For convenience, we adopt the convention that Chain_0 occurs by fiat.

We continue to define a number of ancillary random variables which only take on relevant values in case Chain_t occurs.

Instantaneous common prefix. When Chain_t occurs, the chains adopted by the family of honest players at the outset of slot tR share a common prefix through slot $tR - 2(k+\ell)$; we let $\mathcal{C}^{(t)}$ denote this common chain. This follows directly from the chain properties discussed above, which guarantee k-CP and growth of k blocks over any window of $2(k+\ell)$ slots.

The analysis below critically relies on the property that $\mathcal{C}^{(t)}$ is *immutable*: specifically, when Chain_t occurs all honest participants agree on $\mathcal{C}^{(t)}$ and, as $\mathsf{maxvalid}_k$ can only revise the last k blocks of a currently adopted chain, $\mathcal{C}^{(t)}$ will be a prefix of all future chains held by the honest parties.

If Chain_t does not occur, we define $\mathcal{C}^{(t)} = \perp$.

Instantaneous stake distribution; instantaneous characteristic string. $\mathbb{S}_1$ is defined by the genesis block. For each $t > 1$, we define $\mathbb{S}_t$ to be the stake distribution determined by $\mathcal{C}^{(t-1)}$ if Chain_{t-1} occurs; otherwise, we set $\mathbb{S}_t = \perp$.

For each t, we define a random variable $w^{(t)} \in \{0,1\}^R \cup \{\perp^R\}$ so that:

- If Chain_{t-1} occurs, then $w^{(t)}$ is the characteristic string associated with epoch t. More precisely, this is the characteristic string defined by (i.) the leader election schedule determined by stake distribution $\mathbb{S}_t$ and the beacon ρ^t, and (ii.) the set of parties corrupted, or selected for corruption, by the adversary as of slot $R(t-1) - E - 2(\ell - k)$. (As we work with delay $D = R + E + 2(\ell + k)$, this includes all parties that will be under adversarial control at any point during epoch t.)
- If Chain_{t-1} does not occur, $w^{(t)} = \underbrace{\perp \cdots \perp}_{R}$.
- For later convenience, we also identify the random variables Hon_t which indicate if a majority of slots in epoch t are honest (which is to say that the Hamming weight of $w^{(t)}$ is strictly less than $R/2$). Specifically, we define $\mathsf{Hon}_t = \perp$ if $w^{(t)} = \perp$, $\mathsf{Hon}_t = 1$ if $\sum w_i^{(t)} < R/2$, and $\mathsf{Hon}_t = 0$ otherwise.

We let $w^{(<t)}$ denote the concatenation $w^{(1)} \cdots w^{(t-1)}$.

We use the word "instantaneous" above to emphasize that these random variables are defined by the state of the protocol at a particular time slot, and that the various features they describe (e.g., stake distributions, chains, etc.) evolve over the future course of the protocol.

We note some critical properties of these random variables.

First of all, considering that $\mathsf{Chain}_t \Rightarrow \mathsf{Chain}_s$ for all $s \leq t$, the occurrence of Chain_t implies that a variety of protocol data established during these epoch persists through the rest of the protocol. Specifically, $\mathcal{C}^{(1)} \preceq \cdots \preceq \mathcal{C}^{(t)}$ and, more generally, each $\mathcal{C}^{(s)}$ is a prefix of the chains adopted by all honest parties during epochs $s+1, s+2, \ldots$. As a result, the honest parties unanimously agree on the stake distribution $\mathbb{S}_s$, for each $1 \leq s \leq t+1$, both at the end of the epoch in which they are defined and throughout the rest of the protocol.

Continuing to discuss the ramifications of Chain_t, the fact that the honest parties agree on $\mathbb{S}_1, \ldots, \mathbb{S}_{t+1}$ and, of course, agree on the beacon values, yields persistent agreement on the leader schedules for the first $t+1$ epochs. As a matter of analysis, we remark that Chain_t thus unambiguously defines the characteristic string $w = w^{(1)} \ldots w^{(t+1)}$ and a fork $F_A^{t+1} \vdash w$ associated with all chains adopted by honest parties. Note, in particular, the graph structure of F_A^{t+1} is simply defined by the unambiguous leader schedules, the structure of the chains held by honest players, the fact that no collisions are observed by the hash function, and the ideal signature scheme. To check that $F_A^{t+1} \vdash w$, it suffices to ensure that any slot behaving adversarially vis-a-vis the *structure* of the fork—for example, violating the longest chain rule or generating multiple blocks—is correctly reflected by the definition of w. Note, however, that a block signed by the leader of slot i (and associated with slot number i) can only be adopted by an honest player during the next $2(k+\ell)$ slots, as this is sufficient time for any player to accumulate a chain of length k beyond which no alteration to slot i is possible. As w surely identifies a slot as adversarial if the leader will be under adversarial control during the next $R > 2(k+\ell)$ slots, this definition of w supports the fork F_A^{t+1}, as desired.

We shift our attention to the distribution determined on w. Conditioned on Chain_t, as noted above, the stake distribution $\mathbb{S}_{t+1}$ is unambiguously determined by $\mathcal{C}^{(t)}[0 : Rt - 2(k+\ell) - E]$ (cf. part (a) of the Chain Extension step of Figure 10). As of slot $Rt - E$, when the beacon is exposed to the adversary, this chain is stable in the view of all honest parties; that is—applying ℓ-$\exists\mathsf{CQ}$ and $(1/2, 2k)$-HCG as above—any honest party has added at least k blocks to the chain so that it will exist as a prefix of all honest parties' chains for the remainder of the protocol. It follows that the beacon value ρ^{t+1} is independent of $\mathbb{S}_{t+1}$. Consider now the adversarial stake distribution indicated by $\mathcal{C}[0 : Rt - E - 2(k+\ell)]$. As the last block of this chain might not be honestly generated, we cannot directly apply the guarantee provided by $\mathsf{Bad}^{1/2-\epsilon-\sigma}$ to this distribution; however, in light of ℓ-$\exists\mathsf{CQ}$, there is an honestly generated block on $\mathcal{C}$ appearing no more than ℓ slots beyond the last block of $\mathcal{C}[0 : Rt - E - 2(k+\ell)]$ so that, recalling the guarantee of σ stake shift over any ℓ slots, the adversarial stake ratio given by $\mathcal{C}[0 : Rt - E - 2(k+\ell)]$ is no more than $1/2 - \epsilon$. It follows that

$$\Pr[w_r^{(t)} = 1 \mid w^{(<t)}, w_1^{(t)}, \ldots, w_{r-1}^{(t)}] \leq 1/2 - \epsilon$$

and that the fork F_A^{t+1} determined by epoch $t+1$ of the protocol has the property that $F_A^{t+1} \vdash w^{(1)} \ldots w^{(t+1)}$.

To complete the proof, consider the "virtual characteristic string" x obtained by substituting all $\perp$ in $w = w^{(1)} \ldots w^{(L/R)}$ with 0. Note that if Chain_t fails for some t, the string w—and hence the string x—must violate k-cp, ℓ-$\exists\mathsf{cq}$, or $(1/2, 2k)$-HCG. Observe, also, that the string x, taking values in $\{0,1\}^L$ has the property that for each t

$$\Pr[x_t = 1 \mid x_1, \ldots, x_{t-1}] \leq 1/2 - \epsilon$$

and hence, by Lemma 4.18, that the random variable x is stochastically dominated by the random variable $b = b_1 \ldots, b_L \in \{0,1\}^L$ given by independently assigning each $b_i = 1$ with probability exactly $1/2 - \epsilon$. The event that a characteristic string violates k-cp, ℓ-$\exists\mathsf{cq}$, or $(1/2, 2s)$-cg is clearly monotone, as any violation is preserved by monotonically increasing the set of adversarial slots. We remark, additionally, that if a characteristic string satisfies k-cp then, for any $t \geq k$, the Hamming weight of the string is no less than $t/2$ over any interval of t slots (otherwise there is an immediate k-cp violation). In light of Lemma 4.19, Lemma 4.22, and Theorem 4.31, it follows that

$$\begin{aligned}\Pr[x \text{ violates } k\text{-}\mathsf{cp},\ \ell\text{-}\exists\mathsf{cq}, \text{ or } (1/2,2k)\text{-}\mathsf{cg}] &\leq \Pr[b \text{ violates } k\text{-}\mathsf{cp},\ \ell\text{-}\exists\mathsf{cq}, \text{ and } (1/2,2k)\text{-}\mathsf{cg}] \\ &\leq L\left[O(\exp(-2\epsilon^2\ell) + \exp(-\Omega(\sqrt{k}))\right] + \epsilon_{\mathsf{H}}\,.\end{aligned}$$

To conclude, observe that if x satisfies k-cp, s-$\exists\mathsf{cq}$, and $(1/2, 2s)$-cg, then Chain_t holds for all t and hence $w = x$. In light of the comment above, in this case Hon_t is also satisfied for all t. It follows, in this case, that $\mathcal{E}$ satisfies k-CP, ℓ-$\exists\mathsf{CQ}$, and $(1/2, 2k)$-CG, as desired.

These properties imply persistence with parameter k and liveness with parameter $u = 2(k + \ell)$. □

Liveness and persistence for π_{DPoS}. We observe now that Proposition 5.1 and Theorem 5.3 can be combined to show the security of protocol π_{DPoS} in the setting of a trusted beacon: in particular, if there is an attack against persistence or liveness against π_{DPoS}, it can be transformed to an attack against π_{iDPoS}. It follows that π_{DPoS} possesses the same liveness and persistence properties as π_{iDPoS}, with an extra error term to account for failure of the signature scheme. We omit further details as we turn our focus to the final protocol, where we show how the beacon can be implemented.

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 $\mathcal{F}_{\mathsf{DLS}}$ to perform the leader selection process. In this section, we show how to remove the dependency to $\mathcal{F}_{\mathsf{DLS}}$ through a Protocol π_{DLS}, which allows the stakeholders to compute the randomness and auxiliary information necessary in the leader election. The resulting modified protocol that we will denote $\Pi[\pi_{\mathrm{DPoS}}, \pi_{\mathrm{DLS}}]$ operates in the $\mathcal{F}_{\mathsf{LS}}$ hybrid world.

Recall that the only essential difference between $\mathcal{F}_{\mathsf{LS}}$ and $\mathcal{F}_{\mathsf{DLS}}$ 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."

Protocol π_{DLS} is described in Figure 12 and uses a publicly verifiable secret sharing (PVSS) [44]. The resulting combined protocol $\Pi[\pi_{\mathrm{DPoS}}, \pi_{\mathrm{DLS}}]$ operates as π_{DPoS} while it runs π_{DLS} to support the random beacon generation for each epoch. For simplicity we will describe the combined protocol in the random oracle model, i.e., with access to a random oracle functionality $\mathcal{F}_{\mathsf{RO}}$; nevertheless, it is also possible to achieve alternative realisations that do not depend on the random oracle abstraction. We will take advantage of a simulation argument that will exploit the fact that the PVSS scheme and the coin-flipping protocol built on top of it simulates a perfect beacon with negligible distinguishing advantage. Simulation here suggests that, in the case of honest majority, there is a simulator that interacts with the adversary and produces indistinguishable protocol transcripts when given the beacon value after the commitment stage. While describing such a simulator is outside the scope of our exposition, such a simulator can be inferred from the PVSS schemes of [44] or [18] realized in the random oracle model where one can take advantage of programmability of the oracle. We note that a random oracle is by no means necessary and it is possible to derive a simulator by taking advantage of a "common reference string" (CRS) embedded into the genesis block.

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 [11], 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 $\mathsf{Open}(r, u_1,)$ 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.

Publicly 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 [26], 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 $(\mathsf{Deal}, \mathsf{Rec})$. We are only interested in VSS where the secret is a random string. The dealing algorithm $\mathsf{Deal}(t, n)$ takes as input the number of shares to be generated n as well as a parameter $t \leq n$ and outputs shares $\sigma_1, \ldots, \sigma_n$ of a random value σ. 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). A publicly verifiable secret sharing scheme allows any third party to verify the validity of the shares in a non-interactive way without learning the shares themselves. This is achieved by publishing auxiliary verification information that can later be attested to correspond to the shares when those are revealed. Specifically, there is a Setup algorithm which is assumed to be executed honestly, a key generation algorithm KeyGen that is executed by each shareholder resulting in a secret and a public-key, as well as having an encryption for each σ_i, denoted by β_i, from which σ_i is recoverable via the secret-key of the i-th stakeholder by running algorithm $\mathsf{Decrypt}$. Finally, a Verify algorithm is capable of verifying all encrypted shares using the public-key and setup information. We refer to the discrete logarithm based PVSS of [44] or [18] for more details.

Constructing Protocol π_{DLS}. The main problem to be solved when realizing $\mathcal{F}_{\mathsf{DLS}}$ 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 and it employs two parameters k, ℓ. The stages of the protocol are presented in Figure 11. The *Commitment Phase* proceeds as follows: for $1 \leq i \leq R$, stakeholder U_i engages in PVSS with $\mathsf{Deal}(R, R/2)$.

After $2(k + \ell)$ slots, players remove k blocks from their blockchain and identify if PVSS was executed properly from at least $R/2+1$ of the selected stakeholders; this requires running the PVSS verification for each commitment. Assuming this is the case, the reveal stage commences. In the reveal stage, which lasts for ℓ slots, for $1 \leq i \leq R$, stakeholder U_i reveals the share it received for each commitment, cf., Protocol π_{DLS} in Figure 12. We remark that it is possible, at the expense of

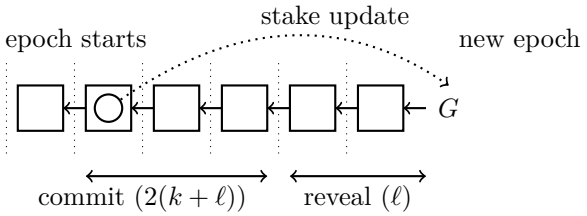


Figure 11: The two stages of the protocol π_{DPoS} that use the blockchain as a broadcast channel in an epoch of $R \geq 2k + 3\ell$ slots.

expanding the length of an epoch somewhat more, to run a more efficient opening stage first and then "force-open" only those commitments that were kept private. Given that, optimistically, no force-open will be required the overall communication complexity in this case would improve.

As a prelude to the next section, we will show how π_{DLS} protocol implements the $\mathcal{F}_{\mathsf{DLS}}$ functionality based on the $\mathcal{F}_{\mathsf{LS}}$ functionality and a "bulletin board" $\mathcal{F}_{\mathsf{BB}}$. Specifically, the functionality operates as follows: it is parameterised by k, ℓ and a stakeholder distribution $\mathbb{S}_0$. Whenever one of the stakeholders submits a message m this is passed to the adversary and after ℓ slots it is reported as pending to any party querying the bulletin board. Finally after $2(k+\ell)$ slots, all pending messages submitted before $2(k + \ell)$ slots are finalized in some order selected by the adversary and are reported to any party querying the functionality. Below we use $\mathcal{F}_{\mathsf{BB}}$ as a global functionality, cf. [17].

Proposition 5.4 (essentially from [44])**.** *For each PPT $\mathcal{A}, \mathcal{Z}$ it holds that there is a PPT $\mathcal{S}$ so that*

$$\mathsf{EXEC}^{\mathcal{F}_{\mathsf{LS}},\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{BB}}}_{\pi_{\text{DLS}},\mathcal{A},\mathcal{Z}}(\lambda) \quad \textit{and} \quad \mathsf{EXEC}^{\mathcal{F}_{\mathsf{DLS}},\mathcal{F}_{\mathsf{BB}}}_{\mathcal{S},\mathcal{Z}}(\lambda)$$

are computationally indistinguishable under the Computational Diffie Hellman assumption where the lookahead parameter of $\mathcal{F}_{\mathsf{DLS}}$ is set to $E = \ell$.

Proof. Fixing any $\mathcal{A}$, the simulator $\mathcal{S}$ runs $\mathcal{A}$ and controls the honest parties' postings with respect to the underlying π_{DLS} protocol following the round structure of the protocol. We recall here the structure of the PVSS protocol of [44]. There are public parameters g, G and each party has a registered public-key of the form $y_i = G^{x_i}$. To make a commitment each party picks a random polynomial $p(\cdot)$ of degree $d = R/2$ with coefficients $a_0, a_1, \ldots, a_d$. It performs a commitment by publishing the values $C_0 = g^{a_0}, \ldots, C_d = g^{a_d}$ as well as $Y_1 = y_1^{p(1)}, \ldots, Y_R = y_R^{p(R)}$. Let $X_i = \prod_{j=0}^{d} C_j^{i^j} = g^{p(i)}$. Using a proof of equality of discrete logarithms the party also publishes for each i, a proof that $\log_g(X_i) = \log_{y_i}(Y_i)$. A commitment is verified by checking all proofs of discrete-log equality. Note that a commitment may be opened by publishing $S = G^{a_0}$ together with a proof of equality of discrete logarithms $\log_G(S) = \log_g(C_0)$. The hash of the product of opened commitments is the value of the next beacon. The parties "force" open the commitments of each other by publishing the values: (i, S_i) with $S_i = Y_i^{x_i^{-1}}$) together with a proof of equality of discrete-logarithms $\log_{S_i}(Y_i) = \log_{Y_i}(G)$. Subsequently it is possible to recover each random value by computing the Lagrange coefficients $\lambda_1, \ldots, \lambda_d$ with respect to the indexes that are available, say $i_1, \ldots, i_d$ and compute $S = \prod_{j=1}^{d} S_{i_j}^{\lambda_j}$.

We next describe a simulated execution of the protocol that is based on a DDH tuple $\langle g, G, C, D\rangle$. During the initial commit stage, $\mathcal{S}$ follows the protocol of each honest party modifying the commitment C_0 to be computed as C^{a_0} as opposed to g^{a_0}. For simplicity, assume that the adversarial

parties are $1, \ldots, R/2$. In the simulated execution, the public-keys $y_{R/2+1}, \ldots, y_R$ are selected as powers of g. The polynomial values $p(1), \ldots, p(R/2)$ are selected on behalf of the honest party at random. Subsequently, the share encryptions $y_1^{p(1)}, \ldots, y_{R/2}^{p(R/2)}$ are calculated. Note that the values $p(R/2+1), \ldots, p(R)$ cannot be evaluated directly since the value $p(0)$ is determined by $\log_g(C_0) = a_0 \cdot \log_g C$ which is unknown to the simulator. However it is possible for $\mathcal{S}$ to calculate

$$Y_{R/2+1} = y_{R/2+1}^{p(R/2+1)}, \ldots, Y_R = y_R^{p(R)}$$

by performing Lagrange interpolation and taking advantage of the fact that $\log_g(y_i)$ for $i \in \{R/2+1, \ldots, R\}$ is known. Specifically, the simulator can calculate $g^{p(R/2)+1}, \ldots, g^{p(R)}$ by a suitable Lagrange interpolation over the bases: $g^{p(1)}, \ldots, g^{p(R/2)}, C^{a_0} = g^{p(0)}$ and then, by raising them to the corresponding honest party secret-keys, the simulator obtains $Y_{R/2+1}, \ldots, Y_R$. In a similar manner the values $C_1 = g^{a_1}, \ldots, C_d = g^{a_d}$ can be calculated by Lagrange interpolation with the bases $g^{p(1)}, \ldots, g^{p(R/2)}, C^{a_0} = g^{p(0)}$. All proofs performed by $\mathcal{S}$ on behalf of the honest party will be simulated taking advantage of random oracle programmability.

Observe that the PVSS opening for the honest party is equal to D^{a_0}. $\mathcal{S}$ follows the above strategy for all honest parties while it receives all the PVSS values contributed by the malicious party. At the conclusion of the $2(k+\ell)$ round, $\mathcal{S}$ queries $\mathcal{F}_{\mathsf{DLS}}$ to obtain the beacon value of the next epoch. At this moment $\mathcal{S}$ is ready to program the random oracle so that the next epoch is fixed to the correct beacon value; we call this the critical moment of the simulation. Note that programmability will be achievable provided that the value D^{a_0} for a PVSS of an honest party can be set to a desired random value; if this value is already defined then $\mathcal{S}$ fails. Assuming the failure event does not happen, given that $\mathcal{S}$ controls a majority of parties, it is capable of extracting the PVSS value contributed by each malicious party; note that this requires the proper secret-key of the honest party, which in turn requires $\log_g(G)$.

Next we argue that the failure event happens with negligible probability. For the sake of contradiction suppose that it happens with some probability α which is non-negligible. We perform the simulation as above setting the values $\langle g, G, C \rangle$ to a given instance of the CDH problem for a random honest party. Subsequently at the critical moment of the simulation we terminate returning as output the value X^{1/a_0} where X is a random input value in the random oracle table. Given that the failure event happens with probability α, we conclude that this algorithm breaks CDH with probability $\alpha/(Rq)$ where is q is the number of queries posed to the random oracle. □

5.3 Robust Transaction Ledger

The key idea is to combine the π_{DLS} protocol and its ability to simulate $\mathcal{F}_{\mathsf{DLS}}$ as shown in Proposition 5.4 with the recursive argument of Theorem 5.3 observing that the blockchain itself can play the role of the $\mathcal{F}_{\mathsf{BB}}$ that π_{DLS} requires.

Theorem 5.5. *Fix parameters k, ℓ, E, ϵ, σ, R, and L, where $R \geq 2k + 3\ell$ and L is an integer multiple of R. Consider an execution $\mathcal{E}$ of $\Pi[\pi_{\mathrm{DPoS}}, \pi_{\mathrm{DLS}}]$ with lifetime L coupled with a $(1/2 - \epsilon)$-initially-bounded adversary $\mathcal{A}$ with corruption delay $D = 2R$ and environment $\mathcal{Z}$ exhibiting a stake-shift of σ over ℓ slots. Then persistence, with parameter k, and liveness, with parameter $u = 2(k+\ell)$, are violated with probability no more than*

$$\varepsilon := L \cdot \left[\exp(-\Omega(\sqrt{k})) + O(\exp(-2\epsilon^2 \ell))\right] + \epsilon_{\mathsf{H}} + (L/R)\epsilon_{\mathrm{DLS}} + \epsilon_{\mathrm{DSIG}}\,.$$

Protocol π_{DLS} with parameter k, ℓ.

π_{DLS} is a protocol run by a subset of elected stakeholders each one corresponding to a slot during an epoch e_j that lasts $R \geq 2k + 3\ell$ slots. The stakeholders without loss of generality are denoted by $U_1, \ldots, U_R$ and they are not necessarily distinct.

Precondition. We assume that the Setup algorithm for the PVSS has been executed and its output is posted in $\mathcal{F}_{\mathsf{BB}}$ as well as each stakeholder U_i has performed KeyGen to obtain y_i, s_i and posted the public-key y_i to $\mathcal{F}_{\mathsf{BB}}$. (If some of the stakeholders' keys are missing the same protocol is executed with R adjusted accordingly).

1. **Commitment Phase** ($2(k + \ell)$ slots) At slot $R - 2(k + \ell) - \ell$ of epoch e_j, for $1 \leq i \leq R$, stakeholder U_i performs $\mathsf{Deal}(R, R/2, y_1, \ldots, y_R)$ to obtain the encryptions $\beta_1, \ldots, \beta_n$ and posts them to $\mathcal{F}_{\mathsf{BB}}$.
2. **Reveal Phase** (ℓ slots) At slot $R-\ell$ of epoch e_j, for $1 \leq i \leq R$, U_i runs $\mathsf{Decrypt}(s_i, \beta_{li})$ to recover the encrypted shares σ_{li} for any U_l that successfully posted in $\mathcal{F}_{\mathsf{BB}}$ R encrypted shares that all verify correctly (which is tested using Verify). Subsequently it posts σ_{li} to the $\mathcal{F}_{\mathsf{BB}}$.

The implementation of $\mathsf{epochrnd\_req}$ is then as follows.

- Given input $(\mathsf{epochrnd\_req}, U_i, e_j)$, the values ρ_l^j are calculated to be equal to $H(\mathsf{Rec}(\sigma_{l1}, \ldots, \sigma_{lR}))$ assuming that U_l posted to $\mathcal{F}_{\mathsf{BB}}$ all shares correctly (otherwise ρ_l^j is set to 0). Finally the epoch randomness is set to $\rho^j = \oplus_{l=1}^{R} \rho_l^j$. The responce is then set to $(\mathsf{epochrnd}, \rho^j)$.

Figure 12: Protocol π_{DLS} using $\mathcal{F}_{\mathsf{LS}}, \mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{RO}}$ and an underlying PVSS scheme.

Here ϵ_{H} denotes the probability of a collision occurring among the queries to H (including those of $\mathcal{A}$), ϵ_{DSIG} is the distinguishing advantage of the digital signature implementation, ϵ_{DLS} is the distance of the DSIG implementation. The above probabilities are in the conditional space $\neg\mathsf{Bad}^{1/2-\epsilon-\sigma}$.

Proof. Fix some environment $\mathcal{Z}$ and adversary $\mathcal{A}$. As before we first consider the execution E of $\Pi[\pi_{\mathrm{iDPoS}}, \pi_{\mathrm{DLS}}]$ first. We consider now a modified execution E^* where the execution of the π_{DLS} protocol is substituted by the simulation that is provided due to Proposition 5.4. In this modified execution E^* we define the same set of random variables Chain_t^* and $\mathcal{C}^{*(t)}, \mathsf{Hon}_t^*$ for each epoch t as in the proof of Theorem 5.3. The stakeholder distribution for each beacon simulation that samples the beacon for the next epoch is drawn from the immutable chain $\mathcal{C}^{*(t)}$.

The simulation might stop for two reasons. First it might be the case that the chain $\mathcal{C}^{*(t)} = \perp$, i.e., the immutable chain of the stakeholders is not defined and hence no stakeholder distribution can be determined to execute the beacon protocol. The second case where it might fail is because of a dishonest majority among the R selected stakeholders, i.e., $\mathsf{Hon}_t = 0$.

We now observe that in the modified execution E^* the proof of Theorem 5.3 can be repeated identically and we can infer that the probability of B^*, the event that persistence or liveness is violated or the simulation fails in E^*, is at most

$$L \cdot \left[\exp(-\Omega(\sqrt{k})) + O(\exp(-2\epsilon^2\ell))\right] + \epsilon_{\mathsf{H}}\,.$$

Now consider the event B that either persistence or liveness is violated in an execution E of the protocol $\Pi[\pi_{\mathrm{iDPoS}}, \pi_{\mathrm{DLS}}]$ and let $\tilde{t}$ be the index of the earliest epoch that this takes place (or $\tilde{t} = \infty$ if the event never happens). We define the intermediate hybrid distributions between E and E^* denoted by $E^{(t)}$ for which it holds that the beacon generation in the first t epochs is performed in simulation as in E^* while the remaining are performed as in E by performing π_{DLS}. Observe that $E^{(0)} = E$ and $E^{(L/R)} = E^*$. Furthermore, conditional on $1 \leq t < \tilde{t}$, it holds that the computational distance between $E^{(t-1)}$ and $E^{(t)}$ is at most ϵ_{DLS}. It follows that the probability of B is bounded

by the probability of B^* plus $(L/R) \cdot \epsilon_{\mathrm{DLS}}$. As a result, persistence and liveness are violated in E with probability at most

$$\left[\exp(-\Omega(\sqrt{k})) + O(\exp(-2\epsilon^2\ell))\right] + \epsilon_{\mathsf{H}} + (L/R) \cdot \epsilon_{\mathrm{DLS}}\,.$$

Armed with this result, and using proposition 5.1 we can infer that there is a simulator $\mathcal{S}_1$ that idealises the digital signature used in the protocol and acts as the attacker in an execution E.

$$\left|\Pr\left[\mathsf{EXEC}^{\mathcal{F}_{\mathsf{LS}}[\mathsf{SIG}],\mathcal{F}_{\mathsf{RO}}}_{\Pi[\pi_{\mathrm{DPoS}},\pi_{\mathrm{DLS}}],\mathcal{A},\mathcal{Z}}(\lambda)\right] - \Pr\left[\mathsf{EXEC}^{\mathcal{F}_{\mathsf{LS}}[\mathcal{F}_{\mathsf{DSIG}}],\mathcal{F}_{\mathsf{RO}}}_{\Pi[\pi_{\mathrm{iDPoS}},\pi_{\mathrm{DLS}}],\mathcal{S}_1,\mathcal{Z}}(\lambda)\right]\right| \leq \epsilon_{\mathrm{DSIG}}\,.$$

From the above it follows immediately that persistence and liveness are violated in an execution of $\Pi[\pi_{\mathrm{DPoS}}, \pi_{\mathrm{DLS}}]$ with the probability stated in the theorem. □

Remark 2. *We note that it is easy to extend the adversarial model to include fail-stop (and recover) corruptions in addition to Byzantine corruptions. The advantage of this mixed corruption setting, is that it is feasible to prove that we can tolerate a large number of fail-stop corruptions (arbitrarily above* 50%*). The intuition behind this is simple: the forkable string analysis still applies even if an arbitrary percentage of slot leaders is rendered inactive. The only necessary provision for this would be expand the parameter k inverse proportionally to the rate of non-stopped parties; if the rate is constant asymptotically the same analysis would apply. For a further investigation in this direction see our follow up work, Ouroboros Genesis [4].*

6 Covert Adversaries

The general notion of fork defined in Definition 4.11 above reflects the possibility that adversarial slot leaders may broadcast multiple blocks for a single slot; such adversaries may simultaneously extend many different chains. While this provides the adversary significant opportunities to interfere with the protocol, it leaves a suspicious "audit trail"—multiple signed blocks for the same slot—which conspicuously deviates from the protocol.

This motivates our consideration of a restricted class of *covert* adversaries, who broadcast no more than one block per slot. Such an adversary may still deviate from the protocol by extending short chains, but does not produce such suspicious evidence and hence its strategy is more "deniable": it can blame network delays for its actions.[6]

6.1 Covert forks, and covertly forkable strings

Such an adversary yields a restricted notion of fork, defined below:

Definition 6.1. *Let $F \vdash w$ be a fork for a string $w \in \{0,1\}^*$. We say that F is* covert *if the labeling $\ell : V \to \{0, 1, \ldots, \}$ is injective. In particular, no adversarial index is labeled by more than one node.*

As in the general case, we define a notion of forkable string for such adversaries.

Definition 6.2. *We say that a string w is* covertly forkable *if there is a flat covert fork $F \vdash w$.*

Covert adversaries and forks have much simpler structure than general adversaries. In particular, a string is covertly forkable if and only if a majority of its indices are adversarial. This provides an analogue of Proposition 4.27 for covertly forkable strings.

[6] Contrast this with a more general adversary that attempts to fork by signing two different blocks for the same slot; such an adversary cannot merely blame the network for such a deviation.

Proposition 6.3. *A string $w \in \{0,1\}^n$ is covertly forkable if and only if* $\mathrm{wt}(w) \geq n/2$.

Proof. Let w be a covertly forkable string and $F \vdash w$ a flat covert fork. As F is flat, there are two edge disjoint tines, t_1 and t_2, with length equal to height(F) and it follows that the number of vertices in F is at least $2 \cdot \mathrm{height}(F) + 1$. In this covert case the labeling function is injective, and it follows that $n \geq 2 \cdot \mathrm{height}(F)$. (Recall that the root vertex is labeled by 0, which is not an index into w.) On the other hand, the height of F is at least the number of honest indices of w. We conclude that the length of w is at least twice the number of honest indices, as desired.

If $\mathrm{wt}(w) \geq n/2$, we can produce a flat covert fork $F \vdash w$ by placing all honest indices on a common tine t_1 and selecting length(t_1) adversarial indices to form an edge-disjoint second tine t_2. □

As the structure of covertly forkable strings is so simple, an analogue of Theorem 4.24 for the density of covertly forkable strings follows directly from standard large deviation bounds.

Theorem 6.4. *Let $\epsilon \in (0,1)$ and let w be a string drawn from $\{0,1\}^n$ by independently assigning each $w_i = 1$ with probability $1/2 - \epsilon$. Then*

$$\Pr[w \text{ is covertly forkable}] = 2^{-\Theta(\epsilon^2 n)}.$$

Proof. This follows from standard estimates for the cumulative density function of the binomial distribution. □

Exact probabilities of covert forkability for explicit values of n. For comparison with the general case, we computed the probability that a string drawn from the binomial distribution is covertly forkable. These results are presented in Figure 13. (Note that these probabilities are simply appropriate evaluations of the cumulative density function of the binomial distribution.) Analogous results for the general case appeared in Figure 8.

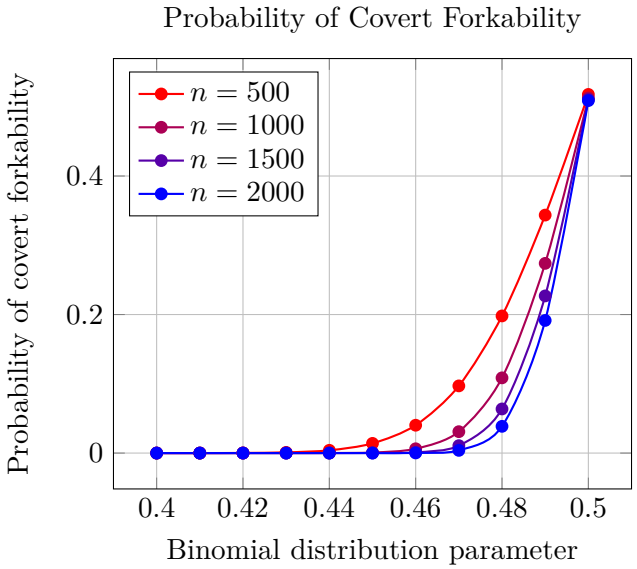


Figure 13: Graphs of the probability that a string drawn from the binomial distribution is covertly forkable. Graphs for string lengths $n = 500, 1000, 1500, 2000$ are shown with parameters $.40, .41, \ldots, .49, .50$.

6.2 Common prefix with covert adversaries

We revisit the notion of common prefix in the setting of covert adversaries. We define the *covert divergence* of w to be the maximum divergence over all possible covert forks for w:

$$\operatorname{cdiv}(w) = \max_{\substack{F \vdash w \\ F \text{ covert}}} \operatorname{div}(F)\,.$$

As in the setting with general adversaries, we wish to establish that a string with large covert divergence must have a large covertly forkable substring. A direct analogue of Theorem 4.31 then implies that characteristic strings arising from π_{iSPoS} are unlikely to have large covert divergence and, hence, possess the common prefix property against covert adversaries.

We record an analogue of Theorem 4.30 for covert adversaries.

Theorem 6.5. *Let $w \in \{0,1\}^*$. Then there is a covertly forkable substring $\breve{w}$ of w with $|\breve{w}| \geq \operatorname{cdiv}(w)$.*

Proof. We are more brief, as portions of the proof have direct analogs in the proof of Theorem 4.30. Consider a covert fork $F \vdash w$ and a pair of viable tines (t_1, t_2) of F for which $\ell(t_1) \leq \ell(t_2)$ and $t_1/t_2 = \operatorname{cdiv}(w)$; we assume, as in the proof of the general case, that this pair of tines minimizes the quantity $|\ell(t_2) - \ell(t_1)|$ among all pairs with divergence equal to $\operatorname{cdiv}(w)$.

Let y denote the last vertex on the tine $t_1 \cap t_2$. In contrast to the setting with a general adversary, it is not clear that y is honest and this motivates a slightly different choice for the beginning of the string $\breve{w}$: define α to be the largest honest index of w on the tine $t_1 \cap t_2$, with the convention that $\alpha = 0$ if there is no such index. As in the proof of Theorem 4.30, define β to be the smallest honest index of w for which $\beta \geq \ell(t_2)$, with the convention that $\beta = n + 1$ if there is no such honest index. Then define $\breve{w} = w_{\alpha+1} \dots w_{\beta-1}$; as in the proof of Theorem 4.30 it is easy to confirm that $|\breve{w}| = (\beta - 1) - \alpha \geq \ell(t_1) - \ell(t_1 \cap t_2) \geq \operatorname{cdiv}(w)$. The remainder of the proof argues that $\breve{w}$ is covertly forkable.

As in the proof of Theorem 4.30, the depth $\mathbf{d}(h)$ of any honest index $h < \beta$ is no more than $\min(\operatorname{length}(t_1), \operatorname{length}(t_2))$: if $h \leq \ell(t_1)$ this follows directly from the definition of viability. Otherwise, $\ell(t_1) < h < \ell(t_2)$ and we consider the tine t_h labeled with h: if $\operatorname{length}(t_h) \geq \min(\operatorname{length}(t_1), \operatorname{length}(t_2))$ then the tine t_h, coupled with either t_1 or t_2, would produce a pair of tines with divergence no less than $\operatorname{div}(t_1, t_2)$, but for which $|\ell(\cdot) - \ell(\cdot)|$ is strictly less than $|\ell(t_1) - \ell(t_2)|$.

To complete the proof, we define an injective function $i : H \to A$, where H denotes the set of honest indices in $\{\alpha+1, \dots, \beta-1\}$ and A the complement—the set of adversarial indices of $\breve{w}$. The existence of such a function implies that $|H| \leq |A|$ and hence that $\breve{w}$ is covertly forkable by the criterion given in Proposition 6.3. Let $A' \subset A$ denote the set of adversarial indices of $\breve{w}$ appearing as a label on either of the two tines t_1 and t_2. The function i is defined as follows: $i(h)$, for an honest index $h \in H$, is the smallest (adversarial) index of A' which labels a vertex at depth equal to $\mathbf{d}(h)$. Assuming that this function is well-defined it is clearly injective, as labels cannot appear on multiple vertices of a covert fork and depths of honest vertices are pairwise distinct.

To confirm that $i(h)$ is well-defined, note that for any $h \in H$ we must have $\mathbf{d}(\alpha) < \mathbf{d}(h) \leq \min(\operatorname{length}(t_1), \operatorname{length}(t_2))$ and hence there is at least one vertex v on each of t_1 and t_2 with depth equal to $\mathbf{d}(h)$; furthermore, by the defining properties of α and β, this vertex is labeled with an index of $\breve{w}$. If $\mathbf{d}(h) \leq \operatorname{length}(t_1 \cap t_2)$, there is a common vertex v on these tines for which $\operatorname{length}(v) = \mathbf{d}(h)$; note that this vertex cannot be honest by the definition of α, so $i(h) = \ell(v)$ is well-defined in this case. If $\mathbf{d}(h) > \operatorname{length}(t_1 \cap t_2)$, the two tines have distinct vertices at depth $\mathbf{d}(h)$, and one of these must then be adversarial—thus $i(h)$ is well-defined in this case as well. □

Finally, we remark that the proof of Theorem 4.31 applies with minor adaptations to the covert case.

Theorem 6.6. *Let $k, R \in \mathbb{N}$ and $\epsilon \in (0, 1/2)$. Let $w = w_1 \dots w_R$ be drawn in $\{0,1\}^R$ so that each w_i independently takes the value 1 with probability $1/2 - \epsilon$. Then*

$$\Pr[\mathrm{cdiv}(w) \geq k] \leq R \exp(-\Omega(\epsilon^2 k)) .$$

This directly bounds the probability that a covert adversay can effect a common prefix violation in an execution of π_{iSPoS}.

Proof. The proof of Theorem 4.31 applies directly; in this case the asymptotics rely on Theorem 6.4 and the fact that

$$\sum_{t=k}^{\infty} e^{-ct} \leq \int_{k-1}^{\infty} e^{-ct}\, dt = O(1) \cdot e^{-ck} = e^{-ck+O(1)} ,$$

where the hidden constant may depend on c, but not k. □

7 Anonymous Communication and Stronger Adversaries

The protocols constructed in the previous section are proven secure against *delayed* adaptive corruptions, meaning that, after requesting to corrupt a given party U_i, the adversary has to wait for D slots before the corruption actually happens. Naturally, it is desirable to make D as small as possible, or even eliminate it altogether to achieve security against a standard adaptive adversary.

The delay is required because the adversary must not be able to corrupt parties in direct response to knowledge of the leader election schedule. (Recall that the protocol determines this schedule on an epoch-by-epoch basis, so that such an attack would be particularly devastating.) However, notice that the slot leaders are selected by weighting public keys by stake, while the adversary can only choose to corrupt a user U_i without knowing its public key. Thus, the adversary must be able to observe communication between U_i and the Diffuse functionality in order to determine which public key is associated with user U_i and detect when U_i is selected as a slot leader. We will show that we can eliminate the delay by extending our model with a *sender anonymous broadcast channel* (provided by the Diffuse functionality) and having the environment activate all parties in every round. We introduce the following modifications in the ideal functionalities:

- Diffuse Functionality: The functionality will work as described in Section 2 except that it will remove all information about the sender U_s of every message before delivering it to the receiver U_r's inbox (input tape), thus ensuring that the sender remain anonymous.[7]
- Key and Transaction Functionality: The functionality will work as described in Section 2 except that it will allow immediate corruption of a user U upon receiving a message $(\mathsf{Corrupt}, U)$ from the adversary.

Apart from these modifications in the ideal functionalities, we also change the environment behavior by requiring that it activates *all* users at every slot sl_j. Having all parties being activated at every slot results in an anonymity set of size equal to the number of honest parties, making it difficult for the adversary to associate a given public key with a user (i.e., any of the honest parties could be associated with a given public key that is not associated with a corrupted party). In this extended model we can reprove Theorem 5.5 without a delay D by strengthening the restrictions that are imposed on the environment in the following way.

[7] In practice, a sender anonymous broadcast channel with properties akin to those of the Diffuse functionality can be implemented by Mix-networks [19] or DC-networks [20] that can be executed by the nodes running the protocol.

- We will say the adversary is *restricted to less than* $1/2 - \delta$ *relative stake for windows of length* D if for all sets of consecutive slots of length D, the sum over all corrupted keys of the maximum stake held by each key during this period of D slots (in any possible $\mathbb{S}_j(r)$ where U_j is an honest party) is no more than $1/2 - \delta$ of the minimum total stake during this period. In case the above is violated an event $\mathsf{Bad}_D^{1/2-\delta}$ becomes true for the given execution.

Using the above strengthened condition, we can remove the corruption delay requirement D in Theorem 5.5 by assuming that $\mathsf{Bad}^{1/2-\delta}$ is substituted with $\mathsf{Bad}_D^{1/2-\delta}$.

8 Incentives

So far our analysis has focused on the cryptographic setting where a set of honest players operate in the presence of an adversary who may corrupt some of the players. In this section we consider the setting of a coalition of rational players and their incentives to deviate from honest protocol operation.

8.1 Input Endorsers

In order to address incentives, we modify further our basic protocol to assign two different roles to stakeholders. As before, in each epoch there is a set of elected stakeholders that are the slot leaders of the epoch responsible for issuing blocks and forming the randomness of the next 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 next slot. (We remark that one can adapt this discussion to a setting with multiple endorsers; however, we assume a single input endorser per slot in this description.) While this seems like an insignificant modification it gives us a room for improvement for the following reason: endorsers' contributions will be acceptable even if they are d slots late, where $d \in \mathbb{N}$ is a parameter of the protocol. (In particular, the protocol calls for slot leaders to include in their block any inputs endorsed by the previous d endorsers and not appearing in the existing chain.) Note that blocks and endorsed inputs are diffused independently with each block containing from 0 up to d endorsed inputs.

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 stake weight and are thus a representative sample of the stakeholder population. In the case of a transaction ledger, the same transaction may 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 ledger. 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 $d \geq \ell$ (the $\exists\mathsf{CQ}$ parameter from previous sections), any set of stakeholders S will be an endorser in a subset of the d slots with probability proportional to its cumulative stake; the result follows.

As in bitcoin, stakeholders 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 stakeholders to invest computational resources to issue blocks. Rather, *availability* and *transaction verification* should be incentivized. Nevertheless, they have to be incentivized to be online often. Any stakeholder, at minimum, must be online and operational 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 as well as any endorsed inputs to include in the block.
- In the slot during which she is the elected shareholder so that she issues the block containing the endorsed inputs.
- In a slot during the commit stage of an epoch where she is supposed to issue the PVSS 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.
- 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) that requires processing all available transactions and verifying them.

In order to incentivize the above actions in the setting of a transaction ledger, fees can be 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 block dependent as advocated in [38]. In our setting, it is possible to collect all fees of transactions included in a sequence of blocks in a pool and then distribute that pool to all shareholders that participated during these slots. For example, all input endorsers that were active may receive reward proportional to the number of inputs they endorsed during a period of 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).

The reward mechanism that we will pair with input endorsers operates as follows. Let $\mathcal{C}$ be a chain consisting of blocks $B_0, B_1, \ldots$. Consider the sequence of blocks that cover the j-th epoch denoted by $B_1, \ldots, B_s$ with timestamps in $\{jR+1, \ldots, (j+1)R\}$ that contain an $r \geq 0$ sequence of endorsed inputs that originate from the j-th epoch (some of them may be included as part of the $j+1$ epoch). We define the reward pool $P_{\mathsf{all}}^{(j)}$ to be equal to the sum of the transaction fees that are included in the endorsed inputs that correspond to the j-th epoch. 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 at that position) in the calculation of $P_{\mathsf{all}}^{(j)}$, where the total order used is induced by the order the endorsed inputs that are included in $\mathcal{C}$. In the sequence of these blocks, we identify by $L_1, \ldots, L_R$ the slot leaders corresponding to the slots of the epoch and by $E_1, \ldots, E_r$ the input endorsers that

actively contributed the sequence of r endorsed inputs. Subsequently, the i-th stakeholder U_i can claim a reward up to the amount

$$\left(\beta \cdot \frac{|\{j \mid U_i = E_j\}|}{r} + (1-\beta) \cdot \frac{|\{j \mid U_i = L_j\}|}{R}\right) \cdot P_{\mathsf{all}}^{(j)},$$

where $\beta \in [0,1]$ is a parameter of the protocol. Claiming a reward is performed by issuing a "coinbase" type of transaction at any point after $2(k+\ell)$ slots in a subsequent epoch to the one that a reward is being claimed from.

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; (iii.) it rewards entities for epoch j after slot $jR + 2(k+\ell)$.

We proceed to show that our system is a δ-Nash (approximate) equilibrium, cf. [35, Section 2.6.6]. Specifically, the theorem states that any coalition deviating from the protocol can add at most an additive δ to its total rewards.

A technical difficulty in the above formulation is that the number of players, their relative stake, as well as the rewards they receive are based on the transactions that are generated in the course of the protocol execution itself. To simplify the analysis we will consider a setting where the number of players is static, the stake they possess does not shift over time and the protocol has negligible cost to be executed. We observe that the total rewards (and hence also utility by our assumption on protocol costs) that any coalition V of honest players are able to extract from the execution lasting $L = tR + 2(k+\ell) + 1$ slots, is equal to

$$\mathcal{R}_V(\mathcal{E}) = \sum_{j=1}^{t} P_{\mathsf{all}}^{(j)} \left(\beta \frac{IE_V^j(\mathcal{E})}{R} + (1-\beta)\frac{SL_V^j(\mathcal{E})}{r_j}\right)$$

provided that $\mathcal{E}$ satisfies CP with parameter k, $\exists\mathsf{CQ}$ is satisfied with parameter ℓ, and HCG is satisfied with parameters $(1/2, 2k)$), and where r_j is the total endorsed inputs emitted in the j-th epoch (and possibly included at any time up to the first ℓ slots of epoch $j+1$), $P_{\mathsf{all}}^{(j)}$ is the reward pool of epoch j, $SL_V^j(\mathcal{E})$ is the number of times a member of V was elected to be a slot leader in epoch j and $IE_V^j(\mathcal{E})$ the number of times a member of V was selected to endorse an input in epoch j. We set by convention the value of $\mathcal{R}_V(\mathcal{E})$ to 0 when $\mathcal{E}$ is an execution where the basic underlying properties of the blockchain fail (in particular CP with parameter k, $\exists\mathsf{CQ}$ is satisfied with parameter ℓ, and HCG is satisfied with parameters $(1/2, 2k)$). Finally, observe that the actual rewards obtained by a set of rational players V in an execution $\mathcal{E}$ might be different from $\mathcal{R}_V(\mathcal{E})$; for instance, the coalition of V may never endorse a set of inputs in which case they will obtain a smaller number of rewards.

We will establish the fact that our protocol is a δ-Nash equilibrium by proving that the coalition V, even deviating from the proper protocol behavior, it cannot obtain utility that exceeds $\mathcal{R}_V(\mathcal{E}) + \delta$ for some suitable constant $\delta > 0$.

Theorem 8.1. *Fix any $\delta > 0$ and polynomially related parameters k, ℓ, λ; under the same conditions and restrictions as in Theorem 5.5, the honest strategy in the protocol is a δ-Nash equilibrium against any coalition of players represented as an adversary $\mathcal{A}$, provided that the maximum total rewards P_{all} provided in all possible protocol executions is bounded by a polynomial in λ.*

Proof. Consider a coalition of rational players V—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 V. Observe that based on Theorem 5.5, no matter the strategy of V, with probability $1 - \varepsilon$ for some function ε negligible in the security parameters k, ℓ, λ the protocol will enable all users to obtain the rewards they are entitled to as slot leaders and input endorsers. The latter stems from the following. First, from liveness, at least one honest block will be included every ℓ slots and hence, in each epoch, all input endorsers that follow the protocol will have the opportunity to enter their endorsed inputs as many times they were elected to be. Second, the rewards received will be proportional to the times each party is an input endorser and issued a block successfully as well as equal to the number of times it is a slot leader. As a result, except with error ε, the utility received by coalition V is equal to $\mathcal{R}_V$. It follows that player V has expected utility at most $E[\mathcal{R}_V] + \varepsilon P_{\mathsf{All}}$, where P_{All} is the maximum amount of rewards produced in any possible execution. The result follows by the assumption in the statement of the theorem since $\varepsilon P_{\mathsf{All}} \leq \delta$ for sufficiently large λ. □

Remark 3. *In the above theorem, for simplicity, we assumed that protocol costs are not affecting the final utility (in essence this means that protocol costs are assumed to be negligible). Nevertheless, it is straightforward to extend the proof to cover a setting where a negative term is introduced in the payoff function for each player proportional to the number of times inputs are endorsed and the number of messages transmitted for the beacon protocol. The proof would be resilient to these modifications because endorsed inputs and beacon protocol messages cannot be stifled by the adversary and hence the reward function can be designed with suitable weights for such actions that offsets their cost. Still note that the rewards provided are assumed to be "flat" for both slots and endorsed inputs and thus the costs would also have to be flat. We leave for future work the investigation of a more refined setting where costs and rewards are proportional to the actual computational steps needed to verify transactions and issue blocks.*

Remark 4. *The reward function described, only considers the number of times an entity was an input endorser without considering the amount of work that was put to verify the given transactions. Furthermore it is not sensitive to whether a slot leader issued a block or not in its assigned time slot. We next provide some context behind these choices. First suppose that slot leaders do not receive a reward when they do not issue a block. It is easy to see that when all parties follow the protocol the parties will receive the proportion from the reward pool that is associated to block issuance roughly proportional to their stake. Nevertheless, a malicious coalition can easily increase the ratio of these rewards by performing a block witholding attack (in this case this would amount to a selfish mining attack). Given that this happens with non-negligible probability a straightforward definition of $\mathcal{R}_V(\mathcal{E})$ that respects this assignment is vulnerable to attack and hence a δ-Nash equilibrium theorem cannot be shown. Next, we consider the case of extending the reward function so that input endorsers that are rewarded based on the transactions they verify (as opposed to the flat reward we considered in the above theorem). Special care is necessary to design this function. Indeed the straightforward way to implement it, which is if the first input endorser to verify a transaction that is part of the pool can make a higher claim for its fee, then there is a strategy for an adversary to deviate from the protocol and improve its ratio of rewards: perform block withholding and/or endorsed input censorship to remove endorsed inputs from the blockchain that originate to honest parties. Then include the removed transactions in an endorsed input that will be transmitted in the last possible opportunity. As before, given the attack, the natural way to define $\mathcal{R}_V(\mathcal{E})$ is susceptible to it and hence a δ-Nash equilibrium theorem cannot be shown. A possible direction for ameliorating this problem is to share the transaction fee of a transaction between all the input endorsers that endorsed it. This suggests the following modification to the protocol: whenever you are an input endorser you should attempt to include all transactions that you have collected for a sequence of k*

slots and retransmit your endorsed input in case it is removed from the main chain. We leave the analysis of such class of reward mechanisms for future work.

9 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 as it is capable to force the size of the committee that runs the protocol to be small (it is worth noting that the delegation mechanism is similar to *mining pools* in proof-of-work blockchain protocols).

9.1 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.)

9.2 Delegation Scheme

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* [12].

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 [12]. 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 [12].

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 a certificate revocation list within the blockchain.

9.2.1 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). 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 value is the case when all stake is distributed in T^{-1} delegates each holding T of the stake).

10 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 5.5. 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). See the next section for an experimental discussion about double spending.

Grinding attacks In stake grinding attacks, the adversary tries to influence the slot leader selection process to improve its chances of being selected to generate blocks (which can be used to perform other attacks such as double spending). Basically, when generating a block that is taken as input by the slot leader selection process, the adversary first tests several possible block headers and block contents in order to find the one that gives it the best chance of being selected as a slot leader again in the future. While this attack affects PoS based cryptocurrencies that collect randomness for the slot leader selection process from raw data in the blockchain itself (*i.e.* from block headers and contents), our protocol uses a coin tossing protocol that is proven to generate unbiased uniform randomness as discussed in Section 5.2. We show that an adversary cannot influence the randomness generated in Figure 12, which is guaranteed to be uniformly random, thus guaranteeing that slot leaders are selected with probability proportional to their stake.

Transaction denial (censorship) 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 5.5. 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 and thus we can get security as long as parties with less than 50% of stake get desynchronized. If more than 50% parties get desynchronized our protocol can fail. More general models like partial synchrony [23, 39] are interesting to consider in the PoS design setting.

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.

Bribery Attacks In bribery attacks [13], an adversary deliberately pays miners (through cryptocurrency or fiat money) to work on specific blocks and forks, aiming at generating an arbitrary fork that benefits the adversary (*e.g.* by supporting a double spending attack). Miners of PoW based cryptocurrencies do not have to own any stake in order to mine blocks, which makes this attack strategy feasible. In this setting, if the adversary offers a bribe higher than the reward for correctly generating a block, any rational miner has a clear incentive to accept the bribe and participate in the attack since it increases the miner's financial outcome. However, in our PoS based protocol, malicious slot leaders who agree to deliberately attack the system not only risk to forego any potential profit they would earn from behaving honestly but may also risk to lose equity. Notice that slot leaders must have money invested in the system in order to be able to generate blocks and if an attack against the system is observed this might bring currency value down. Even if the bribe is higher than the reward for correct behavior, the loss from currency devaluation can easily offset any additional profits made by participating in this attack. Hence, bribery attacks may be be less effective against a PoS based consensus protocol than a PoW based one. Currently our rationality model does not formally encompass this attack strategy and investigating its efficacy against PoS based consensus protocols is left as a future work.

Long-range attacks An attacker who wishes to double spend at a later point in time can mount a long-range attack [14] by computing a longer valid chain that starts right after the genesis block where it is the single stakeholder actively participating in the protocol. Even if this attacker owns a small fraction of the total stake, it can locally compute this chain generating only the blocks for slots where it is elected the slot leader and keep generating blocks ahead of current time until its alternative chain has more blocks than the main chain. Now, the attacker can post a transaction to the main chain, wait for it to be confirmed (and for goods to be delivered in exchange for the transaction) and present the longer alternative chain to invalidate its previously confirmed transaction. This attack is ineffective against Ouroboros for two reasons: Protocol π_{DLS} will only output valid leader selection data allowing for the protocol to continue if a majority of the stakeholders participate (or have delegates participate on their behalf) and stakeholders will reject blocks generated for slots that are far ahead of time. Since the alternative chain is generated artificially with blocks and protocol messages generated solely by an attacker who controls a small fraction of the stake, the leader selection data needed to start new epochs will be considered invalid by other nodes. Even if the attacker could find a strategy to generate an alternative chain with valid leader selection data, presenting this chain and its blocks generated at slots that are far ahead of time would not result in a successful attack since those blocks far ahead of time would be rejected by the honest stakeholders and the final alternative chain would be shorter than the main chain.

Nothing at stake attacks The "nothing at stake" problem 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), and our chain selection rule that instructs players to ignore very deep forks that deviate from the block they received the last time they were online. 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). It is remarked in [15] that the "tragedy of commons" might lead stakeholders in some PoS based schemes to adhere to attacks because they do not have the power to deter attacks by themselves and would incur financial losses even if they did not join the attack. This would lead rational stakeholders to accept small bribes in alternative currencies that might at least obtain some financial gain. However, in the incentive structure of Ouroboros, slot leaders and endorsers who could potentially join an attack would receive rewards in both the main and the adversarial chain, resulting in those stakeholders not achieving higher profits by joining the attack.

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 5.5 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 can be seen a special instance of the nothing at stake problem, where the attacker no longer owns any stake in the system and is thus free from any financial losses when conducting the attack.

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.

11 Experimental Results

We have implemented a prototype instantiation of Ouroboros in Haskell as well as in the Rust-based Parity Ethereum client in order to evaluate its concrete performance. More specifically, we have implemented Protocol π_{DPoS} using Protocol π_{DLS} to generate leader selection parameters (i.e., generating fresh randomness for the weighed stake sampling procedure). For this instantiation, we use the PVSS scheme of [44] 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, [10]) 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.

11.1 Transaction Confirmation Time Under Optimal Network Conditions

We first examine the time required for confirming a transaction in a setting where the network is not under substantial load and transactions are processed as they appear.

Adversary	BTC	OB Covert	OB General
0.10	50	3	5
0.15	80	5	8
0.20	110	7	12
0.25	150	11	18
0.30	240	18	31
0.35	410	34	60
0.40	890	78	148
0.45	3400	317	663

Figure 14: Transaction confirmation times in minutes that achieve assurance 99.9% against a hypothetical double spending attack with different levels of adversarial power for Bitcoin and Ouroboros (both covert and general adversaries).

In Fig. 14 we lay out a comparison in terms of transaction confirmation time between Bitcoin and Ouroboros showing how much a verifier has to wait to be sure that the best possible[8] double-spending attack succeeds with probability less than 0.1%. In the case of Bitcoin, we consider a double-spending attacker that commands a certain percentage of total hashing power and wishes to revert a transaction. The attacker attempts to double-spend via a block-witholding attack as described in the same paper (the attacker mines a private fork and releases it when it is long enough). In the case of Ouroboros we consider a double spending attacker that attempts to brute force the space of all possible forks for the current slot leader distribution in a certain segment of the protocol and commands a certain percentage of the total stake. We consider both the covert and the general adversarial setting for Ouroboros.

In all of the scenarios, we measure the number of minutes that one has to wait in order to achieve probability of double spending less than 0.1%. In Fig. 15 we present a graph that illustrates the speedup graphically.

We note that the above measurements compare our Ouroboros implementation with Bitcoin in the way the two systems are parameterized (with 10 minute block production rate for Bitcoin and 20 second slots for Ouroboros, a conservative parameter selection). Exploring alternative

[8]The "best possible" is only in the the case of Ouroboros, for Bitcoin we use the best known attack.

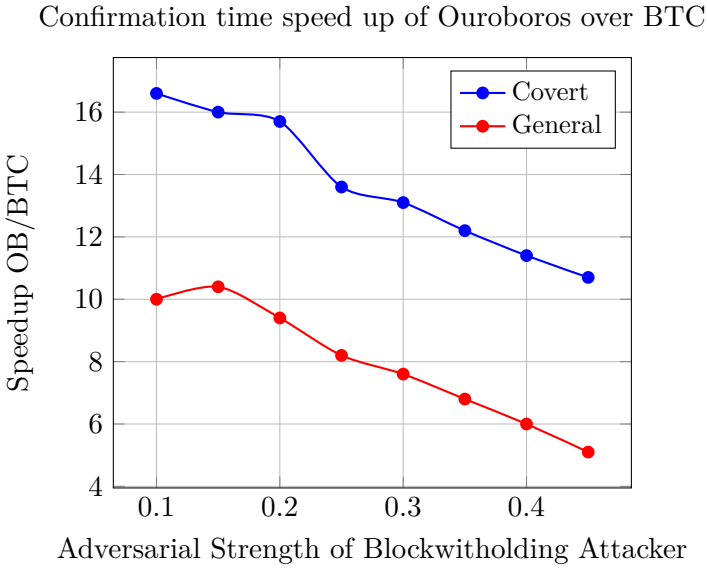


Figure 15: Ouroboros vs. Bitcoin speedup of transaction confirmation time against a hypothetical double spending attacker for assurance level 99.9%. Ouroboros is at least 10 to 5 times faster for regular adversaries and 16 to 10 times faster for covert adversaries.

parameterizations for Bitcoin (such as making the proof-of-work easier) can speed up the transaction processing, nevertheless this cannot be done without carefully measuring the impact on overall security.

11.2 Absolute Performance of Ouroboros

We implemented Ouroboros as an instance of the Rust-based Ethereum Parity client.[9] Subsequently, experiments were run using Amazon's Elastic Compute Cloud (EC2) 'c4.2xlarge' instances in the 'us-east-1' region with a smaller "runner" instance responsible for coordinating each of the "worker" instances.

Each experiment consists of several steps:

1. Each worker instance builds a clean Docker image containing a specific revision of our fork of the Parity software[10] containing the Ouroboros proof-of-concept changes based on the Parity 1.6.8 release.

2. Each worker instance is started in an "isolated" mode where none of the nodes talk to each other. During this period, a Parity account is recovered on each node and a start time for the network is established.

3. Each worker instance is restarted in a production mode that allows communication between the nodes and transactions to be mined.

4. A single worker instance is informed about all the other nodes. All nodes become aware of all other nodes via Parity's peer-to-peer discovery methods.

[9] Ethcore - Parity. `https://ethcore.io/parity.html`

[10] Available from `https://github.com/input-output-hk/parity/tree/experiment-2` (020fd77dc70d3f25e0e0f44bd6b1e19ccf3790d3)

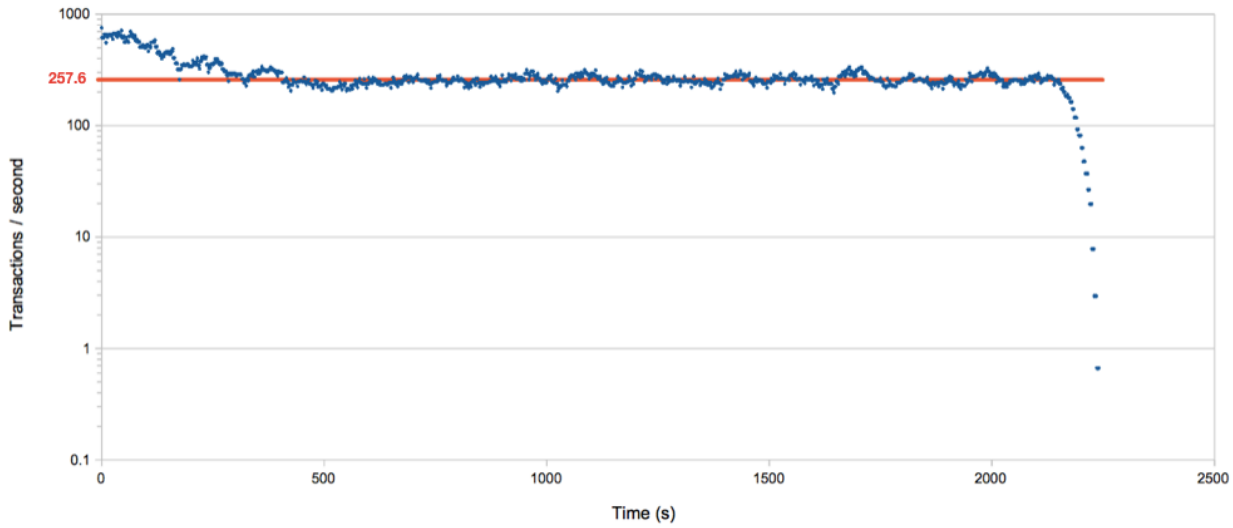


Figure 16: Measuring transactions per second in a 40 node, equal stake deployment with slot length of 5 seconds.

5. Each worker instance has a number of transactions generated and ingested.

In each experiment, 650,000 total transactions are generated between the participating nodes who shared stake equally. The amount transferred in any given transaction is small enough to avoid any account running out of funds. Each instance generates all the transactions using a hard-coded shared random seed, then keeps the transactions originating from the local user account. 20 transactions are saved in a single JSON file, ready to be directly passed to the Parity RPC endpoint using the 'curl' command line tool. During ingestion, a single file of 20 transactions is ingested and one second is spent idle between each file to avoid overwhelming the instances with too many requests.

Various setups were tested, focusing on adjusting the Ouroboros slot duration and the number of participating nodes. 10, 20, 30, and 40 nodes were tested, ultimately limited by the number of instances allowed in a single EC2 region. Slot durations of 5, 10, and 20 seconds were also tested. Variance between experiments was small. In Figure 16 we present the case of 40 nodes and slot length of 5 seconds that exhibits a median value of 257.6 transaction per second.

12 Acknowledgements

We thank Ioannis Konstantinou who contributed in a preliminary version of our protocol. We thank Lars Brünjes, Duncan Coutts, and Kawin Worrasangasilpa for comments on previous drafts of the article. We thank Peter Gaži for comments on previous drafts of the article and assisting us to generalize Theorem 4.30 to viable forks. We thank Saad Quader for noting an inconsistency in previous definitions of common prefix and divergence and assisting us to adapt Theorem 4.30. We thank George Agapov for the prototype implementation of our protocol in Haskell and Jake Goulding for the Parity based implementation.

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] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.

[4] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 913–930. ACM, 2018.

[5] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros chronos: Permissionless clock synchronization via proof-of-stake. Manuscript., 2019.

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

[7] 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.

[8] Iddo Bentov, Rafael Pass, and Elaine Shi. The sleepy model of consensus. *IACR Cryptology ePrint Archive*, 2016:918, 2016.

[9] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.

[10] Daniel J. Bernstein. Chacha, a variant of salsa20. In *SASC: The State of the Art of Stream Ciphers.*, 2008.

[11] 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.

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

[13] Joseph Bonneau. Why buy when you can rent? - bribery attacks on bitcoin-style consensus. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, volume 9604 of *Lecture Notes in Computer Science*, pages 19–26. Springer, 2016.

[14] Vitalik Buterin. Long-range attacks: The serious problem with adaptive proof of work. https://blog.ethereum.org/2014/05/15/long-range-attacks-the-serious-problem-with-adaptive-proof-of-work/, 2014.

[15] Vitalik Buterin. Proof of stake faq. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ, 2016.

[16] Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219. IEEE Computer Society, 2004.

[17] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

[18] Ignacio Cascudo and Bernardo David. SCRAPE: scalable randomness attested by public entities. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings*, volume 10355 of *Lecture Notes in Computer Science*, pages 537–556. Springer, 2017.

[19] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[20] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptology*, 1(1):65–75, 1988.

[21] 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.

[22] Bernardo Machado David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. *IACR Cryptology ePrint Archive*, 2017:573, 2017.

[23] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[24] 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.

[25] 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.

[26] 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.

[27] Bryan Ford. Delegative democracy. http://www.brynosaurus.com/deleg/deleg.pdf, 2002.

[28] 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.

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

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

[31] Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

[32] 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`.

[33] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

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

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

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

[37] 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.

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

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

[40] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. *IACR Cryptology ePrint Archive*, 2016:916, 2016.

[41] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf, January 2016.

[42] Alexander Russell, Cristopher Moore, Aggelos Kiayias, and Saad Quader. Forkable strings are rare. Cryptology ePrint Archive, Report 2017/241, March 2017. `http://eprint.iacr.org/2017/241`.

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

[44] 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 Remarks on Forkable Strings and Divergence

The analysis in the paper demonstrates that if $w \in \{0,1\}^n$ is forkable, then there is in fact a flat fork $F \vdash w$ with the extra property that F has no more than two "components," which is to say that there are two non-empty tines p and q with the property that for any non-empty tine r of F, either $r \sim p$ or $r \sim q$. It is natural to ask whether "two tines suffice," which is to say that there is always a flat fork $F \vdash w$ which is the union of two tines.

In fact, there are forkable strings which require three tines to fork. Specifically, consider the string

$$w = 0^{2k}1^k(1001)1^k0^{2k}\,.$$

The flat fork shown in Figure 17 below proves that w is forkable. However, no two tines can successfully fork w.

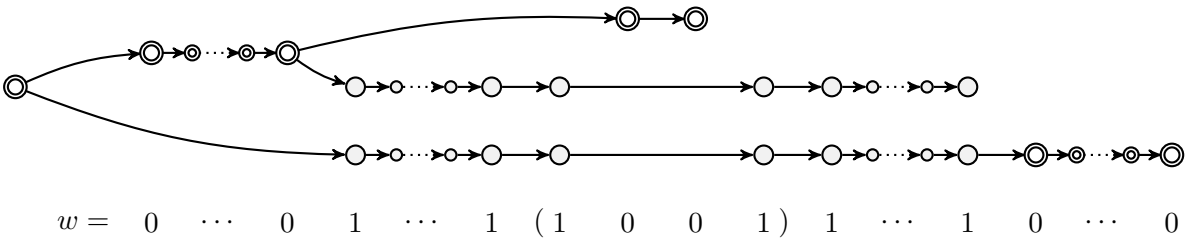


Figure 17: A fork F for the string $w = 0^{2k}1^k(1001)1^k0^{2k}$; honest vertices are highlighted with double borders.