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

Aggelos Kiayias[1†], Alexander Russell[2‡], Bernardo David[3§], and Roman Oliynykov[4]

[1] University of Edinburgh & IOHK, Edinburgh, UK. akiayias@inf.ed.ac.uk
[2] University of Connecticut, CT, USA. acr@cse.uconn.edu
[3] Tokyo Institute of Technology & IOHK, Tokyo, Japan. bernardo.david@iohk.io
[4] IOHK. roman.oliynykov@iohk.io

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.

1 Introduction

A primary consideration regarding the operation of blockchain protocols based on proof of work (PoW)—such as bitcoin [18]—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 [20].

This state of affairs has motivated the investigation of alternative blockchain protocols that would obviate the need for proof of work by substituting it with another, more energy efficient, mechanism that can provide similar guarantees. It is important to point out that the proof of work mechanism of bitcoin facilitates a type of randomized "leader election" process that elects one of the miners to issue the next block. Furthermore, provided that all miners follow the protocol, this selection is performed in a randomized fashion proportionally to the computational power of each miner. (Deviations from the protocol may distort this proportionality as exemplified by "selfish mining" strategies [10, 25].)

[†] 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.
[‡] Research partially supported by National Science Foundation grant #1407205.
[§] Work partly supported by European Research Council Starting Grant # 279447.

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.

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

Previous work. The concept of PoS has been discussed extensively in the bitcoin forum.[5] Proof-of-stake based blockchain design has been more formally studied by Bentov et al., both in conjunction with PoW [4] as well as the sole mechanism for a blockchain protocol [3]. Although Bentov et al. showed that their protocols are secure against some classes of attacks, they do not provide a formal model for analysing PoS based protocols or security proofs relying on precise definitions. Heuristic proof-of-stake based blockchain protocols have been proposed (and implemented) for a number of cryptocurrencies.[6] Being based on heuristic security arguments, these cryptocurrencies have been frequently found to be deficient from the point of view of security. See [3] for a discussion of various attacks.

It is also interesting to contrast a PoS-based blockchain protocol with a classical consensus blockchain that relies on a *fixed* set of authorities (see, e.g., [8]). What distinguishes a PoS-based blockchain from those which assume static authorities is that stake changes over time and hence the trust assumption evolves with the system.

Another alternative to PoW is the concept of *proof of space* [1, 9], which has been specifically investigated in the context of blockchain protocols [21]. In a proof of space setting, a "prover" wishes to demonstrate the utilization of space (storage / memory); as in the case of a PoW, this utilizes a physical resource but can be less energy demanding over time. A related concept is *proof of space-time* (PoST) [16]. In all these cases, however, an expensive physical resource (either storage or computational power) is necessary.

The PoS Design challenge. A fundamental problem for PoS-based blockchain protocols is to simulate the leader election process. In order to achieve a fair randomized election among stakeholders, entropy must be introduced into the system, and mechanisms to introduce entropy may be prone to manipulation

[5] 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.`).

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

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

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

Third, we provide a set of formal arguments establishing that no adversary can break persistence and liveness. Our protocol is secure under a number of plausible assumptions: (1) the network is 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 verify 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. [2], 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 analysis is a natural and fairly general tool that can be applied as part of a security argument the PoS setting.

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* [10, 25] 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 [11]), 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 regarding the 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.[7] 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. Due to lack of space we present the above in the full version [14].

Related 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 [5] 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 perform fail-stop and recover corruptions in addition to Byzantine faults. It is actually 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 that we introduce, and thus is not fully adaptive. Snow White [6] addresses an evolving stakeholder distribution and uses a corruption delay mechanism similar to ours for arguing security. Nevertheless, contrary to our protocol, the Snow White design is susceptible to a "grinding" type of attack that can bias high probability events in favor of the adversary. While this does not hurt security asymptotically, it prevents a concrete parameterisation that does not take into account adversarial computing power. Algorand, [15], 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 length of the execution of the Byzantine agreement protocol). In this respect, despite the existence of forks, blockchain protocols enjoy the flexibility of permitting the clients to set the level of risk that they are willing to undertake, allowing low risk profile clients to enjoy faster processing times. Finally, Fruitchain, [23], 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 have to 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 [12].

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

[7] Nakamoto's simplifications are pointed out in [12]: 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.

synchronized) clocks that indicate the current slot. This will permit them to carry out a distributed protocol intending to collectively assign a block to this current slot. In general, each slot sl_r is indexed by an integer $r \in \{1, 2, \ldots\}$, and we assume that the real time window that corresponds to each slot has the following properties.

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

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

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

In [13, 22] it was shown that persistence and liveness can be derived from the following three elementary properties provided that protocol Π derives the ledger from a data structure in the form of a blockchain.

– **Common Prefix (CP); with parameters $k \in \mathbb{N}$.** The chains $\mathcal{C}_1, \mathcal{C}_2$ possessed by two honest parties at the onset of the slots $sl_1 < sl_2$ 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.
– **Chain Quality (CQ); with parameters $\mu \in (0, 1] \to (0, 1]$ and $\ell \in \mathbb{N}$.** Consider any portion of length at least ℓ of the chain possessed by an honest party at the onset of a round; the ratio of blocks originating from the adversary is at most $1 - \mu$. We call μ the chain quality coefficient.

– **Chain Growth (CG); with parameters $\tau \in (0, 1], s \in \mathbb{N}$.** Consider the chains $\mathcal{C}_1, \mathcal{C}_2$ possessed by two honest parties at the onset of two slots sl_1, sl_2 with sl_2 at least s slots ahead of sl_1. Then it holds that $\mathrm{len}(\mathcal{C}_2) - \mathrm{len}(\mathcal{C}_1) \geq \tau \cdot s$. We call τ the speed coefficient.

Some remarks are in place. Regarding common prefix, we capture a strong notion of common prefix, cf. [13]. Regarding chain quality, the function μ satisfies $\mu(\alpha) \geq \alpha$ for protocols of interest. In an ideal setting, μ would be the identity function: in this case, the percentage of malicious blocks in any sufficiently long chain segment is proportional to the cumulative stake of a set of (malicious) stakeholders.

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

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

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

We stress that contrary to [12], our analysis is in the "standard model", and without a random oracle functionality. Nevertheless we do employ a "diffuse" and "Key and Transaction" functionality with the following interfaces described below.

– Diffuse functionality. It maintains a incoming string for each party U_i that participates. A party, if activated, is allowed at any moment to fetch the contents of its incoming string hence one may think of this as a mailbox. Furthermore, parties can give the instruction to the functionality to diffuse a message. The functionality keeps rounds (called slots) and all parties are allowed to diffuse once in a round. Rounds do not advance unless all parties have diffused a message. The adversary, when activated, can also interact with the functionality and is allowed to read all inboxes and all diffuse requests and deliver messages to the inboxes in any order it prefers. At the end of the round, the functionality will ensure that all inboxes contain all messages that have been diffused (but not necessarily in the same order they have been requested to be diffused). The current slot index may be requested at any time by any party. If a stakeholder 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 stake $s_1, \ldots, s_n$; 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 public-key registered the functionality will allow the adversary to set their public-keys while for honest users the functionality will sample public/secret-key pairs and record them. 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 retrieve its public and secret-key, whereupon, the functionality will return it to the user. (ii) The whole directory of public-keys may be required in whereupon, the functionality will return it to the requesting user. (iii) A new user may be requested to be created by a message $(\mathsf{Create}, U, \mathcal{C})$ from the environment, in which case the functionality will follow the same procedure as before: it will consult the adversary regarding the corruption status of U and will set its public and possibly secret-key depending on the corruption status; moreover it will store $\mathcal{C}$ as the suggested initial state. The functionality will return the public-key back to the environment upon successful completion of this interaction. (iv) A transaction may be requested on behalf of a certain user by the environment, by providing a template for the transaction (which should contain a unique nonce) and a recipient. The functionality will adjust the stake of each stakeholder accordingly. (v) 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 the secret-key will be released after D slots have passed according to the round counter maintained in the Diffuse interface.

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

– At each slot sl_j, the environment $\mathcal{Z}$ is allowed to activate any subset of stakeholders it wishes. Each one of them will possibly produce messages that are to be transmitted to other stakeholders.
– The adversary is activated 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 (independently of whether it is a slot leader).
– 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.
– In each slot sl_r, and for each active stakeholder U_j there will be 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 $j = 1, \ldots, n_r$ where n_r is the number of users introduced up to that slot. Public-keys will be marked as "corrupted" if the corresponding stakeholder has been corrupted. We will say the adversary is restricted to less than 50% relative stake if it holds that the total stake of the corrupted keys divided by the total stake $\sum_i s_i$ is less than 50% in all possible $\mathbb{S}_j(r)$. In case the above is violated an event $\mathsf{Bad}^{1/2}$ becomes true for the given execution.

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

3 Our Protocol: Overview

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

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

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

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

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

Stage 3: Dynamic state without a beacon, epoch period of R slots, $R = \Theta(k)$ and delay $D \in (R, 2R) \ll L$. In the third stage, we remove the dependency to the beacon, by introducing a secure multiparty protocol with "guaranteed output delivery" that simulates it. In this way, we can obtain the long-livedness of the protocol as described in the stage 2 design but only under the assumption of the stage 1 design, i.e., the mere availability of an initial random string and an initial stakeholder distribution with honest majority. The core idea is the following: given we guarantee that an honest majority among elected stakeholders will hold with very high probability, we can further use this elected set as participants to an instance of a secure multiparty computation (MPC) protocol. This will require the choice of the length of the epoch to be sufficient so that it can accommodate a run of the MPC protocol. From a security point of view, the main difference with the previous case, is that the output of the beacon will become known to the adversary before it may become known to the honest parties. Nevertheless, we will prove that the honest parties will also inevitably learn it after a short number of slots. To account for the fact that the adversary gets this headstart (which it may exploit by performing adaptive corruptions) we increase the wait time for corruption from R to a suitable value in $(R, 2R)$ that negates this advantage and depends on the secure MPC design. 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 running the protocol 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. Due to lack of space we refer to the full version for more details, [14].

4 Our Protocol: Static State

4.1 Basic Concepts and Protocol Description

We begin by describing the blockchain protocol π_{SPoS} in the "static stake" setting, where leaders are assigned to blockchain slots with probability proportional to their (fixed) initial stake 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 secure computation.

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

In the static stake case, we assume that a fixed collection of n stakeholders $U_1, \ldots, U_n$ interact throughout the protocol. Stakeholder U_i possesses s_i stake before the protocol starts. For each stakeholder U_i a verification and signing key pair $(\mathsf{vk}_i, \mathsf{sk}_i)$ for a 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 [12].

Definition 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 2 (State). *A* state *is a string* $st \in \{0,1\}^\lambda$.

Definition 3 (Block). *A* block B *generated at a slot* $sl_i \in \{sl_1, \ldots, sl_R\}$ *contains the current state* $st \in \{0,1\}^\lambda$, *data* $d \in \{0,1\}^*$, *the slot number* sl_i *and a signature* $\sigma = \mathsf{Sign}_{\mathsf{sk}_i}(st, d, sl)$ *computed under* sk_i *corresponding to the stakeholder* U_i *generating the block.*

Definition 4 (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 $\mathcal{C}_2$.

Definition 5 (Epoch). *An* epoch *is a set of* R *adjacent slots* $S = \{sl_1, \ldots, sl_R\}$.

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

Definition 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.*

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

Definition 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 "weighing 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 $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$

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

- Upon receiving $(\mathsf{genblock\_req}, U_i)$ from stakeholder U_i, $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ proceeds as follows. If ρ has not been set, $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ samples $\rho \leftarrow \mathcal{D}$. In any case, $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ sends $(\mathsf{genblock}, \mathbb{S}_0, \rho, \mathsf{F})$ to U_i.

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

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

The protocol relies on a $\mathsf{maxvalid}_S(\mathcal{C}, \mathbb{C})$ function that chooses a chain given the current chain $\mathcal{C}$ and a set of valid chains $\mathbb{C}$ that are available in the network.

In the static case we analyze the simple "longest chain" rule. (In the dynamic case the rule is parameterized by a common chain length; see Section 5.)

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

Protocol π_{SPoS}

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

1. **Initialization** 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 $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$, 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 as $\mathcal{C}$ and the initial internal state $st = H(\mathrm{head}(\mathcal{C}))$.
2. **Chain Extension** For every slot $sl_j \in S$, every online stakeholder U_i performs the following steps:
   (a) Collect all valid chains received via broadcast into a set $\mathbb{C}$, verifying that for every chain $\mathcal{C}' \in \mathbb{C}$ and every block $B' = (st', d', sl', \sigma') \in \mathcal{C}'$ it holds that $\mathsf{Vrf}_{\mathsf{vk}'}(\sigma', (st', d', sl')) = 1$, where vk' is the verification key of the stakeholder $U' = \mathsf{F}(\mathbb{S}_0, \rho, sl')$. U_i computes $\mathcal{C}' = \mathsf{maxvalid}(\mathcal{C}, \mathbb{C})$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\mathrm{head}(\mathcal{C}'))$.
   (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 $\sigma = \mathsf{Sign}_{\mathsf{sk}_i}(st, d, sl_j)$ 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 $st = H(\mathrm{head}(\mathcal{C}'))$.

Fig. 2. Protocol π_{SPoS}.

4.2 Forkable Strings

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 8 (Characteristic String). *Fix an execution with genesis block B_0, adversary $\mathcal{A}$, and environment $\mathcal{Z}$. Let $S = \{sl_{i+1}, \ldots, sl_{i+n}\}$ denote a sequence of slots of length $|S| = n$. The* characteristic string *$w \in \{0,1\}^n$ of S is defined so that $w_k = 1$ if and only if the adversary controls the slot leader of slot sl_{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 on our approach to analyze the protocol. Let $w \in \{0,1\}^n$ be a characteristic string for a sequence of slots S. Consider two observers that (i.) go offline immediately prior to the commencement of S, (ii.) have the same view $\mathcal{C}_0$ of the current chain prior to the commencement of S, and (iii.) come back online at the last slot of S and request an update of their chain. A fundamental concern in our analysis is the possibility that such observers can be presented with a "diverging" view over the sequence S: specifically, the possibility that the adversary can force the two observers to adopt two different chains $\mathcal{C}_1, \mathcal{C}_2$ whose common prefix is $\mathcal{C}_0$.

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

To reason about such "forkings" of a characteristic string $w \in \{0,1\}^n$, we define below a formal notion of "fork" that captures the relationship between the chains broadcast by *honest* slot leaders during an execution of the protocol π_{SPoS}. In preparation for the definition, we recall that honest players always choose to extend a maximum length chain among those available to the player on the network. Furthermore, if such a maximal chain $\mathcal{C}$ includes a block 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 $\mathcal{C}$ broadcast 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_{SPoS} are comprised of blocks containing a variety of data that are immaterial for reasoning about forking. 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.

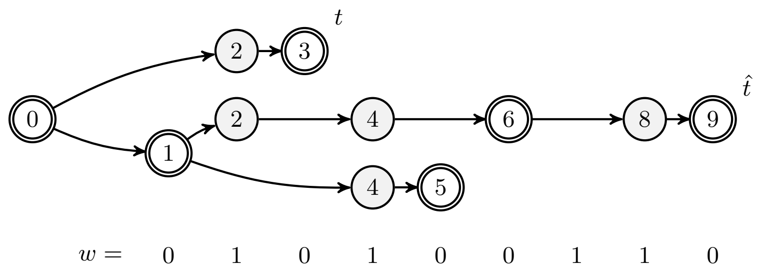

Fig. 3. A fork F for the string $w = 010100110$; vertices appear with their labels and honest vertices are highlighted with double borders. Note that the depths of the (honest) vertices associated with the honest indices of w are strictly increasing. Two tines are distinguished in the figure: one, labeled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

Definition 9 (Fork). *Let $w \in \{0,1\}^n$ and let $H = \{i \mid w_i = 0\}$ denote the set of honest indices. A* fork *for the string w is a directed, rooted tree $F = (V, E)$ with a labeling $\ell : V \to \{0, 1, \ldots, n\}$ so that*

- *each edge of F is directed away from the root;*
- *the root $r \in V$ is given the label $\ell(r) = 0$;*
- *the labels along any directed path in the tree are strictly increasing;*
- *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 10 (Tines 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. The* height *of a fork (as usual for a tree) is defined to be the length of the longest tine. 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.*

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.

See Figure 3 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.

Definition 11. *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_{SPoS} to yield two entirely disjoint chains of maximum length, 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.

Theorem 1. *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 $\Pr[w \textit{ is forkable}] = 2^{-\Omega(\sqrt{n})}$.*

In subsequent work, Russell et al. [24] improved this bound to $2^{-\Omega(n)}$.

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

Definition 12 (Fork prefixes). *If w is a prefix of the string $w' \in \{0,1\}^*$, $F \vdash w$, and $F' \vdash w'$, we say that F is a* prefix *of F', written $F \sqsubseteq F'$, if F is a consistently-labeled subgraph of F'. Specifically, every vertex and edge of F appears in F' and, furthermore, the labels given to any vertex appearing in both F and F' are identical.*

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

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

Definition 13 (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 a closed fork has a unique longest tine (as all maximal tines terminate with an honest vertex, and these must have distinct depths). Note, additionally, that if w is a prefix of w' and $F' \vdash w'$, then there is a unique closed fork $F \vdash w$ for which $F \sqsubseteq F'$.

Definition 14 (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* $\text{gap}(t)$*, to be the difference in length between $\hat{t}$ and t; thus*

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

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*

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

We remark that this quantity depends both on F and the specific string w associated with F. Finally, for a tine t we define

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

Definition 15 (Margin). *For a closed fork $F \vdash w$ we define $\lambda(F)$ to be the maximum reach taken over all tines in F:*

$$\lambda(F) = \max_{t} \text{reach}(t)\,.$$

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

$$\text{margin}(F) = \mu(F) = \max_{t_1 \not\sim t_2} \Big(\min\{\text{reach}(t_1), \text{reach}(t_2)\} \Big)\,. \tag{1}$$

We remark that the maxima above can always obtained by honest tines. Specifically, if t is an adversarial tine of a fork $F \vdash w$, $\text{reach}(t) \leq \text{reach}(\bar{t})$, 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 (1) which satisfy $\text{reach}(t_1) = \lambda(F) \geq \text{reach}(t_2) = \mu(F)$.

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

Proposition 1. *A string w is forkable if and only if there is a closed fork $F \vdash w$ for which* $\text{margin}(F) \geq 0$*.*

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. As mentioned above, there is a unique closed fork $\overline{F} \vdash w$ obtained from F by removing any adversarial vertices from the ends of the tines of F. Note that the tine $\hat{t}$ containing $\hat{i}$ 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 $\text{margin}(\overline{F}) \geq 0$ as desired.

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

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

and, for convenience,

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

Note that this overloads the notation $\lambda(\cdot)$ 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 $\lambda(w)$ and $\mu(w)$ can be achieved by the same fork, though this is established by the full treatment in [14]. In any case, it is clear that $\lambda(w) \geq 0$ and $\lambda(w) \geq \mu(w)$ for all strings w; furthermore, by Proposition 1 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.

With these definitions in place, we are prepared to survey the proof of Theorem 1.

Proof (of Theorem 1; high level survey). The proof proceeds by establishing a recursive description of $\mathbf{m}(w0)$ and $\mathbf{m}(w1)$ in terms of $\mathbf{m}(w)$ and providing an analysis of the Markov chain that arises by considering $\mathbf{m}(\cdot)$ for strings drawn from a binomial distribution. This yields an upper bound on the probability that $\mu(w) \geq 0$ and hence the event that w is forkable. The full proof appears in the e-print version of the paper [14].

Covert adversaries. Observe that an adversary that broadcasts *two distinct* blocks for a particular slot leaves behind a suspicious "audit trail"—multiple signed blocks for the same slot—which conspicuously deviates from the protocol. This may be undesirable for certain practical adversaries, who wish to maintain the facade of honesty. We say that such an adversary is "covert" and note that such adversaries have reduced power to disrupt the protocol. We discuss this in detail and consider the probability of forkability with these weakened adversaries in the full version of the paper [14].

4.3 Common Prefix

Recall that the chains constructed by honest players during an execution of π_{SPoS} correspond to tines of a fork, as defined and studied in the previous sections.

The random assignment of slots to stakeholders given by $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ guarantees that the coordinates of the associated characteristic string w follow the binomial distribution with probability equal to the adversarial stake. Thus Theorem 1 establishes that no execution of the protocol π_{SPoS} can induce two tines (chains) of maximal length with no common prefix.

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

Theorem 2. *Let $k, R \in \mathbb{N}$ and $\epsilon \in (0,1)$. The probability that the π_{SPoS} protocol, when executed with a $(1-\epsilon)/2$ fraction of adversarial stake, violates the common prefix property with parameter k throughout an epoch of R slots is no more than $\exp(-\Omega(\sqrt{k}) + \ln R)$; the constant hidden by the $\Omega()$ notation depends only on ϵ.*

Proof (sketch). The full proof (see [14]) proceeds by showing that if common prefix with parameter k is violated for a particular fork, then the underlying characteristic string must have a forkable substring of length k. Thus

$$\Pr[\text{common prefix violation}] \leq \Pr\begin{bmatrix}\exists \alpha, \beta \in \{1, \dots, R\} \text{ so that } \alpha + k - 1 \leq \beta \\ \text{and } w_\alpha \dots w_\beta \text{ is forkable}\end{bmatrix}$$
$$\leq \sum_{1 \leq \alpha \leq R} \underbrace{\sum_{\alpha+k-1 \leq \beta \leq R} \Pr[w_\alpha \dots w_\beta \text{ is forkable}]}_{(*)} .$$

Recall that the characteristic string $w \in \{0,1\}^R$ for such an execution of π_{SPoS} is determined by assigning each $w_i = 1$ independently with probability $(1-\epsilon)/2$. According to Theorem 1 the probability that a string of length t drawn from this distribution is forkable is no more than $\exp(-c\sqrt{t})$ for a positive constant c. Note that for any $\alpha \geq 1$,

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

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

$$\Pr[\text{common prefix violation}] \leq R \cdot \exp(-\Omega(\sqrt{k})) \leq \exp(\ln R - \Omega(\sqrt{k}))\,,$$

as desired.

4.4 Chain Growth and Chain Quality

Anticipating these two proofs, we record an additive Chernoff–Hoeffding bound. (See, e.g., [17] for a proof.)

Theorem 3 (Chernoff–Hoeffding bound). *Let $X_1, \ldots, X_T$ be independent random variables with $\mathbb{E}[X_i] = p_i$ and $X_i \in [0,1]$. Let $X = \sum_{i=1}^{T} X_i$ and $\mu = \sum_{i=1}^{T} p_i = \mathbb{E}[X]$. Then, for all $\delta \geq 0$,*

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

We will start with the chain growth property.

Theorem 4. *The π_{SPoS} protocol satisfies the chain growth property with parameters $\tau = 1 - \alpha, s \in \mathbb{N}$ throughout an epoch of R slots with probability at least $1 - \exp(-\Omega(\epsilon^2 s) + \ln R)$ against an adversary holding an $\alpha - \epsilon$ portion of the total stake.*

Proof (sketch). The proof proceeds by applying the Chernoff bound to ensure that with high probability a characteristic string drawn from the binomial distribution has a $\approx \tau = (1 - \alpha)$ fraction of honest indices. Note that each honest player will force the length of the resulting chain to increase by one in any execution of π_{SPoS}. See [14] for a complete presentation.

Having established chain growth we now turn our attention to chain quality. Recall that the chain quality property with parameters μ and ℓ asserts that among every ℓ consecutive blocks in a chain (possessed by an honest user), the fraction of adversarial blocks is no more than μ.

Theorem 5. *Let $\alpha - \epsilon$ be the adversarial stake ratio. The π_{SPoS} protocol satisfies the chain quality property with parameters $\mu(\alpha - \epsilon) = \alpha/(1-\alpha)$ and $\ell \in \mathbb{N}$ throughout an epoch of R slots with probability at least*

$$1 - \exp\bigl(-\Omega(\epsilon^2 \alpha \ell) + \ln R\bigr).$$

Proof (sketch). This likewise follows from appropriate application of the Chernoff bound. See [14] for full discussion.

5 Our Protocol: Dynamic Stake

5.1 Using a Trusted Beacon

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

Before describing the protocol for the case of dynamic stake, we need to explain the modification of $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ so that multiple epochs are considered. The resulting functionality, $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$, allows stakeholders to query it for the leader selection data specific to each epoch. $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ is parameterized by the initial stake of each stakeholder before the first epoch e_1 starts; in subsequent epochs, parties will take into consideration the stake distribution in the latest block of the previous epoch's first $R-2k$ slots. Given that there is no predetermined view of the stakeholder distribution, the functionality $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ 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 time stamp at most $(j-1)R-2k$, provided all honest parties agree on it, or is undefined if the honest parties disagree on it. The functionality $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ is defined in Figure 4.

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

$\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ 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, $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ operates as follows:

– Upon receiving $(\mathsf{genblock\_req}, U_i)$ from stakeholder U_i it operates as functionality $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ on that message.
– Upon receiving $(\mathsf{epochrnd\_req}, U_i, e_j)$ from stakeholder U_i, if $j \geq 2$ is the current epoch, $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ proceeds as follows. If ρ^j has not been set, $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ samples $\rho^j \leftarrow \mathcal{D}$. Then, $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ sends $(\mathsf{epochrnd}, \rho^j)$ to U_i.

Fig. 4. Functionality $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$.

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

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

Protocol π_{DPoS} is described in Figure 5 and functions in the $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$-hybrid model.

Protocol π_{DPoS}

π_{DPoS} is a protocol run by a set of stakeholders, initially equal to $U_1, \ldots, U_n$, interacting among themselves and with $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ over a sequence of L slots $S = \{sl_1, \ldots, sl_L\}$. 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 $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$, 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 $st = H(\text{head}(\mathcal{C}))$.
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 time stamp less than $jR - 2k$ as reflected in $\mathcal{C}$ and sends $(\mathsf{epochrnd\_req}, U_i, e_j)$ to $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$, 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 $st = H(\text{head}(\mathcal{C}'))$.
   (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 $st = H(\text{head}(\mathcal{C}'))$.

Fig. 5. Protocol π_{DPoS}

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. It is worth noting that this restriction can be easily lifted if one can trust honest stakeholders to securely erase their memory; in such case, a forward secure signature can be employed to thwart any past corruption attempt that tries to circumvent $\mathsf{Bad}^{1/2}$.

5.2 Simulating a Trusted Beacon

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

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

Protocol π_{DLS} is described in Figure 6 and uses a publicly verifiable secret sharing (PVSS) [26] (we defer to the full version the full description of the scheme).

The assumption we will use about the PVSS scheme is that the resulting coin-flipping protocol simulates a perfect beacon with distinguishing advantage ϵ_{DLS}. 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. We remark that using [26] as a PVSS, a simulator can achieve simulatability in the random oracle model by taking advantage of the programmability of the oracle. Using a random oracle is by no means necessary though and the same benefits may be obtained by a CRS embedded into the genesis block.

5.3 Robust Transaction Ledger

We are now ready to state the main result of the section that establishes that the π_{DPOS} protocol with the protocol π_{DLS} as a sub-routine implements 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 16. *Consider two slots sl_1, sl_2 and an execution $\mathcal{E}$. The stake shift between sl_1, sl_2 is the maximum possible statistical distance of the two weighted-by-stake distributions that are defined using the stake reflected in the chain $\mathcal{C}_1$ of some honest stakeholder active at sl_1 and the chain $\mathcal{C}_2$ of some honest stakeholder active at sl_2 respectively.*

Given the definition above we can now state the following theorem.

Theorem 6. *Fix parameters $k, R, L \in \mathbb{N}, \epsilon, \sigma \in (0,1)$. Let $R = 10k$ be the epoch length and L the total lifetime of the system. Assume the adversary is restricted to $\frac{1-\epsilon}{2} - \sigma$ relative stake and that the π_{SPOS} protocol satisfies the common prefix property with parameters R, k and probability of error ϵ_{CP}, the chain quality*

Protocol π_{DLS}

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 $R = 10k$ slots, without loss of generality denoted by $U_1, \ldots, U_R$ (which are not necessarily distinct), and entails the following phases.

1. **Commitment Phase** ($4k$ slots) When epoch e_j starts, for $1 \leq i \leq n$, stakeholder U_i samples a uniformly random string u_i and randomness r_i for the underlying commitment scheme, generates shares $\sigma_1^i, \ldots, \sigma_n^i \leftarrow \mathsf{Deal}(n, u_i)$ and encrypts each share σ_k^i under stakeholder U_k's public-key. Finally, U_i posts the encrypted shares and commitments $\mathsf{Com}(r_i, u_i)$ to the blockchain.
2. **Reveal Phase** ($4k$ slots) After slot $4k$, for $1 \leq i \leq n$, stakeholder U_i opens its commitment by posting $\mathsf{Open}(r_i, u_i)$ to the blockchain provided that the blockchain contain valid shares from the majority of $U_1, \ldots, U_R$; if not, each U_i terminates.
3. **Recovery Phase** ($2k$ slots) After slot $8k$, for any stakeholder U^a that has not participated in the reveal phase, i.e., it has not posted in $\mathcal{C}^{\lceil k}$ an $\mathsf{Open}(r_a, u_a)$ message, for $1 \leq i \leq R$, U_i submits its share σ_i^a for insertion to the blockchain. When all shares $\sigma_1^a, \ldots, \sigma_n^a$ are available, each stakeholder U_i can compute $\mathsf{Rec}(\sigma_1^a, \ldots, \sigma_n^a)$ to reconstruct u_a (independently of whether U^a opens the commitment or not).

   The simulation of $\mathsf{epochrnd\_req}$ is then as follows.
   - Given input $(\mathsf{genblock\_req}, U_i, e_j, \mathbb{S}_j)$, the stakholder uses the commitment values in the blockchain to compute $\rho^j = \sum_{l \in \mathbb{L}} u_l$ where $\mathbb{L}$ is the subset of stakeholders that were elected in epoch e_j. It returns $(\mathsf{genblock}, B_0, \mathbb{S}_j)$ with $B_0 = (\mathbb{S}_j, \rho^j)$.

Fig. 6. Protocol π_{DLS}.

property with parameters $\mu \geq 1/k, k$ and probability of error ϵ_{CQ} and the chain growth property with parameters $\tau \geq 1/2, k$ and probability of error ϵ_{CG}. Furthermore, assume that π_{DLS} simulates a perfect beacon with distinguishing advantage ϵ_{DLS}.

Then, the π_{DPOS} protocol satisfies persistence with parameters k and liveness with parameters $u = 2k$ throughout a period of L slots (or $\mathsf{Bad}^{1/2}$ happens) with probability $1 - (L/R)(\epsilon_{\mathsf{CQ}} + \epsilon_{\mathsf{CP}} + \epsilon_{\mathsf{CG}} + \epsilon_{\text{DLS}})$, assuming that σ is the maximum stake shift over $10k$ slots, corruption delay $D \geq 2R - 4k$ and no honest player is offline for more than k slots.

Proof. (sketch) Let us first consider the execution of π_{DPOS} when $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ is used instead of π_{DLS}. Let BAD_r be the event that any of the three properties $\mathsf{CP}, \mathsf{CQ}, \mathsf{CG}$ is violated at round $r \geq 1$ while no violation of any of them occurred prior to r. It is easy to see that $\Pr[\cup_{r \leq R} \mathsf{BAD}_r] \leq \epsilon_{\mathsf{CQ}} + \epsilon_{\mathsf{CP}} + \epsilon_{\mathsf{CG}}$. Conditioning now on the negation of this event, we can repeat the argument for the second epoch, since $D \geq R$ and thus the adversary cannot influence the stakeholder selection for the second epoch. It follows that $\Pr[\cup_{r \leq L} \mathsf{BAD}_r] \leq (L/R)(\epsilon_{\mathsf{CQ}} + \epsilon_{\mathsf{CP}} + \epsilon_{\mathsf{CG}})$. It is easy now to see that persistence and liveness hold conditioning on the negation of the above event: a violation of persistence would violate common prefix. On the other hand, a violation of liveness would violate either chain growth or chain quality for the stated parameters.

Observe that the above result will continue to hold even if $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ was weakened to allow the adversary access to the random value of the next epoch $6k$ slots ahead of the end of the epoch. This is because the corruption delay $D \geq 2R - 4k = 16k$.

Finally, we examine what happens when $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ is substituted by $\mathcal{F}_{\mathsf{LS}}^{\mathcal{D},\mathsf{F}}$ and the execution of protocol π_{DLS}. Consider an execution with environment $\mathcal{Z}$ and adversary $\mathcal{A}$ and event BAD that happens with some probability β in this execution. We construct an adversary $\mathcal{A}^*$ that operates in an execution with $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$, weakened as in the previous paragraph, and induces the event BAD with roughly the same probability β. $\mathcal{A}^*$ would operate as follows: in the first $4k$ slots, it will use an honest party to insert in the blockchain the simulated commitments of the honest parties; this is feasible for $\mathcal{A}^*$ as in $4k$ slots, chain growth will result in the blockchain growing by at least $2k$ blocks and thus in the first k blocks there will be at least a single honest block included. Now $\mathcal{A}^*$ will obtain from $\mathcal{F}_{\mathsf{DLS}}^{\mathcal{D},\mathsf{F}}$ the value of the beacon and it will simulate the opening of all the commitments on behalf of the honest parties. Finally, in the last $2k$ slots it will perform the forced opening of all the adversarial commitments that were not opened. The protocol simulation will be repeated for each epoch and the statement of the theorem follows. $\square$

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. We omit further details.

6 Incentives

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

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 runs the secure multiparty coin flipping protocol and are the slot leaders of the epoch. Together with those there is a (not necessarily disjoint) set of stakeholders called the endorsers. Now each slot has two types of stakeholders associated with it; the slot leader who will issue the block as before and the slot *endorser* who will endorse the input to be included in the block. Moreover, contrary to slot leaders, we can elect multiple slot endorsers for each slot, nevertheless, without loss of generality we just assume a single input endorser per slot in this description. While this seems like an insignificant modification it gives us a room for improvement because of the following reason: endorsers' contributions will be acceptable even if they are d slots late, where $d \in \mathbb{N}$ is a parameter.

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

A suitable class of reward mechanisms. The reward mechanism that we will pair with input endorsers operates as follows. First we set the endorsing acceptance window, d to be $d = 2k$. 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 $\{jR + 1, \ldots, (j + 1)R + 2k\}$ 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 total reward pool P_R 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, 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 contributed the sequence of r endorsed inputs. Subsequently, the i-th stakeholder U_i can claim a reward up to the amount $(\beta \cdot |\{j \mid U_i = E_j\}|/r + (1 - \beta) \cdot |\{j \mid U_i = L_j\}|/R)P$ where $\beta \in [0, 1]$. Claiming a reward is performed by issuing a "coinbase" type of transaction at any point after $4k$ blocks 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 $jR + 4k$.

We proceed to show that our system is a δ-Nash (approximate) equilibrium, cf. [19, 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 extract from the execution lasting $L = tR + 4k + 1$ slots, is equal to

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

for any execution $\mathcal{E}$ where common prefix holds with parameter k, where r_j is the total endorsed inputs emitted in the j-th epoch (and possibly included at any time up to the first $2k$ 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.

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. Furthermore, observe that we leave the value of $\mathcal{R}_V(\mathcal{E})$ undefined when $\mathcal{E}$ is an execution where common prefix fails: it will not make sense to consider this value for such executions since the view of the protocol of honest parties can be divergent; nevertheless this will not affect our overall analysis since such executions will happen with sufficiently small probability.

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 7. *Fix any $\delta > 0$; the honest strategy in the protocol is a δ-Nash equilibrium against any coalition commanding a proportion of stake less than $(1-\epsilon)/2 - \sigma$ for some constants $\epsilon, \sigma \in (0,1)$ as in Theorem 6, provided that the maximum total rewards P_{all} provided in all possible protocol executions is bounded by a polynomial in λ, while $\epsilon_{\mathsf{CQ}} + \epsilon_{\mathsf{CP}} + \epsilon_{\mathsf{CG}} + \epsilon_{\mathsf{DLS}}$ is negligible in λ.*

We refer to the full version of the paper, [14], for the proof.

Remark 3. In the above theorem, for simplicity, we assumed that protocol costs are not affective 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 MPC protocol. The proof would be resilient to these modifications because endorsed inputs and MPC 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.

7 Stake Delegation

In this section we introduce a *delegation scheme* whereby the stakeholders of the PoS protocol can delegate the protocol execution rights to another set of parties, the *delegates*. 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).

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* [7].

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 [7]. 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 [7].

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.

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).

References

1. 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.
2. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.
3. Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. *CoRR*, abs/1406.5694, 2014.
4. Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]y. *SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.
5. Iddo Bentov, Rafael Pass, and Elaine Shi. The sleepy model of consensus. *IACR Cryptology ePrint Archive*, 2016:918, 2016.
6. Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.
7. Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure proxy signature schemes for delegation of signing rights. *J. Cryptology*, 25(1):57–115, 2012.
8. 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.
9. 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.

10. 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.
11. Bryan Ford. Delegative democracy. http://www.brynosaurus.com/deleg/deleg.pdf, 2002.
12. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.
13. Aggelos Kiayias and Giorgos Panagiotakos. Speed-security tradeoffs in blockchain protocols. Cryptology ePrint Archive, Report 2015/1019, 2015. `http://eprint.iacr.org/2015/1019`.
14. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Report 2016/889, 2017. `http://eprint.iacr.org/2016/889`.
15. Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.
16. Tal Moran and Ilan Orlov. Proofs of space-time and rational proofs of storage. Cryptology ePrint Archive, Report 2016/035, 2016. `http://eprint.iacr.org/2016/035`.
17. Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.
18. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf, 2008.
19. Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.
20. Karl J. O'Dwyer and David Malone. Bitcoin mining and its energy footprint. *ISSC 2014 / CIICT 2014, Limerick, June 26–27*, 2014.
21. 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.
22. Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR Cryptology ePrint Archive*, 2016:454, 2016.
23. Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. *IACR Cryptology ePrint Archive*, 2016:916, 2016.
24. 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`.
25. Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *CoRR*, abs/1507.06183, 2015.
26. 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.