Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake protocol

Bernardo David[*] Peter Gaži[†] Aggelos Kiayias[‡] Alexander Russell[§]

June 13, 2017

Abstract

We present "Ouroboros Praos", a new proof-of-stake blockchain protocol that provides, for the first time, a robust distributed ledger that is provably secure in the *semi-synchronous adversarial setting*, i.e., assuming a delay Δ in message delivery which is unknown to protocol participants, and *fully adaptively secure*, i.e., the adversary can choose to corrupt any participant of an ever evolving population of stakeholders at any moment as long the stakeholder distribution maintains an honest majority of stake at any given time. To achieve that, our protocol puts to use forward secure digital signatures and a new type of verifiable random functions that maintains unpredictability under malicious key generation, a property we introduce and instantiate in the random oracle model. Our security proof entails a combinatorial analysis of a class of forkable strings tailored to semi-synchronous blockchains that may be of independent interest in the context of security analysis of blockchain protocols.

1 Introduction

The design of *proof-of-stake* blockchain protocols has been identified early on as an important objective in blockchain protocol design; a proof-of-stake blockchain substitutes the proof-of-work component in Nakamoto's blockchain protocol [Nak08] while still providing similar guarantees in terms of transaction processing in the presence of a dishonest minority of users, where this "minority" is to be understood here in the context of stake rather than computational power.

The basic security properties of blockchain protocols from a cryptographic point of view were first identified in [GKL15] and further studied in [KP15, PSS17]; these include common prefix, chain quality and chain growth and refer to resilient qualities of the underlying data structure of the blockchain in the presence of an adversary that attempts to subvert them.

Proof-of-stake protocols follow a structure that is typified by the following characteristics. Based on her local view, a party is capable of deciding, in a publicly verifiable way, whether she is permitted to produce the next block. Assuming the block is valid, other parties update their local views by adopting the block, and proceed in this way ad infinitum. At any moment, the probability of being permitted to issue a block is proportional to the relative stake a player has in the system, as reported by the blockchain itself.

[*]Tokyo Tech and IOHK, `bernardo.david@iohk.io`.

[†]IOHK, `peter.gazi@iohk.io`. Work partly done while the author was a postdoc at IST Austria, supported by the ERC consolidator grant 682815-TOCNeT.

[‡]University of Edinburgh and IOHK. `akiayias@inf.ed.ac.uk`. Work partly supported by H2020 Project #653497, PANORAMIX.

[§]University of Connecticut. `acr@cse.uconn.edu`.

It is easy to see that the above mechanism requires some entropy to be regularly injected into the system. This follows from the fact that as the stake shifts, together with the evolving population of stakeholders, an adversary observing the system can predict possible identities that are likely to be elected and transition the stake it possesses to those identities. Realizing this entropy injection mechanism in a way that provably prevents biasing is a delicate task that so far has eluded a practical solution that is secure against all possible attacks.

Our Results. We present "Ouroboros Praos", a provably secure proof-of-stake protocol that is the first to be secure against adaptive attackers and scalable in a truly practical sense. Our protocol is based on a previous proof-of-stake protocol, Ouroboros [KRDO17], as its analysis relies on some of the core combinatorial arguments that were developed to analyze that scheme. Nevertheless, the protocol construction has a number of novel elements that require a significant recasting and generalization of the previous combinatorial analysis. In more details our results are as follows.

In Ouroboros Praos, deciding whether a certain participant of the protocol is eligible to issue a block is decided via a private test that is executed locally using a special verifiable random function (VRF) on the current time-stamp and a nonce that is determined for a period of time known as an "epoch". The special feature of this VRF primitive, that is novel to our approach, is that the VRF must have strong security characteristics even in the setting of malicious key generation: specifically, if provided with an input that has high entropy, the output of the VRF is unpredictable, even when adversary has subverted the key generation procedure. We call such VRF functions VRF with "unpredictability under malicious key generation." This security property and forward security of the signature scheme are at the core of our security analysis for adaptive corruptions. Using a signature scheme with forward security also allows us to relax the restrictions on the period during which a stakeholder can be offline and on introduction of new stakeholders in comparison to Ouroboros. Ouroboros Praos allows honest stakeholders to be offline for arbitrary amounts of slots and allows a newly spawned stakeholder to be initialized with merely the genesis block.

In more detail, we analyze our protocol in the partial or semi-synchronous model, [DLS88, PSS17]. In this setting, there is a maximum delay Δ that is applied on message delivery and is unknown to the protocol participants. In order to cope with the Δ-semisynchronous setting we introduce the concept of "empty slots" which occur with sufficient frequency to enable short periods of silence that facilitate synchronization. This feature of the protocol gives also its moniker, "Praos", standing for "mellow", or "gentle". Ensuring that the adversary cannot exploit the stakeholder keys that it possesses to confuse or out-maneuver the honest parties, we develop a combinatorial analysis to show that the simple rule of following the longest chain still enables the honest parties to converge to a unique view with high probability. To accomplish this we revisit and expand the forkable strings and divergence analysis of [KRDO17]. We remark that significant alterations are indeed necessary: As we demonstrate in Appendix C, the protocol of [KRDO17] and its analysis is critically tailored to synchronous operation and is susceptible to a desynchronization attack that can completely violate the common prefix property. Our new combinatorial analysis introduces a new concept of characteristic strings and "forks" that reflects silent periods in protocol execution and network delays. To bound the density of forkable strings in this Δ-semisynchronous setting we establish a syntactic reduction from Δ-semisynchronous characteristic strings to synchronous strings of [KRDO17] that preserves the structure of the forks they support. This is followed by a probabilistic analysis that controls the distortion caused by the reduction and concludes that Δ-semisynchronous forkable strings are rare. Finally, we control the effective power of adaptive adversaries in this setting with a stochastic dominance argument that permits us to carry out the analysis of the underlying blockchain guarantees (e.g, common prefix) with a single distribution that provably dominates all distributions on characteristic strings generated by adaptive adversaries. We remark that these arguments yield graceful degradation of the analysis as a function of network delays (Δ), in the sense that the effective stake of the adversary is amplified by a function of Δ.

The above combinatorial analysis is nevertheless only sufficient to provide a proof of the static stake case, i.e., the setting where the stake distribution that facilitates the honest majority assumption remains fixed at the onset of the computation and prior to the selection of the random genesis data that are incorporated in the genesis block. For a true proof-of-stake system, we should permit the set of stakeholders to evolve over time and adapt our honest stakeholder majority assumption. Achieving this requires a bootstrapping argument that allows the protocol to continue ad infinitum by revising its stakeholder distribution as it evolves. We bootstrap our protocol in two conceptual steps. First we show how bootstrapping is possible if a randomness beacon is available to all participants. The beacon at regular intervals emits a new random value and the participants can reseed the election process so the stakeholder distribution used for sampling could be brought closer to the one that is current. A key observation here is that our protocol is resilient even if the randomness beacon is weakened in the following two ways: (i) it leaks its value to the adversary ahead of time by a bounded number of time units, (ii) it allows the adversary to reset its value if it wishes within a bounded time window. We call the resulting primitive a "leaky resettable beacon" and show that our bootstrapping argument still holds in this stronger adversarial setting.

In the final refinement of our protocol, we show how it is possible to implement the leaky resettable beacon via a simple algorithm that concatenates the VRF outputs that were contributed by the participants from the blockchain and passes them via a hash function that is modeled as a random oracle. This implementation explains the reasons behind the beacon relaxation we introduced: leakiness stems from the fact that the adversary can complete the blockchain segment that determines the beacon value before revealing it to the honest participants, while resettability stems from the fact that the adversary can try a bounded number of different blockchain extensions that will stabilize the final beacon value to a different preferred value.

Putting all the above together, we show how our protocol provides a "robust transaction ledger" in the sense that an immutable record of transactions is built that also guarantees that new transactions will be always included. Our security definition is in the Δ-semisynchronous setting with full adaptive corruptions. As mentioned above, security degrades gracefully as Δ increases, and this parameter is unknown to the protocol participants.

Comparison to previous works. The idea of proof-of-stake protocols has been discussed extensively in the bitcoin forum. The concept of using a publicly verifiable test that is computed by each participant locally to determine eligibility was realized in a number of cases, (e.g., NXT is a notable cryptocurrency that early on adopted this approach). The benefits of using a verifiable random function to implement this test[1](that we also use in our protocol) were also put forth and implemented in NXT, with the same motivation as in ours: to control adaptive corruptions; nevertheless none of these early proposals included a formal security model and adversarial analysis of their claims, even though a number of relevant attacks such as "grinding attacks" were identified. Injecting high quality randomness in the PoS blockchain was proposed by Bentov et al. [BLMR14, BGM16], though the proposal does not have a full formal analysis. The Ouroboros proof-of-stake protocol [KRDO17] is provably secure; nevertheless, the corruption model used excludes full adaptive attacks by imposing a corruption delay on the corruption requests of the adversary. The Snow White proof-of-stake [DPS16] is the first protocol to prove security in the Δ-semi-synchronous model but, as in the case of Ouroboros, adopts a weak adaptive corruption model. It is worth noting that Algorand [Mic16]

[1]More precisely in the form $H(\mathsf{Sign}(B.\mathsf{value})) < T$ where $\mathsf{Sign}(\cdot)$ is the signature algorithm of the stakeholder, $B.\mathsf{value}$ a certain value of the previous block, and T a bound that depends on the signer's stake, the current time and other parameters. Note that NXT does not use the terminology of VRFs.

provides a proof-of-stake ledger that is adaptively secure; on the other hand, it employs a byzantine agreement protocol for every block. While the Algorand agreement protocol is very efficient, it still can only produce an inherently slower blockchain compared to an "eventual consensus" protocol like the one we present that can effectively produce a blockchain that advances at the theoretical maximum speed (one block per round). Finally, Sleepy consensus [PS16] puts forth a technique for handling adaptive corruptions in a model that also encompasses fail-stop and recover corruptions; however, the protocol can be applied directly only in a static stake setting. We note that in fact our protocol can be also proven secure in such mixed corruption setting, where both fail-stop and recover as well as byzantine corruptions are allowed (with the former occurring at an arbitrarily high rate); nevertheless this is out of scope for the present exposition and we omit further details. In the present exposition we also put aside the issue of incentives; nevertheless, it is straightforward to adapt the mechanism of input endorsers from protocol of [KRDO17] to our setting and its approximate Nash equilibrium analysis can be ported directly.

2 Preliminaries

We say a function $negl(x)$ is negligible if for every $c > 0$, there exists an $n > 0$ such that $negl(x) < 1/x^c$ for all $x \geq n$. The length of a string w is denoted by $|w|$; ε denotes the empty string. For two strings v, w we use $v \,\|\, w$ to denote their concatenation. Given a string w and two integers $1 \leq a \leq b \leq |w|$, we denote by $w_{a:b}$ the substring consisting of symbols in w on positions $a, a+1, \ldots, b$.

2.1 Transaction Ledger Properties

We adopt the same definitions for transaction ledger properties as stated in Ouroboros [KRDO17].

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 [KP15, PSS17] 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 $\mu \in (0, 1]$ and $k \in \mathbb{N}$.** Consider any portion of length at least k 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.

2.2 The Semi-Synchronous Model

On a high level, we consider the security model of [KRDO17] with a simple modification to account for adversarially-controlled message delays and immediate adaptive corruption. Namely, we allow the adversary $\mathcal{A}$ to selectively delay any messages sent by honest parties for up to $\Delta \in \mathbb{N}$ slots; and corrupt parties without delay.

We now give a description of all the differences of our model from the model of [KRDO17]. A self-contained description of the model can be obtained in conjunction with Appendix A.

Two important functionalities considered here are the Delayed Diffuse functionality and the Key and Transaction functionality, which are defined as follows.

Delayed Diffuse Functionality. This functionality is parametrized by $\Delta \in \mathbb{N}$. It is denoted as $\mathsf{DDiffuse}_\Delta$ and is defined exactly as the functionality $\mathsf{Diffuse}$ given in [KRDO17], except for two differences:

1. When the adversary $\mathcal{A}$ is activated, besides performing any of the actions that were allowed by the $\mathsf{Diffuse}$ functionality, it is also allowed to:
   - For any message m obtained via a diffuse request and any party U_i, $\mathcal{A}$ may move m into a special string $\mathsf{delayed}_i$ instead of the inbox of U_i. $\mathcal{A}$ can decide this individually for each message and each party.
   - For any party U_i, $\mathcal{A}$ can move any message from the string $\mathsf{delayed}_i$ to the inbox of U_i.
2. At the end of each round, the functionality also ensures that for every message that was either (a) diffused in this round and not put to the string $\mathsf{delayed}_i$ or (b) removed from the string $\mathsf{delayed}_i$ in this round, it is present in the inbox of party U_i. If any message currently present in $\mathsf{delayed}_i$ was originally diffused at least Δ slots ago, then the functionality removes it from $\mathsf{delayed}_i$ and appends it to the inbox of party U_i.
3. Upon receiving (Create, U) from the environment, the functionality spawns a new user without providing it an intial chain $\mathcal{C}$ as it was the case in [KRDO17].

Key and Transaction Functionality. This functionality manages user keys and transactions. It is defined exactly as in [KRDO17], with one exception: it allows for instant corruptions of parties, i.e., the delay D from [KRDO17] is set to $D = 0$.

Restrictions on the Environment. Similarly to [KRDO17] we apply some restrictions to the environment in all executions but remark that these restrictions are weaker. As in [KRDO17], we require that at least one honest stakeholder is activated at each slot and that, in every slot, the adversary does not control more than 50% of the stake in the view of any honest stakeholder. If this is violated, an event $\mathsf{Bad}^{\frac{1}{2}}$ becomes true for the given execution. However, in contrast to [KRDO17], we do not require that honest stakeholders are activated at least once every k slots and the environment determine the delay between activations. 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 $Q \vee \mathsf{Bad}^{\frac{1}{2}}$ holds with high probability over all executions. This captures the fact that we exclude environments and adversaries that trigger $\mathsf{Bad}^{\frac{1}{2}}$ with non-negligible probability.

Random Oracle. In some of our results, we also assume the availability of a random oracle. As usually, this is a function $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^w$ available to all parties that answers every fresh query with an independent, uniformly random string from $\{0,1\}^w$, while any repeated queries are answered consistently.

Erasures. We assume that honest users can do secure erasures, which is argued to be a reasonable assumption in protocols with security against adaptive adversaries, see e.g., [Lin09].

2.3 Verifiable Random Functions

Informally, a Verifiable Random Function (VRF) is a pseudorandom function that produces non-interactive and publicly verifiable proofs of correctness of its outputs. Given an input x, a prover who possesses a secret key $\mathsf{VRF}.sk$ can compute a pseudorandom output $y = \mathsf{F}_{\mathsf{VRF}.sk}(\cdot)$ along with a proof $\pi_{\mathsf{VRF}.sk}(x)$ that allows a verifier to check that y is indeed the correct output of $\mathsf{F}_{\mathsf{VRF}.sk}(x)$ with respect to the corresponding public key $\mathsf{VRF}.pk$. VRFs are required to be unique, meaning that for every pair of input and public key there exists only one output and proof pair that is valid in relation to that public key; and pseudorandom, meaning that outputs are computationally indistinguishable from random strings of the same size even after many other outputs and accompanying proofs are revealed. VRFs were first introduced by Micali et al. [MRV99] and the first construction with constant key and proof sizes was proposed by Dodis et al. [DY05]. Formal definitions are presented in Appendix B.1.

2.4 Forward Secure Signatures Schemes

In regular digital signature schemes, an adversary who compromises the signing key of a user can generate signatures for any messages it wishes, including messages that were (or should have been) generated in the past. Forward secure signature schemes [BM99] prevent such an adversary from generating signatures for messages that were issued in the past, or rather allows honest users to verify that a given signature was generated at a certain point in time. Basically, such security guarantees are achieved by "evolving" the signing key after each signature is generated and erasing the previous key in such a way that the actual signing key used for signing a message in the past cannot be recovered but a fresh signing key can still be linked to the previous one. This notion is formalized through *key evolving signature schemes*, which allow signing keys to be evolved into fresh keys for a number of time periods. We remark that efficient constructions of key evolving signature schemes with forward security exist [IR01]. Formal definitions are presented in Appendix B.2.

3 The Static Case

We first consider the static case, where the stake distribution is fixed throughout protocol execution. The general structure of the protocol in the semi-synchronous model is very similar to that of synchronous Ouroboros [KRDO17] but requires two fundamental modifications to the leader selection process: not all slots will be attributed a slot leader, some slots might have more than one slot leader and slot leaders' identities remain unknown until they act. The first modification is used to deal with delays in the semi-synchronous network as the *empty slots* where no block is generated assist the honest parties to synchronize. The last modification is used to deal with adaptive corruptions, as it prevents the adversary from learning the slot leaders' identity ahead of time and using this knowledge to corrupt just enough of them in order to create arbitrary forks. Before describing the specifics of the new leader selection process and the new protocol, we first formally define the static stake scenario and introduce basic definitions as stated in [KRDO17] following the notation of [GKL15].

Setup: In the static stake case, we assume that a fixed collection of n stakeholders $U_1, \dots, U_n$ 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 $(\mathsf{KES}.vk_i, \mathsf{KES}.sk_{i,1})$ for R time periods (corresponding to the slots in the epoch) for a prescribed key evolving signature scheme are generated, a public and secret key pair $(\mathsf{VRF}.pk_i, \mathsf{VRF}.sk_i)$ for a VRF are generated and a stakeholder secret and public key pair $(\mathsf{pk}_i = (\mathsf{KES}.vk_i, \mathsf{VRF}.pk_i), \mathsf{sk}_i = (\mathsf{KES}.sk_{i,1}, \mathsf{VRF}.sk_i)$ is set; we assume without loss of generality that all the stakeholders know the public keys

$$\mathsf{pk}_1 = (\mathsf{KES}.vk_1, \mathsf{VRF}.pk_1), \dots, \mathsf{pk}_n = (\mathsf{KES}.vk_n, \mathsf{VRF}.pk_n)\,.$$

Definition 3.1 (Genesis Block)**.** *The* genesis block B_0 *contains the list of stakeholders identified by their public keys, their respective stakes* $(\mathsf{pk}_1, s_1), \dots, (\mathsf{pk}_n, s_n)$ *and a nonce* η*.*

With foresight we note that the nonce η will be used to seed the slot leader election process.

Definition 3.2 (State)**.** *A* state *is a string* $st \in \{0,1\}^\lambda$*.*

Definition 3.3 (Block Proof)**.** *A* block proof *is a value (or set of values)* B_π *containing information that allows stakeholders to verify if a block is valid.*

Definition 3.4 (Block)**.** *A block* B *generated at a slot* $sl_j \in \{sl_1, \dots, sl_R\}$ *contains the current state* $st \in \{0,1\}^\lambda$*, data* $d \in \{0,1\}^*$*, the slot number* sl_j*, a block proof* $B_{\pi j}$ *and* $\sigma_j = \mathsf{Sign}_{\mathsf{KES}.sk_{i,j}}((st, d, sl_j, B_{\pi j}))$*, a signature under the signing key for the time period of slot* sl_j *of the stakeholder* U_i *generating the block.*

We consider as valid blocks that are generated by a stakeholder in the slot leader set of the slot to which the block is attributed. Later on in this Section we discuss slot leader sets and how they are selected.

Definition 3.5 (Blockchain)**.** *A* blockchain *(or simply* chain*) relative to the genesis block* B_0 *is a sequence of blocks* $B_1, \dots, B_n$ *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 3.6 (Epoch)**.** *An* epoch *is a set of* R *adjacent slots* $S = \{sl_1, \dots, sl_R\}$*.*

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

Definition 3.7 (Absolute and Relative Stake)**.** *Let* $U_\mathcal{P}$*,* $U_\mathcal{A}$ *and* $U_\mathcal{H}$ *denote the sets of all stakeholders, the set of stakeholders controlled by an adversary* $\mathcal{A}$*, and the remaining (honest) stakeholders, respectively. For any party (resp. set of parties)* X *we denote by* s^+_X *(resp.* s^-_X*) the maximum (resp. minimum)* absolute stake *controlled by* X *in the view of all honest stakeholders at a given slot, and by* $\alpha^+_X \triangleq s^+_X / s_\mathcal{P}$ *and* $\alpha^-_X \triangleq s^-_X / s_\mathcal{P}$ *its* relative stake *taken as maximum and minimum respectively across of the view of all honest stakeholders. For simplicity, we use* $s^{\mathrm{s}}_X, \alpha^{\mathrm{s}}_X$ *instead of* s_{U_X}, α_{U_X} *for all* $X \in \{\mathcal{P}, \mathcal{A}, \mathcal{H}\}, \mathrm{s} \in \{+, -\}$*. We also call* $\alpha_\mathcal{A} \triangleq \alpha^+_\mathcal{A}$ *and* $\alpha_\mathcal{H} \triangleq \alpha^-_\mathcal{H}$ *the* adversarial stake ratio *and* honest stake ratio*, respectively.*

3.1 Oblivious Leader Selection and $\mathcal{F}_{\mathsf{INIT}}$

As in synchronous Ouroboros, for each $0 < j \leq R$, a *slot leader* E_j is a stakeholder who is elected to generate a block at sl_j. However, our leader selection process differs from synchronous Ouroboros [KRDO17] in three points: (1.) potentially, multiple slot leaders may be elected for a particular slot (forming a *slot leader set*), (2.) frequently, slots will have *no leaders* assigned to them, and (3.) a priori, only a slot leader is aware that it indeed a leader for a given slot; this assignment is unknown to all the other stakeholders—including other slot leaders of the same slot—until the other stakeholders receive a valid block from this slot leader. The combinatorial analysis presented in §4 shows (with an adaptively-determined honest stake majority) that (i.) forks generated according to these dynamics are well-behaved even if multiple slot leaders are selected for a slot and that (ii.) sequences of slots with no leader provide sufficient stability for honest stakeholders to effectively synchronize. As a matter of terminology, we call slots with an associated nonempty slot leader set *active slots* and slots that are not assigned a slot leader *empty slots*.

The fundamental leader assignment process calls for a stakeholder U_i to be selected as a member of the slot leader set for a particular slot sl_j with probability p_i depending on its stake registered in the genesis block B_0 and an *active slots coefficient*; these assignments are independent between slots. A precise description follows:

Definition 3.8 (Slot Leader Set and Active Slots Coefficient)**.** *A* slot leader set *for slot sl_j with respect to stakeholder distribution* $(\mathsf{pk}_1, s_1), \ldots, (\mathsf{pk}_n, s_n)$ *and* active slots coefficient f *is a set $\mathbb{L}_j$ such that, for all $sl_j \in \{sl_1, \ldots, sl_R\}$, each stakeholder $U_i \in \{U_1, \ldots, U_n\}$ is independently selected to be in $\mathbb{L}_j$ (i.e. $U_i \in \mathbb{L}_j$) with probability*

$$p_i = \phi_f(\alpha_i) \triangleq 1 - (1-f)^{\alpha_i} ,$$

where α_i is the relative stake held by stakeholder U_i; furthermore the family of random variables $\mathbb{L}_j$ are independent.

We sometimes drop the subscript f and write $\phi(\alpha_i)$ when f can be inferred from context.

Remarks about $\phi_f(\cdot)$. Observe that $\phi_f(1) = f$; in particular, the parameter f is the probability that a party holding all the stake will be selected to be a leader for given slot. On the other hand, $\phi_f()$ is not linear, but slightly concave. (See Figure 5, in the next section.) To motivate the choice of the function ϕ_f, we note that it satisfies the "independent aggregation" property:

$$1 - \phi\left(\sum_i \alpha_i\right) = \prod_i (1 - \phi(\alpha_i)) \, . \tag{1}$$

In particular, when leadership is determined according to ϕ_f, the probability of a stakeholder becoming a slot leader in a particular slot is independent of whether this stakeholder acts as a single party in the protocol, or splits its stake among several "virtual" parties. In particular, consider a party U with relative stake α who contrives to split its stake among two virtual subordinate parties with stakes α_1 and α_2 (so that $\alpha_1 + \alpha_2 = \alpha$). Then the probability that one of these virtual parties is elected for a particular slot is $1 - (1 - \phi(\alpha_1))(1 - \phi(\alpha_2))$, as these events are independent. Property (1) guarantees that this is identical to $\phi(\alpha)$. *Thus this selection rule is invariant under arbitrary reapportionment of a party's stake among virtual parties.*

In the static stake case, the genesis block is determined by an ideal initialization functionality $\mathcal{F}_{\mathsf{INIT}}$, defined in Figure 1. Notice that this functionality incorporates the diffuse and key/transaction functionality apart from providing the stake distribution and random nonce data to be included in the genesis block.

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

$\mathcal{F}_{\mathsf{INIT}}$ incorporates the diffuse and key/transaction functionality from Section 2.2 and is parameterized by the public keys and respective stakes of the initial stakeholders

$$\mathbb{S}_0 = \{(\mathsf{pk}_1 = (\mathsf{KES}.vk_1, \mathsf{VRF}.pk_1), s_1), \ldots, (\mathsf{pk}_n(\mathsf{KES}.vk_n, \mathsf{VRF}.pk_1), s_n)\}\,.$$

In addition, $\mathcal{F}_{\mathsf{INIT}}$ interacts with stakeholders $U_1, \ldots, U_n$ as follows:

- Upon receiving the first request of the form $(\mathsf{genblock\_req}, U_i)$ from some stakeholder U_i, $\mathcal{F}_{\mathsf{INIT}}$ samples a random nonce $\eta \xleftarrow{\$} \{0,1\}^\lambda$ and sends $(\mathsf{genblock}, \mathbb{S}_0, \eta)$ to U_i. Later requests from any stakeholder are answered using the same value η.

Figure 1: Functionality $\mathcal{F}_{\mathsf{INIT}}$.

3.2 VRFs with Unpredictability Under Malicious Key Generation

The usual pseudorandomness definition for VRFs (as stated in Definition B.1) captures the fact that an attacker who sees a number of VRF outputs and proofs for a number of adversarially chosen inputs generated under a secret and public key pair that is correctly generated by a challenger cannot distinguish the output of the VRF on a new (also adversarially chosen) input from a truly random string. However, this definition does not take into consideration malicious key generation, *i.e.,* adversaries who are allowed to generate the secret and pubic key pair used in the pseudorandomness experiment. If the adversary is allowed to generate its own key pair, there is no guarantee that it cannot influence the distribution of the VRF outputs. In fact, for some known constructions (*e.g.* [DY05]), an adversary that maliciously generates keys can easily and significantly skew the output distribution. In order to implement oblivious slot leader set selection in our protocols, we need a VRF for which the output distribution is not affected by maliciously chosen key pairs given that the inputs have sufficient min-entropy. We call this property *unpredictability under malicious key generation* and formally define it as follows.

Definition 3.9 (Unpredictability Under Malicious Key Generation)**.** *For an input distribution* D *and any PPT algorithm* $A = (A_K, A_J)$ *that runs for a total of* $s(k)$ *steps when its first input is* 1^k*,*

$$Pr\left[b = b' \left| \begin{array}{l} (\mathsf{VRF}.pk, \mathsf{VRF}.sk, A_{st}) \leftarrow A_K(1^k); \\ x \leftarrow \mathsf{D}; y_0 = \mathsf{F}_{\mathsf{VRF}.sk}(x); \\ y_1 \leftarrow \{0,1\}^{\ell_{\mathsf{VRF}}}; b \leftarrow \{0,1\}; \\ b' \leftarrow A_J(y_b, A_{st}) \end{array} \right. \right] \le \frac{1}{2} + negl(k)$$

The above definition is impossible to achieve for any distribution D but we will show that any standard VRF (*i.e.*, one that matches Definition B.1) can be transformed into a VRF with unpredictability under malicious key generation for distributions D with high min-entropy in the Random Oracle Model. Let $\mathsf{H}(\cdot)$ be a random oracle, $\mathsf{F}.(\cdot) : \{0,1\}^\ell \to \{0,1\}^{\ell_{\mathsf{VRF}}}$ be a family of VRFs with algorithms $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ with standard security as stated in Definition B.1 and D be an input distribution. We construct a family of VRFs with unpredictability under malicious key generation $\mathsf{UF}.(\cdot) : \{0,1\}^\ell \to \{0,1\}^{\ell_{\mathsf{VRF}}}$ with algorithms $(\mathsf{UGen}, \mathsf{UProve}, \mathsf{UVer})$ that works as follows:

- $\mathsf{UGen}(1^k)$ outputs a pair of keys $(\mathsf{UVRF}.pk, \mathsf{UVRF}.sk) \leftarrow \mathsf{Gen}(1^k)$.

Protocol π_{SPoS}

Let $\mathsf{H}(\cdot)$ be a random oracle, $\mathsf{F}_{\cdot}(\cdot) : \{0,1\}^{\ell} \rightarrow \{0,1\}^{\ell_{\mathsf{VRF}}}$ be a family of unpredictable under malicious key generation VRFs with algorithms $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ and $\mathsf{KES} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update})$ be a forward secure key evolving signature scheme. Define $T_i = 2^{\ell_{\mathsf{VRF}}} \phi_f(\alpha_i)$ as the threshold of a stakeholder U_i, where ℓ_{VRF} is the length in bits of the VRF output, f is the active slots coefficient and ϕ_f is the mapping from Definition 3.8. π_{SPoS} is a protocol run by stakeholders $U_1, \ldots, U_n$ interacting among themselves and with $\mathcal{F}_{\mathsf{INIT}}$ 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 key $\mathsf{pk}_i = (\mathsf{KES}.vk_i, \mathsf{VRF}.pk_i)$ and secret key $\mathsf{sk}_i = (\mathsf{KES}.sk_{i,1}, \mathsf{VRF}.sk_i)$. Then it receives the current slot from the diffuse interface and sends $(\mathsf{genblock\_req}, U_i)$ to $\mathcal{F}_{\mathsf{INIT}}$, receiving $(\mathsf{genblock}, \mathbb{S}_0, \eta)$ as answer. U_i sets the local blockchain $\mathcal{C} = B_0 = (\mathbb{S}_0, \eta)$ and the initial internal state $st = H(B_0)$.
2. **Chain Extension** For every slot $sl_j \in S$, every online stakeholder U_i performs the following steps:
   (a) Collect all valid chains received via broadcast into a set $\mathbb{C}$, pruning blocks belonging to future slots and verifying that for every chain $\mathcal{C}' \in \mathbb{C}$ and every block $B' = (st', d', sl', {B_\pi}', \sigma_{j'}) \in \mathcal{C}'$ it holds that the stakeholder U' is in the slot leader set $\mathbb{L}'$ of slot sl' (by parsing ${B_\pi}'$ as (pk', y', π'), verifying that $\mathsf{Ver}_{\mathsf{VRF}.pk'}(\eta \,\|\, sl', y', \pi') = 1$, and that $y' < T'$), $\mathsf{Verify}_{\mathsf{KES}.vk'}((st', d', sl', {B_\pi}'), \sigma_{j'}) = 1$ and that the signature $\sigma_{j'}$ is for the the time period that corresponds to 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) U_i checks whether it is in the slot leader set $\mathbb{L}_j$ of slot sl_j by checking that $y < T_i$, where $(y, \pi) \leftarrow \mathsf{Prove}_{\mathsf{VRF}.sk_i}(\eta \,\|\, sl_j)$. If yes, it generates a new block $B = (st, d, sl_j, B_{\pi j}, \sigma_j)$ where st is its current state, $d \in \{0,1\}^*$ is the transaction data, $B_\pi = (\mathsf{pk}_i, y, \pi)$ and $\sigma_j = \mathsf{Sign}_{\mathsf{KES}.sk_{i,j}}(st, d, sl_j, B_{\pi j})$ is a signature on $(st, d, sl_j, B_{\pi j})$ for slot sl_j. U_i computes $\mathcal{C}' = \mathcal{C}|B$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\mathrm{head}(\mathcal{C}'))$.
   (c) Execute $\mathsf{Update}(\mathsf{KES}.sk_{i,j})$ obtaining the signing key $\mathsf{KES}.sk_{i,j+1}$ for the next slot, erases the current signing key $\mathsf{KES}.sk_{i,j}$. Finally, if U_i has generated a block in the previous step, it broadcasts $\mathcal{C}'$.

Figure 2: Protocol π_{SPoS}.

- $\mathsf{UProve}_{\mathsf{UVRF}.sk}(x)$ outputs a pair $(\mathsf{UF}_{\mathsf{UVRF}.sk}(x), \mathsf{U}.\pi_{\mathsf{UVRF}.sk}(x))$, where

$$\mathsf{UF}_{\mathsf{UVRF}.sk}(x) = (\mathsf{H}(x \,\|\, \mathsf{F}_{\mathsf{UVRF}.sk}(x)), \mathsf{F}_{\mathsf{UVRF}.sk}(x))\,,$$
$$\mathsf{U}.\pi_{\mathsf{UVRF}.sk}(x) = \pi_{\mathsf{UVRF}.sk}(x)\,, \quad \text{and}$$
$$(\mathsf{F}_{\mathsf{UVRF}.sk}(x), \pi_{\mathsf{UVRF}.sk}(x)) \leftarrow \mathsf{Prove}_{\mathsf{UVRF}.sk}(x)\,.$$

- $\mathsf{UVer}_{\mathsf{VRF}.pk}(x, y, \mathsf{U}.\pi_{\mathsf{UVRF}.sk}(x))$ parses y as (y_1, y_2) and outputs 1 if

$$y_1 = \mathsf{H}(x \,\|\, y_2) \qquad \text{and} \qquad \mathsf{Ver}_{\mathsf{UVRF}.pk}(x, y_2, \mathsf{U}.\pi_{\mathsf{UVRF}.sk}(x)) = 1\,.$$

  Otherwise, it outputs 0.

Theorem 3.10. *Let $\mathsf{H}(\cdot)$ be a random oracle, $\mathsf{F}.(\cdot) : \{0,1\}^{\ell} \to \{0,1\}^{\ell_{\mathsf{VRF}}}$ be a family of VRFs with algorithms $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ with standard security as stated in Definition B.1 and D be an input distribution with k bits of min-entropy. The family of VRFs $\mathsf{UF}.(\cdot) : \{0,1\}^{\ell} \to \{0,1\}^{\ell_{\mathsf{VRF}}}$ with algorithms $(\mathsf{UGen}, \mathsf{UProve}, \mathsf{UVer})$ constructed above is secure with respect to Definition B.1 and achieves unpredictability under malicious key generation as stated in Definition 3.9.*

The proof is given in Appendix D.

3.3 The Protocol in the $\mathcal{F}_{\mathsf{INIT}}$-hybrid model

We will construct our protocol for the static case in the $\mathcal{F}_{\mathsf{INIT}}$-hybrid model, where the genesis stake distribution $\mathbb{S}_0$ and the nonce η (to be written in the genesis block B_0) are determined by the ideal functionality $\mathcal{F}_{\mathsf{INIT}}$. Moreover $\mathcal{F}_{\mathsf{INIT}}$ also incorporates the diffuse and key/transaction functionality from Section 2.2, which sets stakeholders' public and secret key pairs and diffuses messages among stakeholders. The stakeholders $U_1, \ldots, U_n$ interact among themselves and with $\mathcal{F}_{\mathsf{INIT}}$ 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.

> 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.

We describe protocol π_{SPoS} in terms of a forward secure key evolving signature scheme KES and a VRF with unpredictability under malicious key generation $\mathsf{F}.(\cdot) : \{0,1\}^{\ell} \to \{0,1\}^{\ell_{\mathsf{VRF}}}$, which we instantiate via constructions proven secure through game based arguments. However, in the security analysis presented in the next section, we refer to π_{iSPoS}, an *idealized* version of π_{SPoS} where these primitives are treated as ideal functionalities. This allows us to focus attention on the combinatorial and stochastic arguments relevant for establishing the structural properties of the resulting blockchain. We remark that the forward secure signature scheme and VRF with unpredictability under malicious key generation we use are variations of standard signature schemes and VRFs, which have been shown to realize corresponding ideal functionalities [Can04, CL07]. Standard UF-CMA signature schemes are shown to be UC secure in [Can04], while [CL07] introduces the notion of simulatable VRFs, which can be cast as an ideal functionality realizable in the random oracle model (adopted in this paper). Generic constructions of forward secure key evolving signature schemes from any UF-CMA secure signature scheme that could be used to realize an ideal functionality for this primitive are shown in [MMM02], while our generic construction of VRFs with unpredictability under malicious key generation operates in the random oracle model, which is amenable to simulation based arguments.

4 Combinatorial Analysis of the Idealized Protocol

The idealized protocol π_{iSPoS} yields a stochastic process for assigning slots to parties which we abstract and study here in detail. Our analysis of the resulting blockchain dynamics proceeds roughly as follows: We begin by generalizing the framework of "forks" [KRDO17] to our semi-synchronous setting—this is a natural bookkeeping tool that reflects the chains possessed by honest players during an execution of the protocol. We then establish a simulation rule that associates with each execution of the semi-synchronous protocol an execution of a related "virtual" synchronous protocol. Motivated by the setting with a *static* adversary—which simply corrupts a family of parties at the outset of the protocol—we identify a natural "generic" probability distribution for this simulation theorem which we prove controls the behavior of adaptive adversaries by stochastic domination. Finally, we prove that this simulation amplifies the effective power of the adversary in a controlled fashion and, furthermore, permits forks of the semi-synchronous protocol to be projected to forks of the virtual protocol in a way that preserves their relevant combinatorial properties. This allows us to apply the density theorems and divergence result of [KRDO17, RMKQ17] to provide strong common prefix (§4.5), chain quality (§4.7), and chain growth (§4.6) guarantees for the semi-synchronous protocol with respect to an adaptive adversary.

We begin in §4.1 with a discussion of characteristic strings, semi-synchronous forks, and their relationship to executions of π_{iSPoS}. §4.2 then develops the combinatorial reduction from the semi-synchronous to the synchronous setting. The "generic, dominant" distribution on characteristic strings is then motivated and defined in §4.3, where the effect of the reduction on this distribution is also described. Sections §4.5–4.6, as described above, establish various guarantees on the resulting blockchain under the dominant distribution. The connection to adaptive adversaries is established in §4.8. Finally, in preparation for applying the idealized protocol in the dynamic stake setting, we formulate a "resettable setting" which further enlarges the power of the adversary by providing some control over the random nonce that seeds the protocol.

4.1 Chains, Forks and Divergence

We begin by suitably generalizing the framework of characteristic strings, forks, and divergence developed in [KRDO17] to our semi-synchronous setting. The leader assignment process given by protocol π_{iSPoS} assigns leaders to slots with the following guarantees: *(i.)* a party with relative stake α becomes a slot leader for a given slot with probability $\phi_f(\alpha) \triangleq 1-(1-f)^{\alpha}$; *(ii.)* the event of becoming a slot leader is independent for each party and for each slot. Clearly, these dynamics may lead to slots with multiple slot leaders and, likewise, slots with no slot leader. For a given adaptive adversary $\mathcal{A}$ and environment $\mathcal{Z}$, we reflect the outcome of this process with a *characteristic* string, as follows.

Definition 4.1 (Characteristic string). *Let $S = \{sl_1, \ldots, sl_R\}$ be a sequence of slots of length R and $\mathcal{A}$ an adversary. For a slot sl_i, let $\mathcal{P}(i)$ denote the set of parties assigned to slot i by the process above. We define the* characteristic string *$w \in \{0,1,\perp\}^R$ of S to be the random variable so that*

$$w_i = \begin{cases} \perp & \text{if } \mathcal{P}(i) = \emptyset, \\ 0 & \text{if } |\mathcal{P}(i)| = 1 \text{ and the assigned party is honest,} \\ 1 & \text{if } |\mathcal{P}(i)| > 1 \text{ or a party in } \mathcal{P}(i) \text{ is adversarial.} \end{cases} \tag{2}$$

We emphasize that in the setting of π_{iSPoS}, the appropriate characteristic string is determined by both the nonce (and the effective leader selection process), the adaptive adversary $\mathcal{A}$, and the environment $\mathcal{Z}$ (which, in particular, determines the stake distribution). For such a characteristic *string $w \in \{0, 1, \perp\}^*$ we say that the index i is* uniquely honest *if $w_i = 0$,* tainted *if $w_i = 1$, and* empty *if $w_i = \perp$. We say that an index is* active *if $w_i \in \{0, 1\}$. Note that an index is "tainted" according to this terminology in cases where multiple honest parties (and no adversarial party) have been assigned to it.*

We denote by $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$ the distribution of the random variable $w = w_1 \dots w_R$ in the experiment with the active slots coefficient f, adversary $\mathcal{A}$, and environment $\mathcal{Z}$.

The notion of "fork", defined in [KRDO17], is a bookkeeping tool that indicates the chains broadcast by honest players during an idealized execution of a blockchain protocol. We now adapt the synchronous notion of [KRDO17] to reflect the effect of message delays.

From executions to forks. An execution of Protocol π_{iSPoS} induces a collection of blocks broadcast by the participants. As we now focus merely on the structural properties of the resulting blockchain, for each broadcast block we now retain only two features: the slot during which it was broadcast and the previous block to which it is "attached" by the idealized digital signature σ_j. (Of course, we only consider blocks with legal structure that meet the idealized verification criteria of π_{iSPoS}.) Note that multiple blocks may be broadcast during a particular slot, either because multiple parties are assigned to the slot or an adversarial party is assigned to a slot. In any case, these blocks induce a natural directed tree by treating the legal broadcast blocks as vertices and introducing a directed edge between each pair of blocks (b, b') for which b' identifies b as the previous block. In the Δ-semisynchronous setting, the maxvalid rule enforces a further critical property on this tree: the depth of any block broadcast by an honest player during the protocol must exceed the depths of any honestly-generated blocks from slots at least Δ in the past. (This follows because such previously broadcast blocks would have been available to the honest player, who always builds on a chain of maximal length.) We call a directed tree with these structural properties a Δ-*fork*, and define them precisely below.

We may thus associate with any idealized execution of π_{iSPoS} a fork. While this fork disregards many of the details of the execution, any violations of common prefix are immediately manifested by certain "viable" diverging paths in the fork. A fundamental element of our analysis relies on controlling the structure of the forks that can be induced in this way for a given characteristic string (which determines which slots have been assigned to uniquely honest parties). In particular, we prove that common prefix violations are impossible for "typical" characteristic strings generated by π_{iSPoS} with an adversary $\mathcal{A}$ by establishing that such diverging paths cannot exist in their associated forks. We then go on to study related structural properties of the blockchain.

Definition 4.2 (Δ-forks)**.** *Let $w \in \{0, 1, \perp\}^k$ and Δ be a non-negative integer. Let $A = \{i \mid w_i \neq \perp\}$ denote the set of active indices, and let $H = \{i \mid w_i = 0\}$ denote the set of uniquely honest indices. A* Δ-fork *for the string w is a directed, rooted tree $F = (V, E)$ with a labeling $\ell : V \to \{0\} \cup A$ so that*

(i) the root $r \in V$ is given the label $\ell(r) = 0$;

(ii) each edge of F is directed away from the root;

(iii) the labels along any directed path are strictly increasing;

(iv) each uniquely honest index $i \in H$ is the label of exactly one vertex of F;

(v) the function $\mathbf{d} : H \to \{1, \dots, k\}$, defined so that $\mathbf{d}(i)$ is the depth in F of the unique vertex v for which $\ell(v) = i$, satisfies the following Δ-monotonicity *property: if $i, j \in H$ and $i + \Delta < j$, then $\mathbf{d}(i) < \mathbf{d}(j)$.*

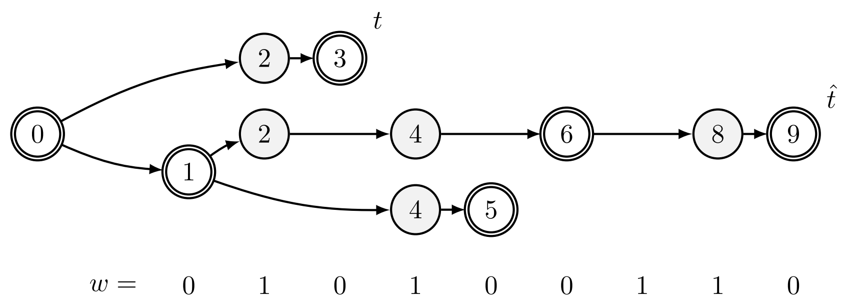
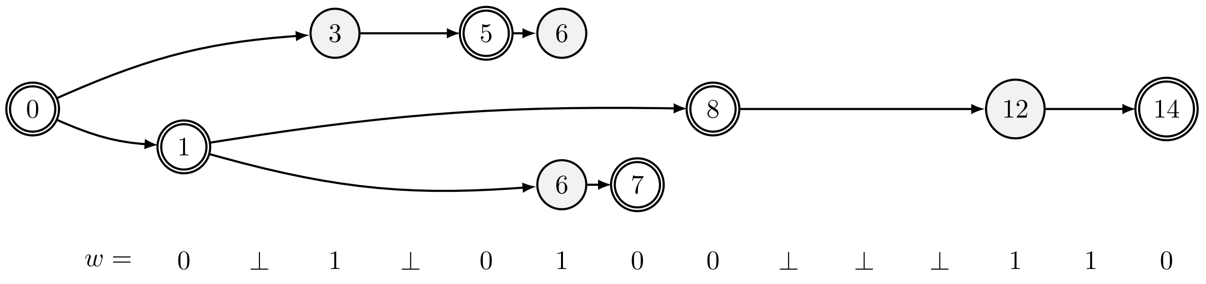


Figure 3: A (synchronous) fork F for the string $w = 010100110$. Vertices appear with their labels and vertices belonging to (uniquely) honest slots are highlighted with double borders. Note that the depths of the (honest) vertices associated with the honest indices of w are strictly increasing. Two 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 divergence of t and $\hat{t}$ is $\mathrm{div}(t, \hat{t}) = 2$.



0
1
3
5
6
8
12
14
6
7
$w =$ 0 $\perp$ 1 $\perp$ 0 1 0 0 $\perp$ $\perp$ $\perp$ 1 1 0

Figure 4: A 3-fork F' for the characteristic string $w = 0\perp1\perp01001\perp\perp10$. Note that F' is not a 2-fork since $\mathbf{d}(8) = 2 \not> 2 = \mathbf{d}(5)$. Indices $\{1, 5, 7, 8, 14\}$ are uniquely honest, $\{3, 6, 12, 13\}$ are tainted, and $\{2, 4, 9, 10, 11\}$ are empty. The index 8 is 4-right-isolated, but not 5-right-isolated.

As a matter of notation, we write $F \vdash_\Delta w$ to indicate that F is an Δ-fork for the string w. When there is no risk of confusion, we refer to a Δ-fork as simply a "fork".

See Figures 3 and 4 for examples.

Definition 4.3 (Tines, length, and viability)**.** *A path in a fork F originating at the root is called a* tine*. For a tine t we let* length(t) *denote its* length*, equal to the number of edges on the path. For a vertex v, we let* depth(v) *denote the length of the tine terminating at v. For convenience, we overload the notation $\ell(\cdot)$ so that it applies to tines by defining $\ell(t) \triangleq \ell(v)$, where v is the terminal vertex on the tine t.*

We say that a tine t is Δ-viable *if*

$$\mathrm{length}(t) \geq \max_{\substack{\text{uniquely honest } h \\ h+\Delta \leq \ell(t)}} \mathbf{d}(h)\,,$$

this maximum extended over all uniquely honest indices h appearing Δ or more slots before $\ell(t)$. Note that any tine terminating in a uniquely honest vertex is necessarily viable by the Δ-monotonicity property.

Remarks on viability and divergence. The notion of viability, defined above, demands that the length of a tine t be no less than that of all tines broadcast by uniquely honest slot leaders prior to slot $\ell(t) - \Delta$. Observe that such a tine could, in principle, be selected according to the $\mathsf{maxvalid}()$ rule by an honest player online at time $\ell(t)$: in particular, if all blocks broadcast by honest parties in slots $\ell(t) - \Delta, \ldots, \ell(t)$ are maximally delayed, the tine can favorably compete with all other tines that the adversary is obligated to deliver by slot $\ell(t)$. The major analytic challenge, both in the synchronous case and in our semisynchronous setting, is to control the possibility of a *common prefix* violation, which occurs when the adversary can manipulate the protocol to produce a fork with two viable tines with a relatively short common prefix. We define this precisely by introducing the notion of divergence.

Definition 4.4 (Divergence)**.** *Let F be a Δ-fork for a string $w \in \{0, 1, \perp\}^*$. For two Δ-viable tines t_1 and t_2 of F, define their* divergence *to be the quantity*

$$\operatorname{div}(t_1, t_2) \triangleq \min\{\operatorname{length}(t_1), \operatorname{length}(t_2)\} - \operatorname{length}(t_1 \cap t_2)\,,$$

where $t_1 \cap t_2$ denotes the common prefix of t_1 and t_2. We extend this notation to the fork F by maximizing over viable tines:

$$\operatorname{div}_\Delta(F) \triangleq \max_{\substack{t_1,t_2\ \Delta\text{-viable}\\ \text{tines of } F}} \operatorname{div}(t_1, t_2)\,.$$

Finally, we define the Δ-divergence of a characteristic string w to be the maximum over all Δ-forks:

$$\operatorname{div}_\Delta(w) \triangleq \max_{F \vdash_\Delta w} \operatorname{div}(F)\,.$$

Our primary goal in this section is to prove that, with high probability, the characteristic strings induced by protocol π_{iSPoS} have small divergence. In particular, such characteristic strings necessarily provide strong guarantees on common prefix.

4.1.1 The Synchronous Case

The original development of [KRDO17] assumed a strictly synchronous environment. Their definitions of characteristic string, fork, and divergence correspond to the case $\Delta = 0$, where characteristic strings are elements of $\{0, 1\}^*$. As this setting will play an important role in our analysis—fulfilling the role of the "virtual protocol" described at the beginning of this section—we set down some further terminology for this synchronous case and establish a relevant combinatorial statement based on a result in [KRDO17] that we will need for our analysis.

Definition 4.5 (Synchronous characteristic strings and forks)**.** *A* synchronous characteristic string *is an element of $\{0, 1\}^*$. A* synchronous fork *F for a (synchronous) characteristic string w is a 0-fork $F \vdash_0 w$.*

An immediate conclusion of the results obtained in [KRDO17, RMKQ17] is the following bound on the probability that a synchronous characteristic string drawn from the binomial distribution has large divergence.

Theorem 4.6. *Let $\ell, k \in \mathbb{N}$ and $\epsilon \in (0, 1)$. Let $w \in \{0, 1\}^\ell$ be drawn according to the binomial distribution, so that* $\Pr[w_i = 1] = (1 - \epsilon)/2$. *Then* $\Pr[\operatorname{div}_0(w) \geq k] \leq \exp(\ln \ell - \Omega(k))$.

A proof of a weaker bound of the form $\exp(\ln \ell - \Omega(\sqrt{k}))$ appears in [KRDO17]. Russell et al. [RMKQ17] then strengthened the basic probabilistic tools of [KRDO17] to achieve a bound of the form $\exp(\ln \ell - \Omega(k))$ for the local notion of *forkability*. For completeness, we include a proof of Theorem 4.6 relying on the results of [RMKQ17] in Appendix F.

4.2 The Semisynchronous to Synchronous Reduction

We will make use of the following mapping, that maps characteristic strings to synchronous characteristic strings.

Definition 4.7 (Reduction mapping)**.** *For $\Delta \in \mathbb{N}$, we define the function $\rho_\Delta \colon \{0,1,\perp\}^* \to \{0,1\}^*$ inductively as follows:*

$$\begin{aligned}
\rho_\Delta(\varepsilon) &= \varepsilon, \\
\rho_\Delta(\perp \parallel w') &= \rho_\Delta(w'), \\
\rho_\Delta(1 \parallel w') &= 1 \parallel \rho_\Delta(w'), \\
\rho_\Delta(0 \parallel w') &= \begin{cases} 0 \parallel \rho_\Delta(w') & \text{if } w' \in \perp^{\Delta-1} \parallel \{0,1,\perp\}^*, \\ 1 \parallel \rho_\Delta(w') & \text{otherwise.} \end{cases}
\end{aligned} \tag{3}$$

We call ρ_Δ the reduction mapping for delay Δ.

Note that for any $w \in \{0,1,\perp\}^*$ we have $|\rho_\Delta(w)| \leq |w|$, in particular $|\rho_\Delta(w)| = |w| - \#_\perp(w)$, where $\#_\perp(x)$ denotes the number of appearances of the symbol $\perp$ in x.

A critical feature of the map ρ_Δ is that it monotonically transforms Δ-divergence to synchronous divergence.

Lemma 4.8. *Let $w \in \{0,1,\perp\}^*$ be a characteristic string. Then* $\operatorname{div}_\Delta(w) \leq \operatorname{div}_0(\rho_\Delta(w))$.

Proof. Let $w \in \{0,1,\perp\}^*$ be a characteristic string with $\operatorname{div}_\Delta(w) = k$ and let $F \vdash_\Delta w$ be a Δ-fork with $\operatorname{div}_\Delta(F) = k$. Let $w' = \rho_\Delta(w)$; to prove that $\operatorname{div}_0(w') \geq k$, we construct a fork $F' \vdash_0 w'$ for which $\operatorname{div}(F') \geq k$. Let $A = \{i \mid w_i \neq \perp\}$ denote the set of active indices (as in Definition 4.2) and note that $|\rho_\Delta(w)| = |A|$; as noted above, each non-$\perp$ symbol of w corresponds to a unique symbol in w'. We let $\pi : A \to \{1, \ldots, |A|\}$ be the (bijective, increasing) function which records the position in w' corresponding to a particular active index i in w. Finally, we define the fork F' as follows: as a graph, F' has the same structure as F; the labeling ℓ' (for F') is given by the rule $\ell'(v) = \pi(\ell(v))$; of course, $\ell'(r) = 0$ for the root vertex r.

To verify that $F' \vdash_0 w' = \rho_\Delta(w)$, we recall the necessary properties from the definition. Properties (i) and (ii) of the Definition 4.2 are immediate; property (iii) follows because π is strictly increasing. For the remaining properties, we recall the definition of ρ_Δ: According the rule, $w_i = 1 \Rightarrow w'_{\pi(i)} = 1$ from which property (iv) follows immediately. It remains to check property (v). The value $w'_{\pi(i)}$ when $w_i = 0$ is determined by the $\Delta - 1$ following symbols of w: if $w_{i+1} = w_{i+2} = \cdots = w_{i+\Delta-1} = \perp$, we say that i is Δ-*right-isolated* (cf. [GKL17], where a similar feature arises in a proof-of-work setting) and in this case $w'_{\pi(i)} = 0$; otherwise $w'_{\pi(i)} = 1$. In particular, if $w'_{\pi(i)} = 0$ we must have $w_i = 0$ and $w_{i+s} = \perp$ for $0 \leq s < \Delta$. As we wish to conclude that F' is a synchronous fork, it must satisfy the Δ-monotonicity property with $\Delta = 0$, which is to say that $\mathbf{d}(\cdot)$ is strictly increasing on the set of uniquely honest indices (of w'). However, in light of the discussion above, any two uniquely honest indices of w' must correspond to uniquely honest indices of w separated by at least $\Delta - 1$ intervening $\perp$ symbols; thus the Δ-monotonicity property of F ensures the 0-monotonicity property of F', as desired.

In preparation for establishing that $\operatorname{div}_0(F') \geq \operatorname{div}(F) = k$, we note that a Δ-viable tine t of $F \vdash_\Delta w$ is 0-viable when viewed as a tine of $F' \vdash w'$. In particular, let h' be a uniquely honest index of w' for which $h' \leq \ell'(t)$ and let h be the uniquely honest index of w for which $\pi(h) = h'$. As $\pi(h)$ is uniquely honest in w', h is Δ-right isolated in w, and we conclude that $\operatorname{length}(t) \geq \mathbf{d}(h)$, because t is Δ-viable. This t is 0-viable in F'.

Finally, let t_1 and t_2 be two Δ-viable tines of F for which $\mathrm{div}_\Delta(t_1, t_2) = \mathrm{div}_\Delta(w)$. In light of the discussion above, these tines are 0-viable in F'; as the two forks have the structure as graphs, we conclude that $\mathrm{div}_0(w') \geq \mathrm{div}_\Delta(t_1, t_2) = \mathrm{div}_\Delta(w)$, as desired. $\square$

4.3 The Dominant Characteristic Distribution

The high-probability results for our desired chain properties depend on detailed information about the distribution on characteristic strings determined by an adversary $\mathcal{A}$, the environment $\mathcal{Z}$, and the parameters f and R. In this section we define a distinguished distribution on characteristic strings which we will see "dominates" the distributions produced by any adaptive adversary. We then study the effect of ρ_Δ on the distribution in preparation for studying common prefix, chain growth, and chain quality.

4.3.1 Motivating the Dominant Distribution: Static Adversaries

To motivate the dominant distribution, consider the distribution induced by a *static* adversary who corrupts—at the outset of the protocol—a set $U_\mathcal{A}$ of parties with total relative stake $\alpha_\mathcal{A}$. Recalling Definition 3.8, a party with relative stake α_i is independently assigned to be a leader for a slot with probability

$$\phi_f(\alpha_i) \triangleq \phi(\alpha_i) \triangleq 1 - (1-f)^{\alpha_i}\,.$$

As indicated, we drop the subscript f when it can be inferred from context. As discussed earlier, ϕ_f satisfies the "independent aggregation" property (1). Noting that

$$\frac{\partial^2 \phi_f}{\partial \alpha^2}(\alpha) = -(\ln(1-f))^2(1-f)^\alpha < 0$$

the function ϕ_f is concave; Figure 5 shows a plot of $\phi_{1/2}$ for illustration. Considering that $\phi_f(0) = 0$ and $\phi_f(1) = f$, concavity implies that $\phi_f(\alpha) \geq f\alpha$ for $\alpha \in [0,1]$. As $\phi_f(0) \geq 0$ and ϕ_f is concave, the function ϕ_f is subadditive. We record these properties:

$$\phi_f\left(\sum_i \alpha_i\right) = 1 - \prod_i (1 - \phi_f(\alpha_i)) \leq \sum_i \phi_f(\alpha_i)\,, \qquad \alpha_i \geq 0\,, \tag{4}$$

$$\frac{\phi_f(\alpha)}{\phi_f(1)} = \frac{\phi_f(\alpha)}{f} \geq \alpha\,, \qquad \alpha \in [0,1]\,. \tag{5}$$

Recalling Definition 4.1, this (static) adversary $\mathcal{A}$ determines a distribution $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$ on strings $w \in \{0,1,\perp\}^R$ by independently assigning each w_i so that

$$\begin{aligned}
p^\mathcal{A}_\perp &\triangleq \Pr[w_i = \perp] = \prod_{i\in\mathcal{P}} (1-\phi(\alpha_i)) = \prod_{i\in\mathcal{P}} (1-f)^{\alpha_i} = (1-f)\,,\\
p^\mathcal{A}_0 &\triangleq \Pr[w_i = 0] = \sum_{h\in\mathcal{H}} \phi(\alpha_h) \cdot \prod_{i\in\mathcal{P}\setminus\{h\}} (1-\phi(\alpha_i)) = \sum_{h\in\mathcal{H}} (1-(1-f)^{\alpha_h}) \cdot (1-f)^{1-\alpha_i}\,, \qquad (6)\\
p^\mathcal{A}_1 &\triangleq \Pr[w_i = 1] = 1 - p^\mathcal{A}_\perp - p^\mathcal{A}_0\,.
\end{aligned}$$

Here $\mathcal{H}$ denotes the set of all honest parties in the stake distribution $\mathcal{S}$ determined by $\mathcal{Z}$. As before, $\mathcal{P}$ denotes the set of all parties.

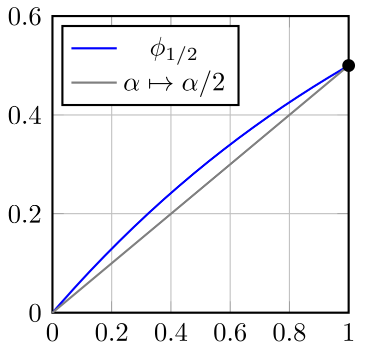


Figure 5: The function $\phi_{1/2}(\alpha) = 1-(1/2)^\alpha$ and the linear function $\alpha \mapsto \alpha/2$, for comparison. The point $(1, 1/2)$ is marked in solid black.

The dominant distribution. It is convenient to work with some bounds on the above quantities that depend only on "macroscopic" features of $\mathcal{S}$ and $\mathcal{A}$: namely, the relative stake of the honest and adversarial parties, and the parameter f. For this purpose we note that

$$p_0^{\mathcal{A}} \geq \sum_{h\in\mathcal{H}} \phi(\alpha_h) \cdot \prod_{i\in\mathcal{P}} (1-\phi(\alpha_i)) \geq \phi(\alpha_{\mathcal{H}}) \cdot p_\perp^{\mathcal{A}} = \phi(\alpha_{\mathcal{H}}) \cdot (1-f)\,, \tag{7}$$

where $\alpha_{\mathcal{H}}$ denotes the total relative stake of the honest parties. Note that this bound applies to all adversaries $\mathcal{A}$ that corrupt no more than a $1-\alpha_{\mathcal{H}}$ fraction of all stake. With this in mind, we define the distribution $\mathcal{D}_\alpha^f$ on strings $w \in \{0,1,\perp\}^R$ that independently assigns each w_i so that

$$\begin{aligned} p_\perp &\triangleq \Pr[w_i = \perp] = 1-f\,,\\ p_0 &\triangleq \Pr[w_i = 0] = \phi(\alpha)\cdot(1-f)\,,\\ p_1 &\triangleq \Pr[w_i = 1] = 1 - p_\perp - p_0\,. \end{aligned} \tag{8}$$

The distribution $\mathcal{D}_\alpha^f$ "dominates" $\mathcal{D}_{\mathcal{Z},\mathcal{A}}^f$ for any static adversary $\mathcal{A}$ that corrupts no more than a relative $1-\alpha$ share of the total stake, in the sense that nonempty slots are more likely to be tainted under $\mathcal{D}_\alpha^f$ than they are under $\mathcal{D}_{\mathcal{Z},\mathcal{A}}^f$.

To make this relationship precise, we introduce the partial order $\preceq$ on the set $\{\perp, 0, 1\}$ associated with the Hasse diagram

$$\begin{tikzcd} & 1 \arrow[dl, dash] \arrow[dr, dash] & \\ \perp & & 0 \end{tikzcd}$$

so that $x \preceq y$ if and only if $x = y$ or $y = 1$. We extend this partial order to $\{\perp, 0, 1\}^R$ by declaring $x_1 \dots x_R \preceq y_1 \dots y_R$ if and only if $x_i \preceq y_i$ for each i. Intuitively, the relationship $x \prec y$ asserts that y is "more adversarial than" x; concretely, any legal fork for x is also a legal fork for y. We record this in the lemma below.

Lemma 4.9. *Let x and y be characteristic strings in $\{0,1,\perp\}^R$ for which $x \preceq y$. Then*

1. *for every fork F, $F \vdash_\Delta x \Longrightarrow F \vdash_\Delta y$;*
2. *for every Δ, $\mathrm{div}_\Delta(x) \leq \mathrm{div}_\Delta(y)$.*

Proof. The proof follows directly from the definition of Δ-fork and div_Δ. $\square$

Finally, we define a notion of stochastic dominance for distributions on characteristic strings, and α-dominated adversaries.

Definition 4.10. *We say that a subset $E \subseteq \{\perp, 0, 1\}^R$ is* monotone *if $x \in E$ and $x \preceq y$ implies that $y \in E$. Let $\mathcal{D}$ and $\mathcal{D}'$ be two distributions on the set of characteristic strings $\{\perp, 0, 1\}^R$. Then we say that $\mathcal{D}'$* dominates *$\mathcal{D}$, written $\mathcal{D} \preceq \mathcal{D}'$, if*

$$\Pr_{\mathcal{D}}[E] \leq \Pr_{\mathcal{D}'}[E]$$

for every monotone set E. An adversary $\mathcal{A}$ is called α-dominated if the distribution $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$ that it induces on the set of characteristic strings satisfies $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}} \preceq \mathcal{D}^f_\alpha$.

In our application, the events of interest are

$$D_\Delta = \{x \mid \mathrm{div}_\Delta(x) \geq k\} , \tag{9}$$

which are monotone by Lemma 4.9. We note that any static adversary that corrupts no more than a $1-\alpha$ fraction of stake is α-dominated,[2] and it follows that

$$\Pr_{\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}}[\mathrm{div}_\Delta(w) \geq k] \leq \Pr_{\mathcal{D}^f_\alpha}[\mathrm{div}_\Delta(w) \geq k] ,$$

where the random variable w is distributed according to $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$ in the first probability, and $\mathcal{D}^f_\alpha$ in the second. This motivates a particular study of the "dominant" distribution $\mathcal{D}^f_\alpha$.

4.3.2 The Induced Distribution $\rho_\Delta(\mathcal{D}^f_\alpha)$

The dominant distribution $\mathcal{D}^f_\alpha$ on $\{0, 1, \perp\}^R$ in conjunction with the definition of ρ_Δ of (3) above implicitly defines a family of random variables $\rho_\Delta(w) = x_1 \ldots x_\ell \in \{0,1\}^*$, where $w \in \{0, 1, \perp\}^R$ is distributed according to $\mathcal{D}^f_\alpha$. As noted above, $\ell = R - \#_\perp(w)$ is precisely the number of active indices of w. We now note a few properties of this resulting distribution that will be useful to us later. In particular, we will see that the x_i random variables are roughly binomially distributed, but subject to an exotic stochastic "stopping time" condition in tandem with some distortion of the last Δ variables.

It simplifies our analysis to treat w as the first R symbols of an infinite string $w_1 w_2 \ldots$ of independent random variables with distribution given by (8) above. (We use the same name for this infinite sequence as it will cause no confusion.) The distribution of the infinite sequence w can be given an alternative description as $b_0 e_1 b_1 e_2 b_2 \ldots$, where the (independent) random variables $e_i \in \{0, 1\}$ and $b_i \in \{\perp\}^*$ have the probability laws

$$e_i = \begin{cases} 0 & \text{with probability } p_0/(p_0 + p_1), \\ 1 & \text{with probability } p_1/(p_0 + p_1), \end{cases}$$

[2]Note that strictly speaking, this dominance is only satisfied for an "idealized" adversary whose actions can be represented also in the setting of the idealized protocol π_{iSPoS}. A general adversary trying to attack the imperfections of the cryptographic building blocks used (such as the signature scheme and the VRF) can increase the desired probabilities by additional negligible additive terms. For the sake of simplicity of our exposition, we will neglect these terms in the present treatment.

and $b_i = \perp^t$ with probability $p_\perp^t(1-p_\perp)$. In this description, the random variables b_i generate the contiguous sequences of $\perp$ symbols that appear between appearances of 0 and 1. Now we observe that $z_1 z_2 \ldots = \rho_\Delta(b_0 e_1 b_1 \ldots)$—which we temporarily treat as operating on an infinite sequence—has an immediate description in terms of the x_i, b_i random variables:

$$z_i = \begin{cases} 1 & \text{if } e_i = 1 \text{ or } |b_i| < \Delta - 1, \\ 0 & \text{if } e_i = 0 \text{ and } |b_i| \geq \Delta - 1. \end{cases}$$

It follows that the variables $z_i \in \{0,1\}$ are independent and binomially distributed, with the property that

$$\Pr[z_i = 0] = \left(\frac{p_0}{p_0 + p_1}\right) p_\perp^{\Delta-1} \overset{(8)}{=} \frac{\phi(\alpha)}{f} \cdot (1-f)^\Delta \overset{(5)}{\geq} \alpha \cdot (1-f)^\Delta \,, \tag{10}$$

where $\overset{(i)}{\geq}$ follows from equation (i) and the equality $p_0 + p_1 = 1 - p_\perp$.

In our setting, the reduction function $\rho_\Delta(\cdot)$ is applied to a prefix of the string w of finite length R. In fact, the resulting "stopping criteria" on the random variables $z_1, z_2, \ldots$ can both introduce correlations and distort the coordinatewise distribution. However, we note that $\rho_\Delta(w_1 \ldots w_R)$ produces a prefix of the sequence $z_1, z_2, \ldots$ with the irritating possibility that the last Δ of the z_i in this prefix may be altered by the fact that there are not sufficient symbols in the string w to satisfy the criteria for $z_i = 0$. Thus we have

$$x_1 \ldots x_{\ell-\Delta} = \rho_\Delta(w_1 \ldots, w_R)^{\lceil \Delta} \quad \text{is a prefix of} \quad z_1 z_2 \ldots \,. \tag{11}$$

where $\cdot^{\lceil \Delta}$ denotes the truncation operator that removes the last Δ symbols, and the sequence $z_1 z_2 \ldots$ is determined by the infinite string $w_1 w_2 \ldots$. Recall that the z_i are binomially distributed with parameter $\approx 1 - \alpha(1-f)^\Delta$.

4.4 Divergence for the Dominant Distribution

Our goal is to apply the reduction ρ_Δ, Lemma 4.8, and Theorem 4.6 to establish an upper bound on the probability that a string drawn from the dominant distribution $\mathcal{D}_\alpha^f$ has large Δ-divergence. The difficulty is that the distribution resulting from applying ρ_Δ to a string drawn from $\mathcal{D}_\alpha^f$ is no longer a simple binomial distribution, so we cannot apply Theorem 4.6 directly. We resolve this obstacle in the proof of the following theorem.

Theorem 4.11. *Let $f \in (0,1]$, $\Delta \geq 1$, and α be such that $\alpha(1-f)^\Delta = (1+\epsilon)/2$ for some $\epsilon > 0$. Let w be a string drawn from $\{0,1,\perp\}^R$ according to $\mathcal{D}_\alpha^f$. Then we have*

$$\Pr[\operatorname{div}_\Delta(w) \geq k + \Delta] = 2^{-\Omega(k) + \log R} \,.$$

Remark. Intuitively, the theorem asserts that sampling the characteristic string in the Δ-semi-synchronous setting with protocol parameter f according to $\mathcal{D}_\alpha^f$ is, for the purpose of analyzing divergence, comparable to the *synchronous* setting in which the honest stake has been reduced from α to $\alpha(1-f)^\Delta$.

Proof of Theorem 4.11. Observe that $\operatorname{div}_0(\cdot)$ is monotone in the sense that if $\check{y}$ is a prefix of y then $\operatorname{div}_0(\check{y}) \leq \operatorname{div}_0(y)$; this follows because any fork $\check{F} \vdash_0 \check{y}$ can be "extended" to a fork $F \vdash y$ which includes all tines of $\check{F}$. Additionally, we note that $\operatorname{div}_0(\cdot)$ has a straightforward "Lipshitz property":

if $|y| \leq |\check{y}| + s$ then $\mathrm{div}_0(y) \leq \mathrm{div}_0(\check{y}) + s$; this follows because any fork $F \vdash_0 y$ can be restricted to a fork $\check{F} \vdash_0 \check{y}$ by retaining only vertices labeled by $\check{y}$—this can trim no more than s vertices from any tine.

In light of Lemma 4.8 we conclude that

$$\mathrm{div}_\Delta(w) \leq \mathrm{div}_0(\rho_\Delta(w)) \leq \mathrm{div}_0(\rho_\Delta(w)^{\lceil\Delta}) + \Delta \leq \mathrm{div}_0(z_1 \dots z_R) + \Delta\,,$$

where the last inequality follows because the random variable $\rho_\Delta(w_1 \dots w_R)$ can certainly have length no more than R. As the random variables z_i are binomial with $\Pr[z_i = 0] \geq \alpha(1-f)^\Delta$, the conclusion of Theorem 4.11 now follows directly from the assumption that $\alpha(1-f)^\Delta \geq (1+\epsilon)/2$ and Theorem 4.6. □

4.5 Common Prefix

Our results on Δ-divergence from the previous section allow us to easily establish the following statement.

Theorem 4.12. *Let $k, R, \Delta \in \mathbb{N}$ and $\varepsilon \in (0,1)$. Let $\mathcal{A}$ be an α-dominated adversary against the protocol π_{iSPoS} for some α satisfying $\alpha(1-f)^\Delta \geq (1+\epsilon)/2$. Then the probability that $\mathcal{A}$, when executed in a Δ-semisynchronous environment, makes π_{iSPoS} violate the common prefix property with parameter k throughout a period of R slots is no more than $\exp(\ln R + \Delta - \Omega(k))$. The constant hidden by the $\Omega(\cdot)$-notation depends on ϵ.*

Proof. Observe that an execution of protocol π_{iSPoS} violates the common prefix property with parameter k precisely when the Δ-fork F induced by this execution has $\mathrm{div}_\Delta(F) \geq k$. We have

$$\Pr[\mathrm{div}_\Delta(F) \geq k] \leq \Pr_{\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}}[\mathrm{div}_\Delta(w) \geq k] \leq \Pr_{\mathcal{D}^f_\alpha}[\mathrm{div}_\Delta(w) \geq k] \leq \exp(\ln R - \Omega(k - \Delta))\,,$$

where the first inequality follows from the definition of $\mathrm{div}_\Delta(\cdot)$; the second one holds since $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}} \preceq \mathcal{D}^f_\alpha$ and the set D_Δ defined in (9) is monotone; and the last one follows from Theorem 4.11. (For convenience, we have moved the Δ outside the asymptotic notation, which only makes the bound weaker as the hidden constant is less than 1.) □

4.6 Chain Growth

To obtain a bound on the probability of a violation of the chain growth property, we again consider the Δ-right-isolated uniquely honest slots introduced in Section 4.2.[3] Intuitively, we argue that the leader of such a slot has already received all blocks that were created in all previous such slots and therefore the block it creates will be having depth larger than all these blocks. It then follows that the length of the chain grows by at least the number of such slots.

Theorem 4.13. *Let $k, R, \Delta \in \mathbb{N}$ and $\varepsilon \in (0,1)$. Let $\mathcal{A}$ be an α-dominated adversary against the protocol π_{iSPoS} for some $\alpha > 0$. Then the probability that $\mathcal{A}$, when executed in a Δ-semisynchronous environment, makes π_{iSPoS} violate the chain growth property with parameters $s \geq 4\Delta$ and $\tau = c\alpha/4$ throughout a period of R slots, is no more than*

$$\exp\left(-\frac{c\alpha s}{20\Delta} + \ln R\Delta + O(1)\right)\,,$$

where c denotes the constant $c := c(f,\Delta) = f(1-f)^\Delta$.

[3] Technically, one could get a slightly better result by giving up on the uniqueness property, as it is not needed for the argument.

Proof. Recall that the definition of chain growth requires that if the longest chain possessed by an honest party at the onset of some slot sl_1 is $\mathcal{C}_1$, and the longest chain possessed by a (potentially different) honest party at the onset of slot $sl_2 \geq sl_1 + s$ is $\mathcal{C}_2$, then $\text{length}(\mathcal{C}_2) - \text{length}(\mathcal{C}_1) \geq \tau s$.

Let $\hat{sl}_1, \dots, \hat{sl}_h$ be the increasing sequence of all Δ-right-isolated uniquely honest slots among the slots in $T := \{sl_1 + \Delta, sl_1 + \Delta + 1, \dots, sl_2 - \Delta\}$. Observe that since $\hat{sl}_1 \geq sl_1 + \Delta$, the leader of $\hat{sl}_1$ will append a block to a chain that is at least as long as $\mathcal{C}_1$, since $\mathcal{C}_1$ will be known to him and will be considered in the $\mathsf{maxvalid}$ function. Therefore, the chain that the leader of $\hat{sl}_1$ diffuses will be at least 1 block longer than $\mathcal{C}_1$. Analogously, the leader of every $\hat{sl}_i$ will diffuse a chain that is at least 1 block longer than the chain diffused by the leader of $\hat{sl}_{i-1}$ since $\hat{sl}_{i-1}$ is Δ-right-isolated. Finally, the chain diffused by the leader of $\hat{sl}_h$ will be known to all parties at slot sl_2 and hence $\text{length}(\mathcal{C}_2)$ will be at least as long as this chain. It follows that $\text{length}(\mathcal{C}_2) - \text{length}(\mathcal{C}_1) \geq h$.

It remains to bound the number h of Δ-right-isolated uniquely honest slots among the slots with indices in T. To make our notation more flexible, let $H_T(x)$ denote the number of Δ-right-isolated uniquely honest slots among the slots from T in $x \in \{0, 1, \perp\}^R$, we hence have $h = H_T(x)$ for $x \leftarrow \mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$. Furthermore, let $E \triangleq \left\{x \in \{0,1,\perp\}^R \mid H_T(x) < c\alpha s/4\right\}$ where $c = c(f, \Delta) = f(1-f)^\Delta$. Observe that E is monotone, and hence $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}} \preceq \mathcal{D}^f_\alpha$ implies

$$\Pr[h < c\alpha s/4] = \Pr_{x \leftarrow \mathcal{D}^f_{\mathcal{Z},\mathcal{A}}}[H_T(x) < c\alpha s/4] \leq \Pr_{x \leftarrow \mathcal{D}^f_\alpha}[H_T(x) < c\alpha s/4]$$

and it is sufficient to bound upper-bound the last probability.

Consider now a characteristic string x sampled according to $\mathcal{D}^f_\alpha$ and for each $t \in T$, let X_t be the indicator random variable for the event that $\hat{sl}_t$ is Δ-right-isolated uniquely honest. Observe that $\mu \triangleq \mathbb{E}[X_t] = p_0 p_\perp^{\Delta-1} \geq \alpha f(1-f)^\Delta$ by the inequalities (8) and (5), and that the random variables X_t and $X_{t'}$ are independent if $|t - t'| \geq \Delta$ (as they depend on the leader sets of non-overlapping sets of slots). If we let $T_z = \{t \in T \mid t \equiv z \mod \Delta\}$, then the family of variables X_t indexed by T_z are independent. Note also that $|T_z| > \lfloor (s - 2\Delta)/\Delta \rfloor \geq (s - 3\Delta)/\Delta$ and that we may write T as the disjoint union $T_0 \cup \dots \cup T_{\Delta-1}$. By the Chernoff bound of Appendix E with $\delta = 1/2$, for each T_z

$$\Pr\left[\sum_{t \in T_z} X_t < \mu|T_z|/2\right] \leq e^{-\mu|T_z|/20} \leq e^{-\frac{\mu(s-3\Delta)}{20\Delta}} .$$

Observe that if $\sum_{t \in T_z} X_t \geq \mu|T_z|/2$ for each z then also $H_T(x) = \sum_{t \in T} X_t \geq \mu|T|/2 \geq \mu\hat{s}/2$, where we let $\hat{s} \triangleq s - 2\Delta$. It follows from the union bound that

$$\Pr_{x \leftarrow \mathcal{D}^f_\alpha}[H_T(x) < \mu\hat{s}/2] \leq \Delta \cdot e^{-\frac{\mu(s-3\Delta)}{20\Delta}} .$$

As $\mu \geq \alpha f(1-f)^\Delta$, we obtain

$$\Pr_{x \leftarrow \mathcal{D}^f_\alpha}[H_T(x) < c\alpha\hat{s}/2] \leq \Pr_{x \leftarrow \mathcal{D}^f_\alpha}[H_T(x) < \mu\hat{s}/2] \leq \Delta \cdot e^{-\frac{c \cdot \alpha(s-3\Delta)}{20\Delta}} .$$

Since $s \geq 4\Delta$, we have $\hat{s} \geq s/2$ and we can conclude that

$$\Pr_{x \leftarrow \mathcal{D}^f_\alpha}[H_T(x) < c\alpha s/4] = \Delta \cdot e^{-\frac{c \cdot \alpha(s-3\Delta)}{20\Delta}} .$$

Applying the union bound over the R slots, we conclude that the probability that there is a chain growth violation with parameters s and $\tau = c\alpha/4$ is no more than

$$R\Delta \exp(-c\alpha(s - 3\Delta)/(20\Delta)) = \exp(-c\alpha(s - 3\Delta)/(20\Delta) + \ln R\Delta) . \qquad \square$$

4.7 Chain Quality

Lemma 4.14. *Let $k, \Delta \in \mathbb{N}$ and $\epsilon \in (0,1)$. Let $\mathcal{A}$ be an α-dominated adversary against the protocol π_{iSPoS} for some $\alpha > 0$ satisfying $\alpha(1-f)^{\Delta} = (1+\epsilon)/2$. Let $B_1, \ldots, B_k$ be a sequence of consecutive blocks in a chain C possessed by an honest party. Then at least one block B_i was created in a Δ-right-isolated uniquely honest slot, except with probability* $\exp(-\Omega(k))$.

Proof sketch. For convenience, let us call a slot *good* if it is Δ-right-isolated uniquely honest, and *bad* if it is neither empty nor good. Moreover, we call a block good (resp. bad) if it comes from a good (resp. bad) slot.

Towards contradiction, assume that all blocks $B_1, \ldots, B_k$ are bad. Let G_1 denote the latest good block preceding B_1 in C, and G_2 the earliest good block appearing after B_k in C (or the last block of C, if there is no good one). Note that all blocks between G_1 and G_2 are bad.

Let $\hat{sl}_1$ (resp. $\hat{sl}_2$) denote the good slot in which G_1 (resp. G_2) was created (if G_2 is not good, let $\hat{sl}_2$ be the current slot). Denote by T the continuous sequence of slots between $\hat{sl}_1$ and $\hat{sl}_2$, excluding $\hat{sl}_1$ and including $\hat{sl}_2$. As we argued in the proof of Theorem 4.13, in each good slot in T the (unique) leader creates a block that has depth increased by at least 1 compared to the block from the previous good slot. Therefore, we have $\mathbf{d}(G_2) \geq \mathbf{d}(G_1) + g$, where g is the number of good slots in T. However, in chain C we have $\mathbf{d}(G_2) \leq \mathbf{d}(G_1) + b$, where b is the number of bad slots in the same sequence T. These two conditions can only be satisfied at the same time if $g \leq b$, we will now show that this is very unlikely.

Consider $E = \left\{x \in \{0,1,\perp\}^R \mid g(x) \leq b(x)\right\}$, where $g(x)$ and $b(x)$, as intuition suggests, denote the numbers of good and bad slots on the positions indexed by T in the string x, respectively. We again observe that E is monotone and therefore $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}} \preceq \mathcal{D}^f_\alpha$ implies

$$\Pr_{x \leftarrow \mathcal{D}^f_{\mathcal{Z},\mathcal{A}}}[g(x) \leq b(x)] \leq \Pr_{x \leftarrow \mathcal{D}^f_\alpha}[g(x) \leq b(x)]$$

and it is sufficient to bound upper-bound the last probability. However, we know that $\alpha(1-f)^{\Delta} = (1+\epsilon)/2$ and as we observed in (10), this implies that good slots are sampled with higher probability than bad slots. Therefore, the probability that $g(x) \leq b(x)$ for $x \leftarrow \mathcal{D}^f_\alpha$ falls exponentially with k. □

Lemma 4.14 implies the following theorem.

Theorem 4.15. *Let $k, R, \Delta \in \mathbb{N}$ and $\epsilon \in (0,1)$. Let $\mathcal{A}$ be an adversary against the protocol π_{iSPoS}, inducing a distribution of the characteristic string $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$ such that $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}} \preceq \mathcal{D}^f_\alpha$ for some $\alpha > 0$ satisfying $\alpha(1-f)^{\Delta} \geq (1+\epsilon)/2$. Then the probability that $\mathcal{A}$, when executed in a Δ-semi-synchronous environment, makes π_{iSPoS} violate the chain quality property with parameters k and $\mu = 1/k$ throughout a period of R slots, is no more than* $\exp(\ln R - \Omega(k))$.

4.8 Adaptive Adversaries

The statements in the previous sections give us guarantees on the common prefix, chain growth, and chain quality properties as long as the adversary is α-dominated for some suitable value of α. In Section 4.3.1 we argued that any *static* adversary that corrupts at most $(1-\alpha)$-fraction of stake is α-dominated. In this section we extend this claim also to *adaptive* adversaries, showing that as long as they corrupt no more than $(1-\alpha)$-fraction of stake adaptively throughout the whole execution, they are still α-dominated.

Theorem 4.16. *Every adaptive adversary $\mathcal{A}$ that corrupts at most $(1-\alpha)$-fraction of stake throughout the whole execution is α-dominated.*

Proof sketch. Let us start by taking a different (but equivalent) view on the choice of slot leaders in the execution of π_{iSPoS}. Assuming that we have a fixed number C of coins (equally-sized units of stake), consider a family of independent, identically distributed boolean random variables $\{c_{t,i} \mid 1 \leq t \leq R, 1 \leq i \leq C\}$ such that for every $c_{t,i}$ we have

$$c_{t,i} = \begin{cases} 1 & \text{with probability } \phi_f(1/C) = 1 - (1-f)^{1/C}, \\ 0 & \text{otherwise.} \end{cases}$$

We can view each of the random variables $c_{t,i}$ as being associated with a concrete, fixed coin; with the intuitive interpretation that if $c_{t,i} = 1$ then the owner of coin i becomes a slot leader for slot t. Thanks to the "independent aggregation property" given in (1), sampling the random variables $c_{t,i}$ is a way of determining the slot leaders that is equivalent to the one used in π_{iSPoS}, i.e., switching to this method of assigning slot leaders does not affect $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$ for any adversary $\mathcal{A}$.

We now make the adversary stronger by allowing it to corrupt not only stakeholders, but individual coins. (Formally, we can see each stakeholder with stake s_i as s_i separate stakeholders where each controls a single coin; corrupting a coin then means corrupting such single-coin stakeholder. In particular, this means that after corrupting coin i in some slot t, the adversary also learns the values of the random variables $c_{t',i}$ for all $t' \geq t$.) To see that this only extends the class of considered adversaries, observe that any adversary $\mathcal{A}$ corrupting stakeholders can be trivially modified into a coin-corrupting adversary $\mathcal{A}_1$ that simply corrupts all the coins belonging to the stake of a player corrupted by $\mathcal{A}$, maintaining $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}} = \mathcal{D}^f_{\mathcal{Z},\mathcal{A}_1}$.

It is now important to observe that at any point during the execution, all the uncorrupted coins are identical from the perspective of the adversary due to symmetry. Therefore, for any coin-corrupting adversary $\mathcal{A}_1$ one can construct another coin-corrupting adversary $\mathcal{A}_2$ that achieves the same outcomes, but corrupts the coins according to some fixed ordering: whenever $\mathcal{A}_1$ corrupts a new coin, $\mathcal{A}_2$ instead corrupts the next coin in this ordering. The only difference this makes from the perspective of the adversary is that with any corruption of a coin in slot t, the index i of random variables $c_{t',i}$ for $t' \geq t$, that are disclosed to it, changes. However, all these variables are independent and identically distributed, hence we again have $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}_1} = \mathcal{D}^f_{\mathcal{Z},\mathcal{A}_2}$.

Finally, consider a static adversary $\mathcal{A}_3$ that corrupts the first $\lfloor (1-\alpha)C \rfloor$ coins with respect to the ordering used by $\mathcal{A}_2$. Then, during the execution, it acts exactly like $\mathcal{A}_2$ would, except for corruptions; this is possible, since any coins corrupted by $\mathcal{A}_2$ must be already corrupted by $\mathcal{A}_3$ from the beginning. Note that if we consider the natural coupling of the two executions with $\mathcal{A}_2$ and $\mathcal{A}_3$, where the same randomness is used, then the sets of coins chosen for slot leaders will be the same in both executions; and moreover, in each slot the set of coins corrupted by $\mathcal{A}_3$ is a superset of those corrupted by $\mathcal{A}_2$. This implies that $\Pr[w^{(2)} \preceq w^{(3)}] = 1$, where $w^{(i)}$ is the random variable corresponding to the characteristic string resulting from the execution with $\mathcal{A}_i$. Using Theorem E.2 from Appendix E, this in turn implies $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}_2} \preceq \mathcal{D}^f_{\mathcal{Z},\mathcal{A}_3}$. The proof is now concluded by observing that $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}_3} \preceq \mathcal{D}^f_{\alpha}$ follows from Section 4.3.1, since $\mathcal{A}_3$ is static and corrupts at most $(1-\alpha)$-share of the stake. $\square$

Theorem 4.16, together with Theorems 4.12, 4.13 and 4.15, gives us the following corollary.

Corollary 4.17. *Let $\mathcal{A}$ be an adaptive adversary against the protocol Π_{iSPoS} that corrupts at most $(1-\alpha)$-fraction of stake. Then the bounds on common prefix, chain growth and chain quality given in Theorems 4.12, 4.13, 4.15 are satisfied for $\mathcal{A}$.*

4.9 The Resettable Protocol

With the analysis of these basic structural events behind us, we remark that the same arguments apply to a modest generalization of the protocol which permits the adversary some control over the nonce. Specifically, we introduce a "resettable" initialization functionality $\mathcal{F}^r_{\mathsf{INIT}}$, which permits the adversary to select the random nonce from a family of r independent and uniformly random nonces. Specifically, $\mathcal{F}^r_{\mathsf{INIT}}$ is identical to $\mathcal{F}_{\mathsf{INIT}}$, with the following exception:

- Upon receiving the first request of the form $(\mathsf{genblock\_req}, U_i)$ from some stakeholder U_i, $\mathcal{F}^r_{\mathsf{INIT}}$ samples a nonce $\eta \overset{\$}{\leftarrow} \{0,1\}^\lambda$, defines a "nonce candidate" set $H = \{\eta\}$, and permits the adversary to carry out up to $r-1$ *reset events*: each reset event draws an independent element from $\{0,1\}^\lambda$, adds the element to the set H, and permits the adversary to replace the current nonce η with any element of H. Finally, $(\mathsf{genblock}, \mathbb{S}_0, \eta)$ is sent to U_i. Later requests from any stakeholder are answered using the same value η.

We remark that an equivalent formulation initially draws a nonce η, draws an independent family of $r-1$ "replacement nonces" $\eta_1, \ldots, \eta_{r-1}$, and permits the adversary to replace η with one from the set $\{\eta_i\}$ if he chooses.

It is immediate that this selection of η from among a set of size r uniformly random candidate nonces can inflate the probability of events during π_{iSPoS} by a factor no more than r. We record this as a corollary below.

Corollary 4.18 (Corollary to Theorems 4.12, 4.13, 4.15)**.** *The protocol Π_{iSPoS}, with initialization functionality $\mathcal{F}^r_{\mathsf{INIT}}$, satisfies the bounds of Theorems 4.12, 4.13, 4.15 with all probabilities scaled by r.*

Proof. The probability of any such event is no more than the probability that the event occurs under *any* of the nonces $\{\eta\} \cup \{\eta_1, \ldots, \eta_{r-1}\}$ which, by the union bound, is no more than r times that the probability the event would have occurred under $\mathcal{F}_{\mathsf{INIT}}$. □

5 The Dynamic Stake Case

In this section, we construct a protocol that handles the dynamic case, where the stake distribution changes as the protocol is executed. As in Ouroboros [KRDO17], we divide protocol execution in a number of independent *epochs* during which the stake distribution used for sampling slot leaders remains unchanged. The strategy we use to bootstrap the static protocol is, at a high level, similar: we first show how the protocol can accommodate dynamic stake utilizing an ideal "leaky beacon" functionality and then we show this beacon functionality can be simulated via an algorithm that collects randomness from the blockchain.

In order to facilitate the implementation of our beacon, we need to allow the leaky beacon functionality to be adversarially manipulated by allowing a number of "resets" to be performed by the adversary. Specifically, the functionality is parameterized by a parameter τ and a parameter r. First, it leaks to the adversary, up to τ slots ahead of an epoch, the beacon value of the next epoch. Second, the adversary can reset the value returned by the functionality a number of times up to r. As expected for a beacon, it reports to honest parties the beacon value once the epoch starts. After the epoch is started no more resets are allowed for the beacon value. This mimics the functionality $\mathcal{F}_{\mathsf{INIT}}$ and its resettable version $\mathcal{F}^r_{\mathsf{INIT}}$. Note that the ability of the adversary to reset the beacon can be quite influential in the protocol execution: for instance, any event that depends deterministically on the nonce of an epoch and happens with probability $1/2$ can be easily forced to happen almost always by the adversary using a small number of resets.

Expectedly, we do not want to assume the availability of a randomness beacon, even if it is leaky and resettable. In our final iteration of the protocol we show how it is possible to simulate such beacon using a hash function that is modeled as a random oracle applied on VRF values from the blockchain itself. The verifiability of those values is a key property that we exploit in the proof. Our proof strategy is to reduce any adversary against the basic properties of the blockchain to a resettable beacon adversary that will simulate the random oracle. The key point of this reduction is that whenever the random oracle adversary makes a query with a sequence of values that is a candidate sequence from the nonce of the next epoch, the resettable attacker detects this as a possible reset opportunity and resets the beacon; it obtains the response from the beacon and sets this as the answer to the random oracle query. The final issue is to bound the number of resets: given that the adversary controls a ratio of stake below $1/2$, he will have assigned about $1/2 - \varepsilon$ fraction of the last $4k$ slots of an epoch and thus can explore an exponentially large space of tines as possibilites for setting the next epoch nonce. It follows that this is equal to the number of random oracle queries that the adversary can afford in a sequence of $4k$ slots. To refine this bound we utilize the q-bounded model of [GKL15] that bounds the number of queries the adversary can pose per round: in that model, the adversary is allowed q queries per adversarial party per round ("slot" in our setting).[4] Without loss of generality we can assume a single adversarial party exists and hence we obtain a bound equal to $4qtk$.

5.1 The Dynamic Stake Case with a Resettable Leaky Randomness Beacon

First we construct a protocol for the dynamic stake case assuming access to a resettable leaky beacon that provides a fresh nonce for each epoch. This beacon is leaky in the sense that it allows the adversary to obtain the nonce for the next epoch before the epoch starts and resettable in the sense that it allows the adversary to reset the nonce a number of times. We model the resettable leaky randomness beacon in functionality $\mathcal{F}_{RLB}^{\tau,r}$ presented in Figure 6.

We now describe protocol π_{DPoS}, which is a modified version of π_{SPoS} that updates its genesis block B_0 (and thus the assignment of slot leader sets) for every new epoch. Protocol π_{DPoS} is described in Figure 7 and functions in the $\mathcal{F}_{RLB}^{\tau,r}$-hybrid model.

We proceed to the security analysis of this protocol in the hybrid world where the functionality $\mathcal{F}_{RLB}^{\tau,r}$ is available to the protocol participants. A key challenge is that in the dynamic stake setting, the honest majority assumption that we have in place for the stakeholders refers to the stakeholder view of the honest stakeholders in each slot. Already in the first few slots this assumption may diverge rapidly from the stakeholder distribution that is built-in the genesis block.

To accomodate the issues that will arise from the movement of stake throughout protocol execution, we recall the notion of stake shift defined in Ouroboros [KRDO17] before the theorem about the security of π_{DPoS}:

Definition 5.1. *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.*

Theorem 5.2. *Fix parameters $k, R, \Delta, L \in \mathbb{N}, \epsilon, \sigma \in (0,1)$. Let $R \geq 8k$ be the epoch length, L the total lifetime of the system, and $\alpha_{\mathcal{H}}(1-f)^{\Delta} \geq (1+\epsilon)/2+\sigma$. The protocol Π_{DPoS}, with initialization*

[4]Note that we utilize the q-bounded model only to provide a more refined analysis; given that the total length of the execution is polynomial in λ one may also use the total execution length as a bound.

Functionality $\mathcal{F}_{RLB}^{\tau,r}$

$\mathcal{F}_{RLB}^{\tau,r}$ incorporates the diffuse and key/transaction functionality from Section 2.2 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 nonce leakage parameter τ and a number of allowed resets r. $\mathcal{F}_{RLB}^{\tau,r}$ interacts with stakeholders $U_1, \ldots, U_n$ and an adversary $\mathcal{A}$ as follows:

- Upon receiving $(\mathsf{genblock\_req}, U_i)$ from stakeholder U_i it operates as functionality $\mathcal{F}_{\mathsf{INIT}}$ on that message.
- Upon receiving $(\mathsf{epochrnd\_req}, U_i, e_j)$ from stakeholder U_i, if $e_j \geq 2$ is the current epoch, $\mathcal{F}_{RLB}^{\tau,r}$ sends $(\mathsf{epochrnd}, \eta_j)$ to U_i.
- For every epoch e_j, at slot $jR-\tau$, $\mathcal{F}_{RLB}^{\tau,r}$ samples the next epoch's nonce $\eta_{j+1} \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and leaks it by sending $(\mathsf{epochrnd\_leak}, e_j, \eta_{j+1})$ to the adversary $\mathcal{A}$. Additionally, $\mathcal{F}_{RLB}^{\tau,r}$ sets an internal reset request counter $\mathsf{Resets} = 0$ and sets $\mathbb{P} = \emptyset$.
- Upon receiving $(\mathsf{epochrnd\_reset}, \mathcal{A})$ from $\mathcal{A}$ at epoch e_j, if $\mathsf{Resets} < r$ and if the current slot is past slot $jR-\tau$, $\mathcal{F}_{RLB}^{\tau,r}$ samples a fresh nonce for the next epoch $\eta_{j+1} \overset{\$}{\leftarrow} \{0,1\}^\lambda$ and leaks it by sending $(\mathsf{epochrnd\_leak}, \eta_{j+1})$ to $\mathcal{A}$. Finally, $\mathcal{F}_{RLB}^{\tau,r}$ increments Resets and adds η_{j+1} to $\mathbb{P}$.
- Upon receiving $(\mathsf{epochrnd\_set}, \mathcal{A}, \eta)$ from $\mathcal{A}$ at epoch e_j, if the current slot is past slot $jR-\tau$ and if $\eta \in \mathbb{P}$, $\mathcal{F}_{RLB}^{\tau,r}$ sets $\eta_{j+1} = \eta$ and sends $(\mathsf{epochrnd\_leak}, \eta_{j+1})$ to $\mathcal{A}$.

Figure 6: Functionality $\mathcal{F}_{RLB}^{\tau,r}$.

functionality $\mathcal{F}_{\mathsf{INIT}}^r$ and access to $\mathcal{F}_{RLB}^{\tau,r}$, with $\tau \leq 4k$ satisfies persistence with parameters k and liveness with parameters $u = 2k$ throughout a period of L slots of Δ-semisynchronous execution with probability $1 - \exp(\ln L + \Delta - \Omega(k - \log r))$ assuming that σ is the maximum stake shift over R slots.

Proof sketch. We first observe that due to the conditions imposed on the leakiness of the $\mathcal{F}_{RLB}^{\tau,r}$ oracle, its level of resettability and the stake shift advantage that reduces the adversarial probability, Corollary 4.18 still applies for the whole execution over L slots. Thus we observe that any violation of persistence in the execution with parameter k results in the violation of CP with parameter k and as a result it will be bounded by error $\exp(\ln L + \Delta - \Omega(k))$. We then examine liveness. Consider any transaction that is provided to the honest parties for a sequence of $u = 8k/(1+\epsilon)$ slots, it holds that except with probability $\exp(-\Omega(k) + \ln L\Delta)$ the chain will grow by $f(1-f)^\Delta \alpha_{\mathcal{H}}/4 \cdot u \geq (1+\epsilon)/8 \cdot u = k$ blocks. By the chain quality property this means that at least one honest block will be added and hence this block will contain the transaction posted. □

5.2 Instantiating $\mathcal{F}_{RLB}^{\tau,r}$

In this section, we show how to substitute the oracle $\mathcal{F}_{RLB}^{\tau,r}$ of protocol π_{DPoS} with a subprotocol π_{RLB} that simulates $\mathcal{F}_{RLB}^{\tau,r}$. The resulting protocol can then operate directly in the $\mathcal{F}_{\mathsf{INIT}}$-hybrid model as in Section 3 (without resets) while utilizing a random oracle $\mathsf{H}(\cdot)$. The sub-protocol π_{RLB} is described in Figure 8.

We will show next that the sub-protocol π_{RLB} can safely substitute $\mathcal{F}_{RLB}^{\tau,r}$ when called from protocol π_{DPoS}. We will perform our analysis in the q-bounded model of [GKL15] assuming that the adversary is capable of issuing q queries per each round of protocol execution.

Lemma 5.3. *Consider the event of violating one of common prefix, chain quality, chain growth in an execution of π_{DPoS} using sub-protocol π_{RLB} with adversary $\mathcal{A}$ and $\mathcal{Z}$. We construct an adversary*

Protocol π_{DPoS}

Let $\mathsf{H}(\cdot)$ be a random oracle, $\mathsf{F}.(\cdot) : \{0,1\}^{\ell} \rightarrow \{0,1\}^{\ell_{\mathsf{VRF}}}$ be a family of unpredictable under malicious key generation VRFs with algorithms $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ and $\mathsf{KES} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update})$ be a forward secure key evolving signature scheme. Define $T_j^i = 2^{\ell_{\mathsf{VRF}}}\phi_f(\alpha_i^j)$ as the threshold of a stakeholder U_i for epoch e_j, where α_i^j is the relative stake of stakeholder U_i at epoch e_j, ℓ_{VRF} is the length in bits of the VRF output and f is the active slots coefficient. π_{DPoS} is a protocol run by a set of stakeholders, initially equal to $U_1, \ldots, U_n$, interacting among themselves and with $\mathcal{F}_{RLB}^{\tau,r}$ over a sequence of L slots $S = \{sl_1, \ldots, sl_L\}$ and L/R epochs with R slots (w.l.o.g. we assume L is divisible by R). 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 sends $(\mathsf{genblock\_req}, U_i)$ to $\mathcal{F}_{RLB}^{\tau,r}$, receiving $(\mathsf{genblock}, \mathbb{S}_0, \eta)$ as answer. U_i sets the local blockchain $\mathcal{C} = B_0 = (\mathbb{S}_0, \eta)$ and the initial internal state $st = H(B_0)$.
2. **Chain Extension** For every slot $sl \in S$, every online stakeholder U_i performs the following steps:
   (a) If a new epoch e_j, with $j \geq 2$, has started, U_i defines $\mathbb{S}_j$ to be the stakeholder distribution drawn from the most recent block with time stamp less than $jR - \tau$ as reflected in $\mathcal{C}$ (where τ parameterizes $\mathcal{F}_{RLB}^{\tau,r}$) and sends $(\mathsf{epochrnd\_req}, U_i, e_j)$ to $\mathcal{F}_{RLB}^{\tau,r}$, receiving $(\mathsf{epochrnd}, \eta_j)$ 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 $B' = (st', d', sl', B_{\pi}', \rho', \sigma_{j'}) \in \mathcal{C}'$ it holds that the stakeholder U' is in the slot leader set $\mathbb{L}'$ of slot sl' (by parsing B_{π}' as (pk', y', π'), verifying that $\mathsf{Ver}_{\mathsf{VRF}.pk'}(\eta_j \,\|\, sl' \,\|\, \mathtt{TEST}, y', \pi') = 1$, and that $y' < T_j'$ where T_j' is the threshold of stakeholder U' for the epoch e_j to which sl' belongs), $\mathsf{Ver}_{\mathsf{VRF}.pk'}(\eta_j \,\|\, sl' \,\|\, \mathtt{NONCE}, \rho_y', \rho_{\pi}') = 1$ (parsing ρ' as (ρ_y', ρ_{π}')), $\mathsf{Verify}_{\mathsf{KES}.vk'}((st', d', sl', B_{\pi}', \rho'), \sigma_{j'}) = 1$ and that the signature $\sigma_{j'}$ is for the the time period that corresponds to 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) U_i checks whether it is in the slot leader set $\mathbb{L}$ of slot sl with respect to the current epoch e_j by checking that $y < T_j^i$, where $(y, \pi) \leftarrow \mathsf{Prove}_{\mathsf{VRF}.sk_i}(\eta_j \,\|\, sl \,\|\, \mathtt{TEST})$. If yes, it generates a new block $B = (st, d, sl, B_{\pi}, \rho, \sigma)$ where st is its current state, $d \in \{0,1\}*$ is the transaction data, $B_{\pi} = (\mathsf{pk}_i, y, \pi)$, $\rho = (\rho_y, \rho_{\pi}) \leftarrow \mathsf{Prove}_{\mathsf{VRF}.sk_i}(\eta_j \,\|\, sl \,\|\, \mathtt{NONCE})$ and $\sigma_j = \mathsf{Sign}_{\mathsf{KES}.sk_{i,j}}(st, d, sl, B_{\pi}, \rho)$ is a signature on $(st, d, sl, B_{\pi}, \rho)$ for slot sl. U_i computes $\mathcal{C}' = \mathcal{C}|B$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\text{head}(\mathcal{C}'))$.
   (d) **E**xecute $\mathsf{Update}(\mathsf{KES}.sk_{i,j})$ obtaining the signing key $\mathsf{KES}.sk_{i,j+1}$ for the next slot, erases the current signing key $\mathsf{KES}.sk_{i,j}$. Finally, if U_i has generated a block in the previous step, it broadcasts $\mathcal{C}'$.

Figure 7: Protocol π_{DPoS}

Protocol π_{RLB}

Let $\mathsf{H}(\cdot)$ be a random oracle. π_{RLB} is a sub-protocol of π_{DPoS} and proceeds as follows:

- Upon receiving $(\mathsf{epochrnd\_req}, U_i, e_j)$ from stakeholder U_i, if $e_j \geq 2$ is the current epoch, it performs the following: for every block $B' = (st', d', sl', B_\pi{}', \rho', \sigma_{j'}) \in \mathcal{C}$ (where $\mathcal{C}$ is the callee's U_i's internal chain) belonging to epoch e_{j-1} up to the slot with timestamp less than $jR - 2k$, concatenate the values ρ' into a value v. Return $\eta_j = \mathsf{H}(j||v)$.

Figure 8: Protocol π_{RLB}.

$\mathcal{A}'$ so that corresponding event happens with the same probability in an execution of π_{DPoS} in the $\mathcal{F}^{\tau,r}_{RLB}$-hybrid world with adversary $\mathcal{A}'$ and environment $\mathcal{Z}$ assuming that $\tau \leq 4k$ and $r = 4kq$.

Proof. (Sketch) The adversary $\mathcal{A}'$ simulates $\mathcal{A}$ by maintaining locally the table for the random oracle $\mathsf{H}(\cdot)$. The key point in the simulation of $\mathcal{A}$ is to detect when is appropriate for $\mathcal{A}'$ to issue a reset query to its $\mathcal{F}^{\tau,r}_{RLB}$ oracle. Specifically, a reset query will be triggered whenever $\mathcal{A}$ queries $\mathsf{H}(\cdot)$ with concatenated valid VRF values $j \parallel \rho_i \parallel \cdots \parallel \rho_{i'}$ that are drawn from a valid chain and specifically from the first block of epoch e_j to a block of that epoch with time stamp at least $R-4k$. We observe that it will happen that the nonce of an epoch will be determined by the VRF values of a sequence of blocks that has time stamp less than $R-4k$ (if this is not the case liveness could be violated). Finally, when the epoch e_j reaches an end, $\mathcal{A}'$ will issue $(\mathsf{epochrnd\_set}, \mathsf{w},)$ query to $\mathcal{F}^{\tau,r}_{RLB}$ to set the value of the beacon to the correct value w of the $\mathsf{H}(\cdot)$ table as it has been determined by the chain that is on the common prefix. Note that if the $j \parallel \rho_i \parallel \cdots \parallel \rho_{i'}$ sequence corresponding to that chain in the common prefix was never queried to $\mathsf{H}(\cdot)$, $\mathcal{A}'$ will do a final reset query in order to obtain a value for this sequence, store it in its $\mathsf{H}(\cdot)$ table and set it for the next epoch. $\square$

Based on the above lemma, it is easy now to revisit Theorem 5.2, and show that the same result holds for r in the q-bounded model assuming $r = 4kq$ and $\tau \leq 4k$ which permits to set our epoch length R to $8k$.

References

[BGM16] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pages 142–157. Springer, Heidelberg, February 2016.

[BLMR14] 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.

[BM99] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 431–448. Springer, Heidelberg, August 1999.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219. IEEE Computer Society, 2004.

[CL07] Melissa Chase and Anna Lysyanskaya. Simulatable VRFs with applications to multi-theorem NIZK. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 303–322. Springer, Heidelberg, August 2007.

[DLS88] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[DPS16] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. http://eprint.iacr.org/2016/919.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, January 2005.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.

[GKL17] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. Cryptology ePrint Archive, Report 2014/765, updated version, March 2017. http://eprint.iacr.org/2014/765.

[IR01] Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 332–354. Springer, Heidelberg, August 2001.

[KKO77] T. Kamae, U. Krengel, and G. L. O'Brien. Stochastic inequalities on partially ordered spaces. *Ann. Probab.*, 5(6):899–912, 12 1977.

[KP15] Aggelos Kiayias and Giorgos Panagiotakos. Speed-security tradeoffs in blockchain protocols. Cryptology ePrint Archive, Report 2015/1019, 2015. http://eprint.iacr.org/2015/1019.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Proceedings of the 37th Annual International Cryptology Conference (CRYPTO)*, Lecture Notes in Computer Science. Springer, 2017. To appear. Preliminary version: iacr Cryptology ePrint archive 2016/889. http://eprint.iacr.org/2016/889.

[Lin09] Andrew Y. Lindell. Adaptively secure two-party computation with erasures. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 117–132. Springer, Heidelberg, April 2009.

[Mic16] Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

[MMM02] Tal Malkin, Daniele Micciancio, and Sara K. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 400–417. Springer, Heidelberg, April / May 2002.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.

[Nak08] Satoshi Nakamoto. "the proof-of-work chain is a solution to the byzantine generals' problem". The Cryptography Mailing List, https://www.mail-archive.com/cryptography@metzdowd.com/msg09997.html, November 2008.

[PS16] Rafael Pass and Elaine Shi. The sleepy model of consensus. Cryptology ePrint Archive, Report 2016/918, 2016. http://eprint.iacr.org/2016/918.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 643–673, 2017.

[RMKQ17] 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.

[Str65] V. Strassen. The existence of probability measures with given marginals. *Ann. Math. Statist.*, 36(2):423–439, 04 1965.

A The Model Based on [KRDO17]

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

- The current slot is determined by a publicly-known and monotonically increasing function of current time.
- Each player has access to the current time. Any discrepancies between parties' local time are insignificant in comparison with the length of time represented by a slot.

Security Model. We adopt the model introduced by [GKL15] for analysing security of blockchain protocols enhanced with an ideal functionality $\mathcal{F}$. We denote by $\mathsf{VIEW}^{P,\mathcal{F}}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda)$ the view of party P after the execution of protocol Π with adversary $\mathcal{A}$, environment $\mathcal{Z}$, security parameter λ and access to ideal functionality $\mathcal{F}$. We note that multiple different "functionalities" can be encompassed by $\mathcal{F}$.

In our model we employ the "Delayed Diffuse" and "Key and Transaction" functionalities. The former is based on the Diffuse functionality from [KRDO17] that we describe below, modified as outlined in Section 2.2. The latter contains a modification to account for immediate adaptive corruption, and we describe it in full here.

Diffuse functionality from [KRDO17]. It maintains a incoming string for each party U_i that participates. A party, if activated, is allowed at any moment to fetch the contents of its incoming string hence one may think of this as a mailbox. Furthermore, parties can give the instruction to the functionality to diffuse a message. The functionality keeps rounds and all parties are allowed to diffuse once in a round. Rounds do not advance unless all parties have diffused a message. The adversary, when activated, can also interact with the functionality and is allowed to read all inboxes and all diffuse requests and deliver messages to the inboxes in any order it prefers. At the end of the round, the functionality will ensure that all inboxes contain all messages that have been diffused (but not necessarily in the same order they have been requested to be diffused).

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. (v) A transaction may be requested on behalf of a certain user by the environment, by providing a template for the transaction (which should contain a unique nonce) and a recipient. The functionality will adjust the stake of each stakeholder accordingly. (iv) An existing user may be requested to be corrupted by the adversary via a message $(\mathsf{Corrupt}, U)$. A user can only be corrupted immediately, i.e., without any delay.

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).
- If a stakeholder does not fetch in a certain slot the messages written to its incoming string in the diffuse functionality they are flushed.

Restrictions imposed on the environment. 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. These environment restrictions are described in Section 2.2.

B Definitions

In this appendix, we present formal definitions of Verifiable Random Functions and Key Evolving Signature Schemes with Forward Security.

B.1 Verifiable Random Functions

We present formal definitions of Verifiable Random Functions from [DY05].

Definition B.1 (Verifiable Random Function)**.** *A function family* $\mathsf{F}.(\cdot) : \{0,1\}^{\ell} \to \{0,1\}^{\ell_{\mathsf{VRF}}}$ *is a family of VRFs if there exist algorithms* $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ *such that (i.)* $\mathsf{Gen}(1^k)$ *outputs a pair of keys* $(\mathsf{VRF}.pk, \mathsf{VRF}.sk)$*, (ii.)* $\mathsf{Prove}_{\mathsf{VRF}.sk}(x)$ *outputs a pair* $(\mathsf{F}_{\mathsf{VRF}.sk}(x), \pi_{\mathsf{VRF}.sk}(x))$*, where* $\mathsf{F}_{\mathsf{VRF}.sk}(x) \in \{0,1\}^{\ell_{\mathsf{VRF}}}$ *is the function value and* $\pi_{\mathsf{VRF}.sk}(x)$ *is the proof of correctness, and (iii.)* $\mathsf{Ver}_{\mathsf{VRF}.pk}(x, y, \pi_{\mathsf{VRF}.sk}(x))$ *verifies that* $y = \mathsf{F}_{\mathsf{VRF}.sk}(x)$ *using proof* $\pi_{\mathsf{VRF}.sk}(x)$*, outputting* 1 *if* y *is valid and* 0 *otherwise. Additionally, we require the following properties:*

1. ***Uniqueness:*** *no values* $(\mathsf{VRF}.pk, x, y, y', \pi_{\mathsf{VRF}.sk}(x), \pi_{\mathsf{VRF}.sk}(x)')$ *can satisfy both*

$$\mathsf{Prove}_{\mathsf{VRF}.pk}(x, y, \pi_{\mathsf{VRF}.sk}(x)) = 1 \qquad \textit{and} \qquad \mathsf{Prove}_{\mathsf{VRF}.pk}(x, y', \pi_{\mathsf{VRF}.sk}(x)') = 1$$

unless $y = y'$*.*

2. ***Provability:*** *if* $y, \pi_{\mathsf{VRF}.sk}(x) = \mathsf{Prove}_{\mathsf{VRF}.sk}(x)$*, then* $\mathsf{Ver}_{\mathsf{VRF}.pk}(x, y, \pi_{\mathsf{VRF}.sk}(x)) = 1$*.*

3. ***Pseudorandomness:*** *for any PPT algorithm* $A = (A_E, A_J)$*, which runs for a total of* $s(k)$ *steps when its first input is* 1^k*, and does not query the* $\mathsf{Prove}(\cdot)$ *oracle on* x*,*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{l} (\mathsf{VRF}.pk, \mathsf{VRF}.sk) \leftarrow \mathsf{Gen}(1^k); \\ (x, A_{st}) \leftarrow A_E^{\mathsf{Prove}(\cdot)}(\mathsf{VRF}.pk); \\ y_0 = \mathsf{F}_{\mathsf{VRF}.sk}(x); y_1 \leftarrow \{0,1\}^{\ell_{\mathsf{VRF}}}; \\ b \leftarrow \{0,1\}; b' \leftarrow A_J^{\mathsf{Prove}(\cdot)}(y_b, A_{st}) \end{array} \right] \leq \frac{1}{2} + negl(k)\,.$$

B.2 Forward Secure Signatures Schemes

We present the formal definitions of key evolving signature schemes and forward security of [BM99, IR01].

Definition B.2 (Key Evolving Signature Schemes)**.** *A key evolving signature scheme* $\mathsf{KES} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update})$ *is a tuple of algorithms such that:*

1. $\mathsf{Gen}(1^k, T)$ *is a probabilistic key generation algorithm that takes as input a security parameter* 1^k *and the total number of periods* T*, outputting a key pair* $(\mathsf{KES}.sk_1, \mathsf{KES}.vk)$*, where* $\mathsf{KES}.vk$ *is the verification key and* $\mathsf{KES}.sk_1$ *is the initial signing key (we assume that the period* j *to which a signing key* $\mathsf{KES}.sk_j$ *corresponds is encoded in the signing key itself).*

2. $\mathsf{Sign}_{\mathsf{KES}.sk_j}(m)$ *is a probabilistic signing algorithm that takes as input a secret key* $\mathsf{KES}.sk_kest$ *for the time period* $j \leq T$ *and a message* m*, outputting a signature* σ_j *on* m *for time period* j *(we assume that the period* j *for which a signature* σ_j *was generated is encoded in the signature itself).*

3. $\mathsf{Verify}_{\mathsf{KES}.vk}(m, \sigma_j)$ *is a deterministic verification algorithm that takes as input a public key* $\mathsf{KES}.vk$*, a message* m *and a signature* σ_j*, outputting* 1 *if* σ_j *is a valid signature on message* m *for time period* j *and* 0 *otherwise.*

4. $\mathsf{Update}(\mathsf{KES}.sk_j)$ *is a probabilistic* secret key update *algorithm that takes as input a secret key* $\mathsf{KES}.sk_j$ *for the current time period* j *and outputs a new secret key* $\mathsf{KES}.sk_{j+1}$ *for time period* $j+1$*. We define* $\mathsf{KES}.sk_{T+1}$ *as the empty string and set it as the output of* $\mathsf{Update}(\mathsf{KES}.sk_T)$*.*

Correctness: *for every key pair* $(\mathsf{KES}.sk_1, \mathsf{KES}.vk) \leftarrow \mathsf{Gen}(1^k, T)$*, every message* m *and every time period* $j \leq T$*,*
$\mathsf{Verify}_{\mathsf{KES}.vk}(m, \mathsf{Sign}_{\mathsf{KES}.sk_j}(m)) = 1$*.*

Given a key evolving signature scheme, forward security is defined by a game that starts as the standard Chosen Message Attack (CMA) experiment but after a number of queries to the signing oracle allows the adversary to learn the signing key for the current time period. The adversary is successful if it can produce a valid signature on a message of its choice for an earlier time period. The experiment and forward security are formally defined as follows.

Definition B.3 (Forward Security Experiment)**.** *A forger is a pair of algorithms* $F = (F_{\mathsf{cma}}, F_{\mathsf{forge}})$ *such that* F_{cma} *has access to a signing oracle. For a key pair* $(\mathsf{KES}.vk, \mathsf{KES}.sk_1) \leftarrow \mathsf{Gen}(1^k, T)$*,* F_{cma} *is given* $\mathsf{KES}.vk$ *and* T *and queries the signing oracle* q_{sig} *times with adaptively chosen message and time period pairs, outputting the set of queried message and time period pairs* CM*, the set of corresponding signatures* $sign(CM)$ *and a time period* b*. Given* CM*,* $sign(CM)$ *and the signing key* $\mathsf{KES}.sk_b$ *for time period* b*,* F_{forge} *outputs* $(m, \sigma_j) \leftarrow F_{\mathsf{forge}}(CM, sign(CM), \mathsf{KES}.sk_b)$*.* F *is successful if* $(m, j) \notin CM$*,* $j < b$ *and* $\mathsf{Verify}_{\mathsf{KES}.vk}(m, \sigma_j) = 1$*. (The two components of* F *can communicate the necessary information, including* T *and* b *through* CM*.)*

Definition B.4 (Forward Security)**.** *Let* $\mathbf{Succ}^{fwsig}(\mathsf{KES}[k, T], F)$ *be the probability (over the random coins of* KES *and* F*) that* F *is successful in the forward security experiment of Definition B.3. Let the function* $\mathbf{InSec}^{fwsig}(\mathsf{KES}[k, T], t, q_sig)$ *(the* insecurity *function) be the maximum of* $\mathbf{Succ}^{fwsig}(\mathsf{KES}[k, T], F)$*, over all algorithms* F *that are restricted to running time* t *and* q_sig *signature queries. A key evolving signature scheme* KES *is forward secure against an adversary that runs in time* t *and makes* q_sig *signature queries if* $\mathbf{Succ}^{fwsig}(\mathsf{KES}[k, T], F)$ *is negligible in* k*.*

C Insecurity of the original Ouroboros against adversarial message delays

This appendix informally describes several attacks against the Proof-of-Stake protocol Ouroboros proposed in [KRDO17], when used in various environments that allow the adversary to control message delays to some extent.

We consider two variants of the semi-synchronous model. With *sender-side delays*, each message can be delayed on the side of its sender, and hence after being delayed, it arrives to all recipients in the same round. On the other hand, if we allow for *recipient-side delays*, the each message can be delayed for a different time period for each of its recipients. The latter model is the one that we consider for our positive results in the main body of the paper. Clearly this latter model gives more power to the adversary, hence we explore it first here.

C.1 Recipient-Side Delays

The Model. The model for recipient-side delays is identical to the one given in Section 2.2.

Attack Description. Intuitively, the adversary aims to violate the common prefix property by maintaining two tines that are growing at approximately the same rate: so that their lengths never differ by more than one block. This is achieved by disclosing the blocks mined in the past Δ rounds (which are distributed via the $\mathsf{DDiffuse}$ functionality and hence can be delayed by $\mathcal{A}$) in a controlled way to affect the decision of the current slot leader (in case he is honest) about which of the two tines to extend.

The attack can be performed even in the simple setting with a static stake and slot leaders sampled by an idealized beacon. Moreover, it can be carried out without any corrupted parties at all (i.e., also if the adversarial stake ratio $\alpha_{\mathcal{A}} = 0$), as long as $\mathcal{A}$ maintains control over message delays.

In detail, $\mathcal{A}$ behaves as follows:

1. Internally, $\mathcal{A}$ maintains two tines T_0 and T_1, initially empty. Whenever any party diffuses a chain C such that some T_i is a prefix of C, $\mathcal{A}$ replaces T_i with C (except for the trivial initial case when any chain is a prefix of both T_0 and T_1, here $\mathcal{A}$ only replaces T_0).
2. In each slot sl_r:
   (a) Determine T_s, the tine that is currently not longer, i.e., such that it satisfies $|T_s| \leq |T_{1-s}|$.
   (b) Let U_i denote the slot leader for the upcoming slot sl_{r+1}. If a message containing T_s was diffused in this round, $\mathcal{A}$ delivers it to the inbox of U_i and to the $\mathsf{delayed}_j$-strings for all other parties $j \neq i$. Otherwise, if a message containing T_s is already present in $\mathsf{delayed}_i$, $\mathcal{A}$ removes it from $\mathsf{delayed}_i$ and delivers it to the inbox of party U_i.
   (c) $\mathcal{A}$ moves all messages diffused in this round into the $\mathsf{delayed}$-strings of all parties.

C.2 Sender-Side Delays

We now argue that the original Ouroboros protocol is insecure even against sender-side adversarial message delays.

The Model. We consider an ideal functionality $\mathsf{SDiffuse}_\Delta$ that is defined exactly as the functionality $\mathsf{Diffuse}$ given in [KRDO17], except for two differences:

1. When the adversary $\mathcal{A}$ is activated, besides performing any of the actions that were allowed by the $\mathsf{Diffuse}$ functionality, it is also allowed to:
   - move any message obtained in a diffuse request from a party to a special string $\mathsf{delayed}$,
   - move any message from the string $\mathsf{delayed}$ to the inboxes of all parties.
2. At the end of each round, the functionality ensures that for every message that was either (a) diffused in this round and not put to the string $\mathsf{delayed}$ or (b) removed from the string $\mathsf{delayed}$ in this round, it is present in the inboxes of all parties. If any message currently present in $\mathsf{delayed}$ was originally diffused at least Δ slots ago, then the functionality removes it from $\mathsf{delayed}$ and appends it to the inbox of all parties.

We again define our model by replacing $\mathsf{Diffuse}$ by $\mathsf{SDiffuse}_\Delta$ in the model of [KRDO17] (this gives us a class of models parametrized by Δ, setting $\Delta = 0$ again results in the original model of [KRDO17]).

Attack Description. The adversary again aims to violate the common prefix property by maintaining two tines that are growing at approximately the same rate. However, this time it cannot deliver messages selectively to future slot leaders, and hence the attack requires a slight modification.

The details of the attack depend on the exact definition of the maxvalid function that honest parties use to choose the winning chain, namely on how it does tie-breaking in case of several equal-length chains. According to [KRDO17], maxvalid should favor the current chain C if it is the longest, otherwise choose arbitrarily. There are several natural possibilities to perform this choice:

(i) Choose a chain that was delivered first out of those with maximal length.

(ii) Choose a chain at random out of those with maximal length.

(iii) Prefer an extension of the current chain C. This is not fully specified, a rule to choose among several extensions of C with maximal length is also needed.

(iv) Apply some fixed ordering rule, e.g. take the lexicographically first out of the chains with maximal length.

We now sketch an attack for each of the cases above. The attacks can again be performed even in the simple setting with a static stake, slot leaders sampled by an idealized beacon, and without any corrupted parties.

Case (i). The adversary starts by partitioning the stakeholders into two sets S_0 and S_1 so that each of these sets controls about one half of the total stake. It again maintains two tines T_0 and T_1, and also keeps track of the prefixes T'_i of each T_i that were already delivered by $\mathsf{SDiffuse}_\Delta$ to all parties. The goal of $\mathcal{A}$ is to maintain $|T'_0| = |T'_1|$, and make all parties in S_i believe that T'_i is their current chain. This is achieved as follows:

- In each slot sl_j, the slot leader $U_j \in S_i$ will extend T'_i.
- $\mathcal{A}$ will delay this new block unless there is already also an existing block in $T_{1-i} \setminus T'_{1-i}$ that can be used to extend both T'_0 and T'_1 by one block at the same time.
- If this is the case, $\mathcal{A}$ delivers both delayed blocks, extends both T'_0 and T'_1 by one block, and uses its power to reorder messages in the inboxes of honest parties to maintain that parties in S_i still consider the new T'_i to be their current chain (note that parties follow the rule ((i)) above).

The probability that a message would need to be delayed by $\mathcal{A}$ for more than Δ slots to follow this strategy decreases exponentially with Δ.

Case (ii). A similar approach as in the case (i) will work, with one small change. Here $\mathcal{A}$ does not need to choose partitions S_0 and S_1 and maintain them using its inbox-reordering capability. Instead, it can simply observe which of the chains T'_0 and T'_1 are being extended, and again only deliver extensions for both of them at the same time. By rule (ii), each stakeholder will choose its current chain by choosing at random between the new T'_0 and T'_1. This will guarantee a quite even distribution of parties into S_0 and S_1 unless there are parties with a very large stake.

Case (iii). The same attack as in the case (i) will work. Here the partitions S_0 and S_1 don't need to be maintained by inbox-reordering, each party will stay in the same partition thanks to following the rule (iii).

Case (iv). The attacker $\mathcal{A}$ again maintains two tines T_0 and T_1, and also keeps track of the prefixes T_i' of each T_i that were already delivered by $\mathsf{SDiffuse}_\Delta$ to all parties. The goal of $\mathcal{A}$ is to make T_0 and T_1 grow at roughly the same speed.

The attack starts by letting the honest slot leaders mine two separate length-1 tines from the genesis block (by delaying the first one). Denote these blocks $B_0^{(1)}$ and $B_1^{(1)}$, these will be the first blocks of T_0 and T_1, respectively. Now, $\mathcal{A}$ delivers to all parties the one of these two blocks (say $B_i^{(1)}$) that has lower preference in the fixed ordering given by the rule (iv), and hence the next honest slot leader will extend this tine by mining some block $B_i^{(2)}$ on top of $B_i^{(1)}$. $\mathcal{A}$ witholds $B_i^{(2)}$ but now publishes $B_{1-i}^{(1)}$ and due to the rule (iv), the next honest slot leader will mine a block on top of $B_{1-i}^{(1)}$, call it $B_{1-i}^{(2)}$. Now $\mathcal{A}$ is in the same situation as before, hence it again delivers the one of the blocks $B_0^{(2)}$ and $B_1^{(2)}$ that has lower preference according to the rule (iv). The attack continues analogously.

This attack only requires $\Delta \geq 2$.

D Proof of Theorem 3.10

Theorem 3.10. *Let* $\mathsf{H}(\cdot)$ *be a random oracle,* $\mathsf{F}.(\cdot) : \{0,1\}^\ell \to \{0,1\}^{\ell_{\mathsf{VRF}}}$ *be a family of VRFs with algorithms* $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Ver})$ *with standard security as stated in Definition B.1 and* D *be an input distribution with* k *bits of min-entropy. The family of VRFs* $\mathsf{UF}.(\cdot) : \{0,1\}^\ell \to \{0,1\}^{\ell_{\mathsf{VRF}}}$ *with algorithms* $(\mathsf{UGen}, \mathsf{UProve}, \mathsf{UVer})$ *constructed above is secure with respect to Definition B.1 and achieves unpredictability under malicious key generation as stated in Definition 3.9.*

Proof. First we show that The family of VRFs $\mathsf{UF}.(\cdot) : \{0,1\}^\ell \to \{0,1\}^{\ell_{\mathsf{VRF}}}$ with algorithms $(\mathsf{UGen}, \mathsf{UProve}, \mathsf{UVer})$ constructed above achieves standard VRF security as stated in Definition B.1. Provability holds by definition. If there exists an adversary $\mathcal{A}$ that breaks the uniqueness of $\mathsf{UF}.(\cdot)$ outputting $(\mathsf{UVRF}.pk, x, y, y', \mathsf{U}.\pi_{\mathsf{UVRF}.sk}(x),$
$\mathsf{U}.\pi_{\mathsf{VRF}.sk}(x)')$ such that $\mathsf{UProve}_{\mathsf{UVRF}.pk}(x, y, \mathsf{U}.\pi_{\mathsf{UVRF}.sk}(x)) = \mathsf{UProve}_{\mathsf{UVRF}.pk}(x, y', \mathsf{U}.\pi_{\mathsf{VRF}.sk}(x)')$, this adversary generates $(y_{\mathsf{U}} = (\mathsf{H}(x \,\|\, y), y), \mathsf{U}.\pi = \pi)$ and $(y_{\mathsf{U}} = (\mathsf{H}(x \,\|\, y'), y'), \mathsf{U}.\pi' = \pi')$ where it is ensured that x is the same (otherwise the verification would fail) but $y \neq y'$ in both y_{U} and y_{U}', while $\mathsf{Prove}_{\mathsf{VRF}.pk}(x, y, \pi) = \mathsf{Prove}_{\mathsf{VRF}.pk}(x, y', \pi')$. Thus, we can construct an adversary $\mathcal{A}^*$ that breaks uniqueness for the underlying family of VRFs $\mathsf{F}.(\cdot)$ by outputting the same as $\mathcal{A}$.

If there exists an adversary $\mathcal{A}$ that breaks the pseudorandomness of $\mathsf{UF}.(\cdot)$, we can construct an adversary $\mathcal{A}^*$ that breaks pseudorandomness of $\mathsf{F}.(\cdot)$ as follows: $\mathcal{A}^*$ gives public key $\mathsf{UVRF}.pk$ to $\mathcal{A}$ and every time $\mathcal{A}$ queries the $\mathsf{UProve}(\cdot)$ oracle on x, $\mathcal{A}^*$ queries the $\mathsf{Prove}(\cdot)$ oracle on x and returns $((\mathsf{H}(x \,\|\, \mathsf{F}_{\mathsf{UVRF}.sk}(x)),$
$\mathsf{F}_{\mathsf{UVRF}.sk}(x)), \pi_{\mathsf{UVRF}.sk}(x))$ to $\mathcal{A}$. When $\mathcal{A}$ outputs (x, A_{st}), $\mathcal{A}^*$ outputs the same, receives its challenge y_b and gives a challenge $y_{\mathsf{U}} = (\mathsf{H}(x \,\|\, y_b), y_b), y_b)$ to $\mathcal{A}$. $\mathcal{A}^*$ outputs whatever $\mathcal{A}$ outputs. Notice that the oracle queries are answered exactly as in the original pseudorandomness experiment and that the challenge provided by $\mathcal{A}^*$ is indistinguishable from the actual challenge in the experiment since y_b is pseudorandom and the output $\mathsf{H}\cdot$ is random. The reduction only fails if $x \,\|\, y_b$ has been queried to random oracle before, which happens with negligible probability.

As for unpredictability under malicious key generation, notice that challenge $y_0 = \mathsf{F}_{\mathsf{VRF}.sk}(x) = (\mathsf{H}(x \,\|\, \mathsf{F}_{\mathsf{UVRF}.sk}(x)), \mathsf{F}_{\mathsf{UVRF}.sk}(x))$ has at least k bits of min-entropy for $x \leftarrow \mathsf{D}$. Hence, if there exists an adversary $\mathcal{A}$ that distinguishes y_0 from y_1 (a random string of same size) with proability higher than $\frac{1}{2} + 2^{-k}$, we can construct an adversary $\mathcal{A}^*$ that distinguishes the output of the random oracle from a random string of same size with probability better than $\frac{1}{2} + 2^{-k}$ as well. $\square$

E Useful Probability-Theoretic Tools

In our arguments, we are using the following standard variant of the Chernoff bound. See, e.g., [MR95] for a proof.

Theorem E.1 (Chernoff 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 $\delta \geq 0$,*

- $\mathsf{P}[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu}$;
- $\mathsf{P}[X \leq (1-\delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu}$.

We also employ the following theorem.

Theorem E.2 ([Str65])**.** *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two distributions on a finite partially ordered set V with partial order $\preceq$. Then $\mathcal{D}_1 \preceq \mathcal{D}_2$ iff there is a pair of (typically dependent) random variables, X_1 and X_2, taking values in V so that each X_i is distributed according to $\mathcal{D}_i$, and $\Pr[X_1 \preceq X_2] = 1$.*

(Note that the statement of this theorem overloads the notation $\preceq$, applying it both to distributions in the sense of Definition 4.7 and elements of the partial order.) This result is implicit in Strassen's 1965 article [Str65]; a presentation with terminology closer to ours appears in Kamae et al. [KKO77].

F Further Details on Forks, Forkability and Divergence

We introduce the notion of a forkable string that was central to the analysis in [KRDO17].

Definition F.1 (Height and fork intersection)**.** *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.*

The common prefix property in the synchronous case is studied by focusing on a local property of "forkability".

Definition F.2 (Flat forks and forkable strings)**.** *We say that a synchronous 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 synchronous fork $F \vdash_0 w$.*

A fundamental tool in the security analysis in the synchronous case is an estimate of the number of forkable strings of a particular length k. The original bound of $2^{-\Omega(\sqrt{k})}$ in [KRDO17] was strengthened to $2^{-\Omega(k)}$ in [RMKQ17].

Theorem F.3 ([KRDO17, RMKQ17])**.** *Let $\epsilon \in (0,1)$ and let w be a string drawn from $\{0,1\}^k$ by independently assigning each $w_i = 1$ with probability $(1-\epsilon)/2$. Then $\Pr[w$ is forkable$] = 2^{-\Omega(k)}$. The constant hidden by the $\Omega(\cdot)$ notation depends only on ϵ.*

As mentioned above, the notion of forkability is directly related to (synchronous) divergence; this is reflected by the theorem below.

Theorem F.4 ([KRDO17])**.** *Let $w \in \{0,1\}^*$ with $\mathrm{div}_0(w) \geq k$. Then there is a forkable substring $\check{w}$ of w with $|\check{w}| \geq k$.*

An immediate conclusion of Theorems F.3 and Theorem F.4 is the following bound on the probability that a synchronous characteristic string drawn from the binomial distribution has large divergence.

Theorem F.5 (Restatement of Theorem 4.6)**.** *Let $\ell, k \in \mathbb{N}$ and $\epsilon \in (0,1)$. Let $w \in \{0,1\}^\ell$ be drawn according to the binomial distribution, so that* $\Pr[w_i = 1] = (1-\epsilon)/2$*. Then*

$$\Pr[\mathrm{div}_0(w) \geq k] \leq \exp(\ln \ell - \Omega(k)) \; .$$

A proof of Theorem 4.6 is developed for a weaker bound on forkability in [KRDO17]; here we include a proof adapted to the bound of Theorem F.3 for completeness.

Proof of Theorem 4.6. It follows from Theorem F.4 that if $\mathrm{div}_0(w) \geq k$, there is a forkable substring $\check{w}$ of length at least k. Thus

$$\begin{aligned}\Pr[\mathrm{div}_0(w) \geq k] &\leq \Pr\left[\begin{array}{l}\exists \alpha, \beta \in \{1,\ldots,\ell\} \text{ so that } \alpha + \\ k-1 \leq \beta \text{ and } w_\alpha \ldots w_\beta \text{ is fork-} \\ \text{able}\end{array}\right] \\ &\leq \sum_{1 \leq \alpha \leq \ell} \underbrace{\sum_{\alpha+k-1 \leq \beta \leq \ell} \Pr[w_\alpha \ldots w_\beta \text{ is forkable}] \; .}_{(*)}\end{aligned}$$

According to Theorem F.3 the probability that a string of length t drawn from this distribution is forkable is no more than $\exp(-ct)$ for a positive constant c. Note that for any $\alpha \geq 1$,

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

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

$$\Pr[\mathrm{div}_0(w) \geq k] \leq \ell \cdot \exp(-\Omega(k)) \leq \exp(\ln \ell - \Omega(k)) \, ,$$

as desired. □