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

Bernardo David[⋆], Peter Gaži[⋆⋆], Aggelos Kiayias[⋆⋆⋆], and Alexander Russell[†]

November 14, 2017

Abstract. We present "Ouroboros Praos", a proof-of-stake blockchain protocol that, for the first time, provides security against *fully-adaptive corruption* in the *semi-synchronous setting*: Specifically, the adversary can corrupt any participant of a dynamically evolving population of stakeholders at any moment as long the stakeholder distribution maintains an honest majority of stake; furthermore, the protocol tolerates an adversarially-controlled message delivery delay unknown to protocol participants.
To achieve these guarantees we formalize and realize in the universal composition setting a suitable form of forward secure digital signatures and a new type of verifiable random function that maintains unpredictability under malicious key generation. Our security proof develops a general combinatorial framework for the analysis of semi-synchronous blockchains that may be of independent interest. We prove our protocol secure under standard cryptographic assumptions in the random oracle model.

1 Introduction

The design of *proof-of-stake* blockchain protocols was identified early on as an important objective in blockchain design; a proof-of-stake blockchain substitutes the costly 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 stability and security properties of blockchain protocols were first rigorously formulated 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 typically possess the following basic 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 continuously. 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.

A particularly challenging design aspect is that the above probabilistic mechanism should be designed so that the adversary cannot bias it to its advantage. As the stake shifts, together with the evolving population of stakeholders, so does the honest majority assumption, and hence the function that appoints stakeholders should continuously take the ledger status into account. Preventing the biasing of the election mechanism in a context of a blockchain protocol is a delicate task that so far has eluded a practical solution that is secure against all 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 builds on some of the core combinatorial arguments that were developed to analyze that scheme. Nevertheless,

[⋆] Tokyo Institute of Technology and IOHK, bdavid@c.titech.ac.jp.
[⋆⋆] 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.

the protocol construction has a number of novel elements that require a significant recasting and generalization of the previous combinatorial analysis. In more detail, 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". A special feature of this VRF primitive, 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 an adversary has subverted the key generation procedure. We call such VRF functions "VRF with unpredictability under malicious key generation" and we present a strong embodiment of this notion with a novel Universal Composable (UC) formulation. We also present a very efficient realization of this primitive under the Computational Diffie Hellman (CDH) assumption in the random oracle model. Beyond this VRF notion, we also formalize in a UC fashion key evolving signatures that provide the forward security that is necessary for handling the adaptive corruption setting.

In more detail, we analyze our protocol in the *partial* or *semi-synchronous* model [DLS88,PSS17]. In this setting, we still divide the protocol execution in time units which, as in [KRDO17], are called slots, but there is a maximum delay of Δ slots that is applied to message delivery and it is unknown to the protocol participants.[1] 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", meaning "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 D, the protocol of [KRDO17] and its analysis are 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 relevant to 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 must permit the set of stakeholders to evolve over time and appropriately adapt our honest stakeholder majority assumption. Achieving this requires a bootstrapping argument that allows the protocol to continue unboundedly 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

[1] It is worth pointing out that the notion of slots we use in this work can be substantially shorter in terms of real time elapsed compared to the slots of [KRDO17], where each slot represented a full round of interaction between all participants.

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 subjects them to 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.

Note that implementing the beacon via hashing VRF values will make feasible a type of "grinding attack" where the adversary can trade hashing power for a slight bias of the protocol execution to its advantage. We show how this bias can be controlled by suitably increasing the relevant parameters depending on the hashing power that is available to the adversary.

Comparison to related work. The idea of proof-of-stake protocols has been discussed extensively in the bitcoin forum.[2] The manner that a stakeholder determines eligibility to issue a block is always publicly verifiable and the proof of eligibility is either computed publicly (via a calculation that is verifiable by repeating it) or by using a cryptographic mechanism that involves a secret-key computation and a public-key verification. The first example of the former approach appeared in PPCoin [KN12], and was followed by others including Ouroboros and Snow White [BGM14,KRDO17,DPS16]; while the first example of the latter approach (that we also employ in our work) appeared in NXT (cf. Section 2.4.1 of [Com14]) and was then also used elsewhere, most notably in Algorand [Mic16]. The virtue of the latter approach is exactly in its potential to control adaptive corruptions: due to the fact that the adversary cannot predict the eligibility of a stakeholder to issue a block prior to corrupting it, she cannot gain an advantage by directing its corruption quota to specific stakeholders. Nevertheless, none of these previous works isolated explicitly the properties of the primitives that are required to provide a full proof of security in the setting of adaptive corruptions. Injecting high quality randomness in the PoS blockchain was proposed by Bentov et al. [BLMR14,BGM16], though their proposal does not have a full formal analysis. The Ouroboros proof-of-stake protocol [KRDO17] is provably secure in a corruption model that excludes fully adaptive attacks by imposing a corruption delay on the corruption requests of the adversary. The Snow White proof-of-stake [DPS16] is the first to prove security in the Δ-semi-synchronous model but—as in the case of Ouroboros—adopts a weak adaptive corruption model.

A recent work close to ours is Algorand [Mic16] that also provides a proof-of-stake ledger that is adaptively secure. It follows an entirely different construction approach that runs a Byzantine agreement protocol for every block and achieves adaptive-corruption security via a novel, appealing concept of player-replaceability. However, Algorand is only secure against a 1/3 adversary bound; and while the protocol itself is very efficient, it yields an inherently slower block production rate compared to an "eventual consensus" protocol (like Bitcoin, Snow White, and Ouroboros). In principle, proof-of-stake blockchain protocols can advance at the theoretical maximum speed (of one block per communication round), while protocols relying on Byzantine agreement, like Algorand, would require a larger number of rounds to settle each block.

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 (i.e., permissioned) 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.

[2] Refer e.g., to the posts by QuantumMechanic and others from 2011 `https://bitcointalk.org/index.php?topic=27787.0` (Last Accessed 19/09/2017).

Note that the possibility of adversarial grinding in Ouroboros Praos is also present in previous work that derives randomness by hashing [Mic16,DPS16], as opposed to a dedicated coin-tossing protocol as in [KRDO17]. Following the examples of [Mic16,DPS16], we show that security can be guaranteed despite any adversarial bias resulting from grinding. In fact, we show how to use the q-bounded model of [GKL15] to derive a bound that shows how to increase the relevant security parameters given the hashing power that is available to the adversary.

Finally, in the present exposition we also put aside incentives; nevertheless, it is straightforward to adapt the mechanism of input endorsers from the 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. We let $v \,\|\, w$ denote concatenation of strings.

2.1 Transaction Ledger Properties

We adopt the same definitions for transaction ledger properties as [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 $\text{len}(\mathcal{C}_2) - \text{len}(\mathcal{C}_1) \geq \tau \cdot s$. We call τ the speed coefficient.

2.2 The Semi-Synchronous Model

On a high level, we consider the security model of [KRDO17] with simple modifications 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.

Time and slots. We consider a setting where time is divided into discrete units called *slots*. A ledger, described in more detail above, 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 two properties: (1) The current slot is determined by a publicly-known and monotonically increasing function of current time. (2) 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 note that multiple different "functionalities" can be encompassed by $\mathcal{F}$. In our model we employ the "Delayed Diffuse" functionality, which allows for adversarially-controlled delayed delivery of messages diffused among stakeholders.

The Diffuse Functionality. This functionality is parameterized by $\Delta \in \mathbb{N}$ and denoted as $\mathsf{DDiffuse}_\Delta$. It keeps rounds, executing one round per slot. $\mathsf{DDiffuse}_\Delta$ interacts with the environment $\mathcal{Z}$, stakeholders $U_1, \ldots, U_n$ and an adversary $\mathcal{A}$, working as follows for each round:

1. $\mathsf{DDiffuse}_\Delta$ maintains an incoming string for each party U_i that participates. A party, if activated, is allowed at any moment to fetch the contents of its incoming string, hence one may think of this as a mailbox. Furthermore, parties can give an instruction to the functionality to diffuse a message. Activated parties are allowed to diffuse once in a round.
2. When the adversary $\mathcal{A}$ is activated, it is allowed to: (a) Read all inboxes and all diffuse requests and deliver messages to the inboxes in any order it prefers; (b) 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; (c) For any party U_i, $\mathcal{A}$ can move any message from the string $\mathsf{delayed}_i$ to the inbox of U_i.
3. At the end of each round, the functionality also ensures that 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 is delivered to 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.
4. Upon receiving $(\mathsf{Create}, U, \mathcal{C})$ from the environment, the functionality spawns a new stakeholder with chain $\mathcal{C}$ as its initial local chain (as it was the case in [KRDO17]).

Modelling Protocol Execution and Adaptive Corruptions. Given the above we will assume that the execution of the protocol is with respect to a functionality $\mathcal{F}$ that incorporates $\mathsf{DDiffuse}$ as well as possibly additional functionalities to be explained in the following sections. The environment issues transactions on behalf of any stakeholder U_i by requesting a signature on the transaction as described in Protocol π_{SPoS} of Figure 4 and handing the transaction to stakeholders to put them into blocks. Beyond any restrictions imposed by $\mathcal{F}$, the adversary can only corrupt a stakeholder U_i if it is given permission by the environment $\mathcal{Z}$ running the protocol execution. The permission is in the form of a message $(\mathsf{Corrupt}, U_i)$ which is provided to the adversary by the environment. Upon receiving permission from the environment, the adversary immediately corrupts U_i without any delay, differently from [KRDO17,DPS16], where corruptions only take place after a given delay. Note that a corrupted stakeholder U_i will relinquish its entire state to $\mathcal{A}$; from this point on, the adversary will be activated in place of the stakeholder U_i. The adversary is able to control transactions and blocks generated by corrupted parties by interacting with $\mathcal{F}_{\mathsf{DSIG}}$,$\mathcal{F}_{\mathsf{KES}}$ and $\mathcal{F}_{\mathsf{VRF}}$, as described in Protocol π_{SPoS} of Section 3. In summary, regarding activations we have the following: (a) At each slot sl_j, the environment $\mathcal{Z}$ activates all honest stakeholders.[3] (b) The adversary is activated at least as the last entity in each sl_j (as well as during all adversarial party activations and invocations from the ideal functionalities as prescribed); (c) If a stakeholder does not fetch in a certain slot the messages written to its incoming string from the diffuse functionality they are flushed.

[3] We assume this to simplify our formal treatment, a variant of our protocol can actually accomodate "lazy honesty" as introduced in [Mic16]. In this variant, honest stakeholders only come online at the beginning of each epoch and at a few infrequent, predictable moments, see Appendix H.

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. We require 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. When the environment spawns a new stakeholder by sending message $(\mathsf{Create}, U, \mathcal{C})$ to the Key and Transaction functionality, the initial local chain $\mathcal{C}$ can be the chain of any honest stakeholder even in the case of "lazy honest" stakeholders as described in Appendix H, without requiring this stakeholder to have been online in the past slot as in [KRDO17]. 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. 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].

3 The Static Stake Protocol

We first consider the static stake case, where the stake distribution is fixed throughout protocol execution. The general structure of the protocol in the semi-synchronous model is similar to that of (synchronous) Ouroboros [KRDO17] but introduces several fundamental modifications to the leader selection process: not all slots will be attributed a slot leader, some slots might have multiple slot leaders, 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 strategically corrupt coalitions of parties with large (future) influence. Moreover, instead of using concrete instantiations of the necessary building blocks, we describe the protocol with respect to *ideal functionalities*, which we later realize with concrete constructions. This difference allows us to reason about security in the ideal model through a combinatorial argument without having to deal with the probability that the cryptographic building blocks fail. 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].

In the static stake case, we assume that a fixed collection of n stakeholders $U_1, \ldots, U_n$ interact throughout the protocol. Stakeholder U_i is attributed stake s_i at the beginning of the protocol.

Definition 1 (Genesis Block). *The* genesis block B_0 *contains the list of stakeholders identified by a label U_i, their respective public keys and respective stakes*

$$\mathbb{S}_0 = \left((U_1, v_1^{\text{vrf}}, v_1^{\text{kes}}, v_1^{\text{dsig}}, s_1), \ldots, (U_n, v_n^{\text{vrf}}, v_n^{\text{kes}}, v_n^{\text{dsig}}, s_n)\right) ,$$

and a nonce η.

We note that the nonce η will be used to seed the slot leader election process and that $v_i^{\text{vrf}}, v_i^{\text{kes}}, v_i^{\text{dsig}}$ will be determined by $\mathcal{F}_{\mathsf{VRF}}$, $\mathcal{F}_{\mathsf{KES}}$ and $\mathcal{F}_{\mathsf{DSIG}}$, respectively.

Definition 2 (State, Block Proof, Block, Blockchain, Epoch). *A* state *is a string $st \in \{0,1\}^\lambda$. A* block proof *is a value (or set of values) B_π containing information that allows stakeholders to verify if a block is valid. A* block $B = (sl_j, st, d, B_{\pi j}, \sigma_j)$ *generated at a slot $sl_j \in \{sl_1, \ldots, sl_R\}$ 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 σ_j, a signature on $(st, d, sl_j, B_{\pi j})$ under the signing key for the time period of slot sl_j of the stakeholder U_i generating the block.

A blockchain *(or simply* chain*) relative to the genesis block B_0 is a sequence of blocks $B_1, \ldots, B_n$ associated with a strictly increasing sequence of slots for which the state st_i of B_i is equal to $H(B_{i-1})$, where H is a prescribed collision-resistant hash function. The* length *of a chain* $\text{len}(\mathcal{C}) = n$ *is its number of blocks. The block B_n is the* head *of the chain, denoted* $\text{head}(\mathcal{C})$*. We treat the empty string ε as a legal chain and by convention set* $\text{head}(\varepsilon) = \varepsilon$*. Let $\mathcal{C}$ be a chain of length n and k be any non-negative integer. We denote by $\mathcal{C}^{\lceil k}$ the chain resulting from removal of the k rightmost blocks of $\mathcal{C}$. If $k \geq \text{len}(\mathcal{C})$ we define $\mathcal{C}^{\lceil k} = \varepsilon$. We let $\mathcal{C}_1 \preceq \mathcal{C}_2$ indicate that the chain $\mathcal{C}_1$ is a prefix of the chain $\mathcal{C}_2$.*

An epoch *is a set of R adjacent slots $S = \{sl_1, \ldots, sl_R\}$. (The value R is a parameter of the protocol we analyze in this section.)*

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 in Section 3.3 we discuss slot leader sets and how they are selected.

Definition 3 (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 $s_X^{\mathrm{s}}, \alpha_X^{\mathrm{s}}$ 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 Forward Secure Signatures and $\mathcal{F}_{\mathsf{KES}}$

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] but no previous work has fully specified them in the UC setting. Previous (game-based) definitions are presented in Appendix A.3.

We present a UC definition of the type of key-evolving signatures that we will take advantage of in our constructions. $\mathcal{F}_{\mathsf{KES}}$ allows us to achieve forward security with erasures (*i.e.*, assuming that parties securely delete old signing keys as the protocol proceeds). This functionality embodies ideal key evolving signature schemes allowing an adversary that corrupts the signer to forge signatures only under the current and future signing keys, but not under a previous signing key that has been updated. Our starting point for $\mathcal{F}_{\mathsf{KES}}$ is the standard digital signature functionality defined in [Can04] with the difference that packs together with the signing operation a key-evolving operation. During verification, $\mathcal{F}_{\mathsf{KES}}$ lets the adversary set the response to a verification query (taking as input a given time period) only if no key update has been performed since that time period and no entry exists in its internal table for the specific message, signature and time period specified in the query. We present $\mathcal{F}_{\mathsf{KES}}$ in Figure 1. In Appendix B, we will show that $\mathcal{F}_{\mathsf{KES}}$ can be realized by a construction based on key evolving signature schemes as defined in Appendix A.3.

Theorem 1. *The π_{KES} construction presented in Appendix B, realizes $\mathcal{F}_{\mathsf{KES}}$ with erasures assuming $\mathsf{KES} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update})$ is a key evolving signature scheme with forward security as per Definition 15 and Definition 17.*

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

$\mathcal{F}_{\mathsf{KES}}$ is parameterized by the total number of signature updates T, interacting with a signer U_S and stakeholders U_i as follows:

- **Key Generation.** Upon receiving a message $(\mathsf{KeyGen}, sid, U_S)$ from a stakeholder U_S , send $(\mathsf{KeyGen}, sid, U_S)$ to the adversary. Upon receiving $(\mathsf{VerificationKey}, sid, U_S, v)$ from the adversary, send $(\mathsf{VerificationKey}, sid, v)$ to U_S, record the triple (sid, U_S, v) and set counter $\mathsf{k}_{\mathsf{ctr}} = 1$.
- **Sign and Update.** Upon receiving a message $(\mathsf{USign}, sid, U_S, m, j)$ from U_S, verify that (sid, U_S, v) is recorded for some sid and that $\mathsf{k}_{\mathsf{ctr}} \leq j \leq T$. If not, then ignore the request. Else, set $\mathsf{k}_{\mathsf{ctr}} = j + 1$ and send $(\mathsf{Sign}, sid, U_S, m, j)$ to the adversary. Upon receiving $(\mathsf{Signature}, sid, U_S, m, j, \sigma)$ from the adversary, verify that no entry $(m, j, \sigma, v, 0)$ is recorded. If it is, then output an error message to U_S and halt. Else, send $(\mathsf{Signature}, sid, m, j, \sigma)$ to U_S, and record the entry $(m, j, \sigma, v, 1)$.
- **Signature Verification.** Upon receiving a message $(\mathsf{Verify}, sid, m, j, \sigma, v')$ from some stakeholder U_i do:
  1. If $v' = v$ and the entry $(m, j, \sigma, v, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds.)
  2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, j, \sigma', v, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, j, \sigma, v, 0)$. (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
  3. Else, if there is an entry (m, j, σ, v', f') recorded, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
  4. Else, if $j < \mathsf{k}_{\mathsf{ctr}}$, let $f = 0$ and record the entry $(m, j, \sigma, v, 0)$. Otherwise, if $j = \mathsf{k}_{\mathsf{ctr}}$, hand $(\mathsf{Verify}, sid, m, j, \sigma, v')$ to the adversary. Upon receiving $(\mathsf{Verified}, sid, m, j, \phi)$ from the adversary let $f = \phi$ and record the entry (m, j, σ, v', ϕ). (This condition guarantees that the adversary is only able to forge signatures under keys belonging to corrupted parties for time periods corresponding to the current or future slots.)

  Output $(\mathsf{Verified}, sid, m, j, f)$ to U_i.

Fig. 1: Functionality $\mathcal{F}_{\mathsf{KES}}$.

3.2 UC-VRFs with Unpredictability Under Malicious Key Generation

The usual pseudorandomness definition for VRFs (as stated in Appendix A.1, Definition 14) captures the fact that an attacker, seeing a number of VRF outputs and proofs for adversarially chosen inputs under a 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. This definition is too weak for our purposes for two reasons: first, we need a simulation-based definition so that the VRF can be composed directly within our protocol; second, we need the primitive to provide some level of unpredictability even under malicious key generation, *i.e.,* against adversaries who are allowed to generate the secret and pubic key pair.

Our UC formulation of VRFs cannot be implied by the standard VRF security definition or even the simulatable VRF notion of [CL07]. For instance, the VRF proofs in our setting have to be simulatable without knowlege of the VRF output (which is critical as we would like to ensure that the VRF output is not leaked to the adversary prematurely); it is easy to construct a VRF that is secure in the standard definition, but it is impossible to simulate its proofs without knowledge of the VRF output. Furthermore, if the adversary is allowed to generate its own key pair it is easy to see that the distribution of the VRF outputs cannot be guaranteed. Indeed, even for known constructions (*e.g.* [DY05]), an adversary that maliciously generates keys can easily and significantly skew the output distribution.

We call the latter property *unpredictability under malicious key generation* and we present, in Figure 2, a UC definition for VRF's that captures this stronger security requirement.[4] The functionality operates as follows. Given a key generation request from one of the stakeholders,

[4] In fact our UC formulation captures a stronger notion: even for adversarial keys the VRF function will act as a random oracle. We note that while we can achieve this notion in the random oracle model, a weaker condition of mere unpredictability can be sufficient for the security of our protocol. A UC version

it returns a new verification key v that is used to label a table. Two methods are provided for computing VRF values. The first provides just the VRF output and does not interact with the adversary. In the second, whenever invoked on an input m that is not asked before by a stakeholder that is associated to a certain table labeled by v, the functionality will query the adversary for the value of the proof π, and subsequently sample a random element ρ to associate with m. Verification is always consistent and will validate outputs that have already being inserted in a table. Unpredictability against malicious key generation is captured by imposing the same random selection of outputs even for the function tables that correspond to keys of corrupted stakeholders. Finally, the adversary is allowed to query all function tables maintained by the functionality for which either a proof has been computed, or they correspond to adversarial keys. In Appendix C, we show how to realize $\mathcal{F}_{\mathsf{VRF}}$ in the random oracle model under the CDH assumption based on the 2-Hash-DH verifiable oblivious PRF construction of [JKK14].

Functionality $\mathcal{F}_{\mathsf{VRF}}$.

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

- **Key Generation.** Upon receiving a message (KeyGen, sid) from a stakeholder U_i, hand $(\mathsf{KeyGen}, sid, U_i)$ to the adversary. Upon receiving $(\mathsf{VerificationKey}, sid, U_i, v)$ from the adversary, if U_i is honest, verify that v is unique, record the pair (U_i, v) and return $(\mathsf{VerificationKey}, sid, v)$ to U_i. Initialize the table $T(v, \cdot)$ to empty.
- **Malicious Key Generation.** Upon receiving a message $(\mathsf{KeyGen}, sid, v)$ from $\mathcal{S}$, verify that v has not being recorded before; in this case initialize table $T(v, \cdot)$ to empty and record the pair $(\mathcal{S}, v)$.
- **VRF Evaluation.** Upon receiving a message (Eval, sid, m) from U_i, verify that some pair (U_i, v) is recorded. If not, then ignore the request. Then, if the value $T(v, m)$ is undefined, pick a random value y from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set $T(v, m) = (y, \emptyset)$. Then output $(\mathsf{Evaluated}, sid, y)$ to P, where y is such that $T(v, m) = (y, S)$ for some S.
- **VRF Evaluation and Proof.** Upon receiving a message $(\mathsf{EvalProve}, sid, m)$ from U_i, verify that some pair (U_i, v) is recorded. If not, then ignore the request. Else, send $(\mathsf{EvalProve}, sid, U_i, m)$ to the adversary. Upon receiving $(\mathsf{Eval}, sid, m, \pi)$ from the adversary, if value $T(v, m)$ is undefined, verify that π is unique, pick a random value y from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set $T(v, m) = (y, \{\pi\})$. Else, if $T(v, m) = (y, S)$, set $T(v, m) = (y, S \cup \{\pi\})$. In any case, output $(\mathsf{Evaluated}, sid, y, \pi)$ to P.
- **Malicious VRF Evaluation.** Upon receiving a message $(\mathsf{Eval}, sid, v, m)$ from $\mathcal{S}$ for some v, do the following. First, if $(\mathcal{S}, v)$ is recorded and $T(v, m)$ is undefined, then choose a random value y from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set $T(v, m) = (y, \emptyset)$. Then, if $T(v, m) = (y, S)$ for some $S \neq \emptyset$, output $(\mathsf{Evaluated}, sid, y)$ to $\mathcal{S}$, else ignore the request.
- **Verification.** Upon receiving a message $(\mathsf{Verify}, sid, m, y, \pi, v')$ from some party P, send $(\mathsf{Verify}, sid, m, y, \pi, v')$ to the adversary. Upon receiving $(\mathsf{Verified}, sid, m, y, \pi, v')$ from the adversary do:
  1. If $v' = v$ for some (U_i, v) and the entry $T(U_i, m)$ equals (y, S) with $\pi \in S$, then set $f = 1$.
  2. Else, if $v' = v$ for some (U_i, v), but no entry $T(U_i, m)$ of the form $(y, \{\ldots, \pi, \ldots\})$ is recorded, then set $f = 0$.
  3. Else, initialize the table $T(v', \cdot)$ to empty, and set $f = 0$.

  Output $(\mathsf{Verified}, sid, m, y, \pi, f)$ to P.

Fig. 2: Functionality $\mathcal{F}_{\mathsf{VRF}}$.

Theorem 2. *The 2Hash-DH construction presented in Appendix C, realizes $\mathcal{F}_{\mathsf{VRF}}$ in the random oracle model assuming the CDH.*

3.3 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 Ouroboros [KRDO17]

of the notion of verifiable pseudorandom permutations, cf. [DP07], could potentially be used towards a standard model instantiation of the primitive.

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 is 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 Section 4 shows (with an honest stake majority) that (i.) blockchains 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*.

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

$\mathcal{F}_{\mathsf{INIT}}$ incorporates the delayed diffuse functionality from Section 2.2 and is parameterized by the number of initial stakeholders n and their respective stakes $s_1, \ldots, s_n$. $\mathcal{F}_{\mathsf{INIT}}$ interacts with stakeholders $U_1, \ldots, U_n$ as follows:

- In the first round, upon a request from some stakeholder U_i of the form $(\mathsf{ver\_keys}, sid, U_i, v_i^{\mathrm{vrf}}, v_i^{\mathrm{kes}}, v_i^{\mathrm{dsig}})$, it stores the verification keys tuple $(U_i, v_i^{\mathrm{vrf}}, v_i^{\mathrm{kes}}, v_i^{\mathrm{dsig}})$ and acknowledges its receipt. If any of the n stakeholders does not send a request of this form to $\mathcal{F}_{\mathsf{INIT}}$, it halts. Otherwise, it samples and stores a random value $\eta \xleftarrow{\$} \{0,1\}^{\lambda}$ and constructs a genesis block $(\mathbb{S}_0, \eta)$, where $\mathbb{S}_0 = \Big((U_1, v_1^{\mathrm{vrf}}, v_1^{\mathrm{kes}}, v_1^{\mathrm{dsig}}, s_1), \ldots, (U_n, v_n^{\mathrm{vrf}}, v_n^{\mathrm{kes}}, v_n^{\mathrm{dsig}}, s_n)\Big)$.
- In later rounds, upon a request of the form $(\mathsf{genblock\_req}, sid, U_i)$ from some stakeholder U_i, $\mathcal{F}_{\mathsf{INIT}}$ sends $(\mathsf{genblock}, sid, \mathbb{S}_0, \eta)$ to U_i.

Fig. 3: Functionality $\mathcal{F}_{\mathsf{INIT}}$.

The idealized slot leader assignment and the active slots coefficient. The fundamental leader assignment process calls for a stakeholder U_i to be independently selected as a leader for a particular slot sl_j with probability p_i depending only on its relative stake. (In this static-stake analysis, relative stake is simply determined by the genesis block B_0.) The exact relationship between p_i and the relative stake α_i is determined by a parameter f of the protocol which we refer to as the *active slots coefficient*. Specifically,

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

where α_i is the relative stake held by stakeholder U_i. We occasionally drop the subscript f and write $\phi(\alpha_i)$ when f can be inferred from context. As the events "U_i is a leader for sl_j" are independent, this process may indeed generate multiple (or zero) leaders for a given slot.

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 7. To motivate the choice of the function ϕ_f, we note that it satisfies the "independent aggregation" property:

$$1 - \phi\left(\sum_i \alpha_i\right) = \prod_i (1 - \phi(\alpha_i)) \ . \tag{2}$$

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 (2) 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.*

3.4 The Protocol in the $\mathcal{F}_{\mathsf{INIT}}$-hybrid Model

We will construct our protocol for the static stake 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 $\mathcal{F}_{\mathsf{INIT}}$ defined in Figure 3. Moreover, $\mathcal{F}_{\mathsf{INIT}}$ also incorporates the diffuse functionality from Section 2.2, which is implicitly used by all parties to send messages and keep synchronized with a global clock. $\mathcal{F}_{\mathsf{INIT}}$ also takes stakeholders' public keys from them and packages them into the genesis block at the outset of the protocol. Blocks are signed with a forward secure signature scheme modelled by $\mathcal{F}_{\mathsf{KES}}$, while transactions are signed with a regular $\mathsf{EUF\text{-}CMA}$ secure digital signature modelled by $\mathcal{F}_{\mathsf{DSIG}}$ (described in Appendix A). For simplicity, transactions are assumed to be simple assertions of the form "Stakeholder U_i transfers stake s to Stakeholder $(U_j, v_j^{\mathrm{vrf}}, v_j^{\mathrm{kes}}, v_j^{\mathrm{dsig}})$" (In an implementation the different public-keys can be hashed into a single value). Protocol π_{SPoS} ensures that the environment learns every stakeholder's public keys and provides an interface for the environment to request signatures on arbitrary transactions.

Notice that the implicit leader assignment process described in π_{SPoS} calls for a party U_i to act as a leader for a slot sl_j when $y < T_i$; this is an event that occurs with probability (exponentially close to) $\phi_f(\alpha_i)$ as y is uniform according to the functionality $\mathcal{F}_{\mathsf{VRF}}$. The stakeholders $U_1, \ldots, U_n$ interact among themselves and with $\mathcal{F}_{\mathsf{INIT}}$ through Protocol π_{SPoS} described in Figure 4. 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 stake 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.

4 Combinatorial Analysis of the Static Stake Protocol

Throughout this section, we focus solely on analysis of the protocol π_{SPoS} using the idealized functionalities $\mathcal{F}_{\mathsf{VRF}}$ and $\mathcal{F}_{\mathsf{KES}}$ for VRFs and digital signatures, respectively—we refer to it as the *hybrid experiment*. As argued in Theorems 1 and 2, any property of the protocol that we prove true in the hybrid experiment (such as achieving common prefix, chain growth and chain quality) will remain true (with overwhelming probability) in the setting where $\mathcal{F}_{\mathsf{VRF}}$ and $\mathcal{F}_{\mathsf{KES}}$ are replaced by their real-world implementations—in the so-called *real experiment*.

The hybrid experiment yields a stochastic process for assigning slots to parties which we now abstract and study 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—forks are a natural bookkeeping tool that reflect 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 special case of 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, chain growth, and chain quality (4.4) guarantees for the semi-synchronous protocol with respect to an adaptive adversary.

We begin in Section 4.1 with a discussion of characteristic strings, semi-synchronous forks, and their relationship to executions of π_{SPoS} in the hybrid experiment. Section 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 Section 4.3, where the effect of the reduction on this distribution is also described. Section 4.4, as described above, establishes various guarantees on the resulting blockchain under the dominant distribution. The full power of adaptive adversaries is considered in Section 4.5. Finally, in preparation for applying the 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.

Protocol π_{SPoS}

The protocol π_{SPoS} is run by stakeholders $U_1, \ldots, U_n$ interacting among themselves and with ideal functionalities $\mathcal{F}_{\mathsf{INIT}}, \mathcal{F}_{\mathsf{VRF}}, \mathcal{F}_{\mathsf{KES}}, \mathcal{F}_{\mathsf{DSIG}}, \mathsf{H}$ over a sequence of slots $S = \{sl_1, \ldots, sl_R\}$. Define $T_i \triangleq 2^{\ell_{\mathsf{VRF}}} \phi_f(\alpha_i)$ as the threshold for a stakeholder U_i, where α_i is the relative stake of U_i, ℓ_{VRF} denotes the output length of $\mathcal{F}_{\mathsf{VRF}}$, f is the active slots coefficient and ϕ_f is the mapping from Definition 1. Then π_{SPoS} proceeds as follows:

1. **Initialization.** The stakeholder U_i sends $(\mathsf{KeyGen}, sid, U_i)$ to $\mathcal{F}_{\mathsf{VRF}}$, $\mathcal{F}_{\mathsf{KES}}$ and $\mathcal{F}_{\mathsf{DSIG}}$; receiving $(\mathsf{VerificationKey}, sid, v_i^{\mathrm{vrf}})$, $(\mathsf{VerificationKey}, sid, v_i^{\mathrm{kes}})$ and $(\mathsf{VerificationKey}, sid, v_i^{\mathrm{dsig}})$, respectively. Then, in case it is the first round, it sends $(\mathsf{ver\_keys}, sid, U_i, v_i^{\mathrm{vrf}}, v_i^{\mathrm{kes}}, v_i^{\mathrm{dsig}})$ to $\mathcal{F}_{\mathsf{INIT}}$ (to claim stake from the genesis block). In any case, it terminates the round by returning $(U_i, v_i^{\mathrm{vrf}}, v_i^{\mathrm{kes}}, v_i^{\mathrm{dsig}})$ to $\mathcal{Z}$. In the next round, it sends $(\mathsf{genblock\_req}, sid, U_i)$ to $\mathcal{F}_{\mathsf{INIT}}$, receiving $(\mathsf{genblock}, sid, \mathbb{S}_0, \eta)$ as the answer. If U_i is initialized in the first round, it sets the local blockchain $\mathcal{C} = B_0 = (\mathbb{S}_0, \eta)$ and its initial internal state $st = H(B_0)$. In case U_i is initialized after the first round, it sets its initial state to $st = H(\mathrm{head}(\mathcal{C}))$ where $\mathcal{C}$ is the initial local chain provided by the environment.
2. **Chain Extension.** After initialization, for every slot $sl_j \in S$, every online stakeholder U_i performs the following steps:
   (a) U_i receives from the environment the transaction data $d \in \{0,1\}^*$ to be inserted into the blockchain.
   (b) U_i collects all valid chains received via diffusion 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 who created it is in the slot leader set of slot sl' (by parsing $B_\pi{}'$ as (U_s, y', π') for some s, verifying that $\mathcal{F}_{\mathsf{VRF}}$ responds to $(\mathsf{Verify}, sid, \eta \,\|\, sl', y', \pi', v_s^{\mathrm{vrf}})$ by $(\mathsf{Verified}, sid, \eta \,\|\, sl', y', \pi', 1)$, and that $y' < T_s$), and that $\mathcal{F}_{\mathsf{KES}}$ responds to $(\mathsf{Verify}, sid, (st', d', sl', B_\pi{}'), sl', \sigma_{j'}, v_s^{\mathrm{kes}})$ by $(\mathsf{Verified}, sid, (st', d', sl', B_\pi{}'), sl', 1)$. U_i computes $\mathcal{C}' = \mathsf{maxvalid}(\mathcal{C}, \mathbb{C})$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\mathrm{head}(\mathcal{C}'))$.
   (c) U_i sends $(\mathsf{EvalProve}, sid, \eta \,\|\, sl_j)$ to $\mathcal{F}_{\mathsf{VRF}}$, receiving $(\mathsf{Evaluated}, sid, y, \pi)$. U_i checks whether it is in the slot leader set of slot sl_j by checking that $y < T_i$. If yes, it generates a new block $B = (st, d, sl_j, B_\pi, \sigma)$ where st is its current state, $d \in \{0,1\}^*$ is the transaction data, $B_\pi = (U_i, y, \pi)$ and σ is a signature obtained by sending $(\mathsf{USign}, sid, U_i, (st, d, sl_j, B_\pi), sl_j)$ to $\mathcal{F}_{\mathsf{KES}}$ and receiving $(\mathsf{Signature}, sid, (st, d, sl_j, B_\pi), sl_j, \sigma)$. U_i computes $\mathcal{C}' = \mathcal{C}|B$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\mathrm{head}(\mathcal{C}'))$. Finally, if U_i has generated a block in this step, it diffuses $\mathcal{C}'$.
3. **Signing Transactions.** Upon receiving $(\mathsf{sign\_tx}, sid', tx)$ from the environment, U_i sends $(\mathsf{Sign}, sid, U_i, tx)$ to $\mathcal{F}_{\mathsf{DSIG}}$, receiving $(\mathsf{Signature}, sid, tx, \sigma)$. Then, U_i sends $(\mathsf{signed\_tx}, sid', tx, \sigma)$ back to the environment.

Fig. 4: Protocol π_{SPoS}.

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_{SPoS} in the hybrid experiment 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 (both points follow from the construction of π_{SPoS} and the independent random sampling of every new output in $\mathcal{F}_{\mathsf{VRF}}$). 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 described below.

Definition 4 (Execution). *For an (adaptive) adversary $\mathcal{A}$ and an environment $\mathcal{Z}$, an* execution *$\mathcal{E}$ of π_{SPoS} is a transcript including the inputs provided by $\mathcal{Z}$, the random coins of the parties, the random coins of the adversary, the responses of the ideal functionalities and the random oracle. This data determines the entire dynamics of the protocol: messages sent and delivered, the internal states of the parties at each step, the set of corrupt parties at each step, etc.*

Definition 5 (Characteristic string). *Let $S = \{sl_1, \ldots, sl_R\}$ be a sequence of slots of length R and $\mathcal{E}$ be an execution (with adversary $\mathcal{A}$ and environment $\mathcal{Z}$). For a slot sl_j, let $\mathcal{P}(j)$ denote the*

set of parties assigned to be slot leaders for slot j by the protocol π_{SPoS} (specifically, those parties U_i for which $y < 2^{\ell_{\mathsf{VRF}}}\phi_f(\alpha_i)$, where $(y,\pi) \leftarrow \mathsf{Prove}_{\mathsf{VRF}.sk_i}(\eta \,\|\, sl_j)$). We define the characteristic string *$w \in \{0,1,\perp\}^R$ of S to be the random variable so that*

$$w_j = \begin{cases} \perp & \text{if } \mathcal{P}(j) = \emptyset, \\ 0 & \text{if } |\mathcal{P}(j)| = 1 \text{ and the assigned party is honest,} \\ 1 & \text{if } |\mathcal{P}(j)| > 1 \text{ or a party in } \mathcal{P}(i) \text{ is adversarial.} \end{cases} \tag{3}$$

For such a characteristic string $w \in \{0,1,\perp\}^*$ we say that the index j is uniquely honest *if $w_j = 0$,* tainted *if $w_j = 1$, and* empty *if $w_j = \perp$. We say that an index is* active *if $w_j \in \{0,1\}$. 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 hybrid experiment with the active slots coefficient f, adversary $\mathcal{A}$, and environment $\mathcal{Z}$. For a fixed execution $\mathcal{E}$, we denote by $w_{\mathcal{E}}$ the (fixed) characteristic string resulting from that execution.

We emphasize that in an execution of π_{SPoS}, the resulting 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).

From executions to forks. The notion of a "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.

An execution of Protocol π_{SPoS} 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* associated with the block 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 verification criteria of π_{SPoS}.) Note that multiple blocks may be associated with a particular slot, either because multiple parties are assigned to the slot or an adversarial party is assigned to a slot (who may choose to deviate from the protocol by issuing multiple blocks). In any case, these blocks induce a natural directed tree by treating the 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 execution of π_{SPoS} a fork. While this fork disregards many of the details of the execution, any violations of common prefix are immediately manifested by certain 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_{SPoS} with an adversary $\mathcal{A}$ by establishing that such diverging paths cannot exist in their associated forks.

Definition 6 (Δ-fork). *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 $\mathbf{d}(i) < \mathbf{d}(j)$.*

As a matter of notation, we write $F \vdash_\Delta w$ to indicate that F is a Δ-fork for the string w. We typically refer to a Δ-fork as simply a "fork".

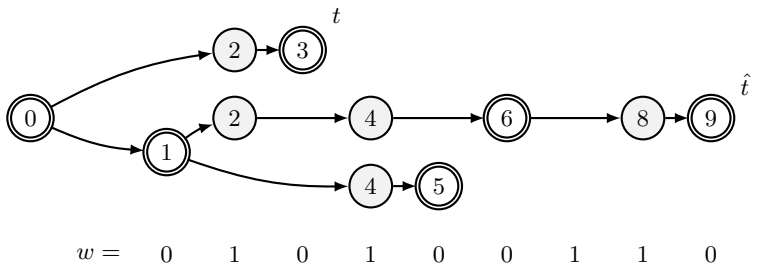

Fig. 5: A (synchronous) fork F for the string $w = 010100110$. Vertices appear with their labels and vertices belonging to (uniquely) honest slots are highlighted with double borders. Note that the depths of the (honest) vertices associated with the honest indices of w are strictly increasing. Two tines 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 $\operatorname{div}(t, \hat{t}) = 2$.

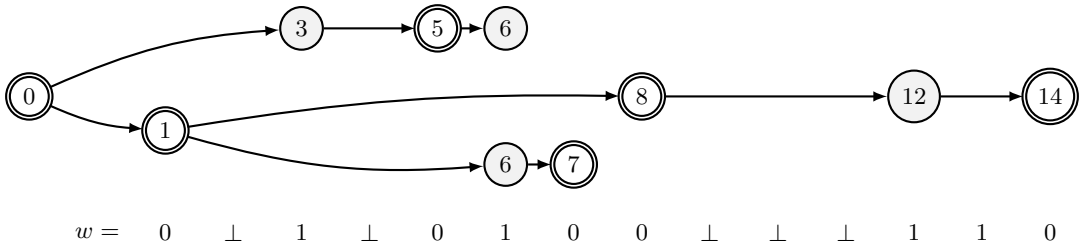

Fig. 6: A 3-fork F' for the characteristic string $w = 0\bot1\bot01001\bot\bot10$. Note that F' is not a 2-fork since $\mathbf{d}(8) = 2 \not> 2 = \mathbf{d}(5)$. Indices $\{1, 5, 7, 8, 14\}$ are uniquely honest, $\{3, 6, 12, 13\}$ are tainted, and $\{2, 4, 9, 10, 11\}$ are empty. The index 8 is 4-right-isolated, but not 5-right-isolated.

See Figures 5 and 6 for examples of forks. Also note that our notion of a fork deliberately models honest parties that do not exploit all the information available to them thanks to the delivery guarantees provided by the DDiffuse functionality. In particular, an honest party that is permanently online could (in our communication model) safely discard any (adversarial) blocks that were not received in the correct time window.

Nonetheless, it remains true that any execution of the hybrid experiment leads to a fork as we defined it, a relationship that we make fully formal in Appendix F. Given this relationship, we can later focus on investigating the properties of the distribution $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$. Roughly speaking, if we prove that a characteristic string sampled from $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$, with overwhelming probability, does not allow for *any* "harmful" forks, then this also implies that a random execution with overwhelming probability results in a "harmless" outcome.

Now we continue with the adaptation of the framework from [KRDO17] to the semi-synchronous setting.

Definition 7 (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 call the length of the tine terminating at v the* depth of 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* length$(t) \geq \max_{h+\Delta \leq \ell(t)} \mathbf{d}(h)$*, 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 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 8 (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: $\operatorname{div}_\Delta(F) \triangleq \max_{t_1,t_2} \operatorname{div}(t_1, t_2)$, taken over all pairs of Δ-viable tines of F. Finally, we define the Δ-divergence of a characteristic string w to be the maximum over all Δ-forks: $\operatorname{div}_\Delta(w) \triangleq \max_{F \vdash_\Delta w} \operatorname{div}_\Delta(F)$.

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

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 9 (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 3. *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 $\Pr[\operatorname{div}_0(w) \geq k] \leq \exp(\ln \ell - \Omega(k))$.*

4.2 The Semisynchronous to Synchronous Reduction

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

Definition 10 (Reduction mapping). *For $\Delta \in \mathbb{N}$, we define the function $\rho_\Delta \colon \{0,1,\perp\}^* \to \{0,1\}^*$ inductively as follows: $\rho_\Delta(\varepsilon) = \varepsilon$, $\rho_\Delta(\perp \,\|\, w') = \rho_\Delta(w')$,*

$$\begin{aligned} \rho_\Delta(1 \,\|\, w') &= 1 \,\|\, \rho_\Delta(w'), \\ \rho_\Delta(0 \,\|\, w') &= \begin{cases} 0 \,\|\, \rho_\Delta(w') & \text{if } w' \in \perp^{\Delta-1} \,\|\, \{0,1,\perp\}^*, \\ 1 \,\|\, \rho_\Delta(w') & \text{otherwise.} \end{cases} \end{aligned} \tag{4}$$

We call ρ_Δ the reduction mapping for delay Δ.

A critical feature of the map ρ_Δ is that it monotonically transforms Δ-divergence to synchronous divergence. We state this in the following lemma.

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

Proof. Let $w \in \{0,1,\perp\}^*$ be a characteristic string with $\mathrm{div}_\Delta(w) = k$ and let $F \vdash_\Delta w$ be a Δ-fork with $\mathrm{div}_\Delta(F) = k$. Let $w' = \rho_\Delta(w)$; to prove that $\mathrm{div}_0(w') \geq k$, we construct a fork $F' \vdash_0 w'$ for which $\mathrm{div}(F') \geq k$. Let $A = \{i \mid w_i \neq \perp\}$ denote the set of active indices (as in Definition 6) 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 6 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. [GKL15], 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 $\mathrm{div}_0(F') \geq \mathrm{div}(F) = k$, 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 $\mathrm{length}(t) \geq \mathbf{d}(h)$, 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. □

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 $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$ determined by the 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 static adversary. Later in Section 4.5 we show that the same is true also for adaptive adversaries. We then study the effect of ρ_Δ on this distribution in preparation for studying common prefix, chain growth, and chain quality.

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 $\alpha_\mathcal{A}$. (Formally, one can model this by restricting to environments that only allow static corruption.) Recalling Definition 1, a party with relative stake α_i is independently assigned to be a leader for a slot with probability

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

The function ϕ_f is concave since

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

Figure 7 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. This immediately implies the following proposition that will be useful during the analysis.

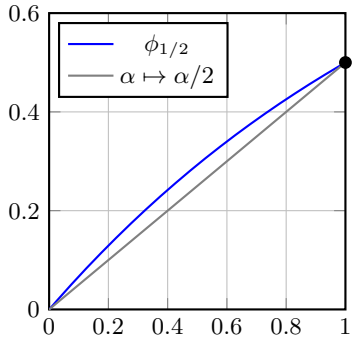

Fig. 7: The function $\phi_{1/2}(\alpha) = 1 - (1/2)^\alpha$ and the linear function $\alpha \mapsto \alpha/2$, for comparison. The point $(1, 1/2)$ is marked in solid black.

Proposition 1. *The function $\phi_f(\alpha)$ satisfies the following properties.*

$$\phi_f\left(\sum_i \alpha_i\right) = 1 - \prod_i (1 - \phi_f(\alpha_i)) \leq \sum_i \phi_f(\alpha_i)\,, \qquad \alpha_i \geq 0\,, \tag{5}$$

$$\frac{\phi_f(\alpha)}{\phi_f(1)} = \frac{\phi_f(\alpha)}{f} \geq \alpha\,, \qquad \alpha \in [0,1]\,. \tag{6}$$

Recalling Definition 5, 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

$$\begin{aligned} p_\perp^{\mathcal{A}} &\triangleq \Pr[w_i = \perp] = \prod_{i \in \mathcal{P}} (1 - \phi(\alpha_i)) = \prod_{i \in \mathcal{P}} (1-f)^{\alpha_i} = (1-f)\,, \\ p_0^{\mathcal{A}} &\triangleq \Pr[w_i = 0] = \sum_{h \in \mathcal{H}} (1 - (1-f)^{\alpha_h}) \cdot (1-f)^{1-\alpha_i}\,, \\ p_1^{\mathcal{A}} &\triangleq \Pr[w_i = 1] = 1 - p_\perp^{\mathcal{A}} - p_0^{\mathcal{A}}\,. \end{aligned} \tag{7}$$

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.

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

$$p_0^{\mathcal{A}} \geq \sum_{h \in \mathcal{H}} \phi(\alpha_h) \cdot \prod_{i \in \mathcal{P}} (1 - \phi(\alpha_i)) \geq \phi(\alpha_{\mathcal{H}}) \cdot p_\perp^{\mathcal{A}} = \phi(\alpha_{\mathcal{H}}) \cdot (1-f)\,, \tag{8}$$

where $\alpha_{\mathcal{H}}$ denotes the total relative stake of the honest parties. Note that this bound applies to all static adversaries $\mathcal{A}$ that corrupt no more than a $1 - \alpha_{\mathcal{H}}$ fraction of all stake. With this in mind, we define the dominant distribution as follows.

Definition 11 (The dominant distribution $\mathcal{D}^f_\alpha$). *For two parameters f and α, define $\mathcal{D}^f_\alpha$ to be the distribution on strings $w \in \{0,1,\perp\}^R$ that independently assigns each w_i so that $p_\perp \triangleq \Pr[w_i = \perp] = 1 - f$, $p_0 \triangleq \Pr[w_i = 0] = \phi(\alpha) \cdot (1-f)$, and $p_1 \triangleq \Pr[w_i = 1] = 1 - p_\perp - p_0$.*

The distribution $\mathcal{D}^f_\alpha$ "dominates" $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$ 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 $\mathcal{D}^f_\alpha$ than they are under $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$.

To make this relationship precise, we introduce the partial order $\preceq$ on the set $\{\perp, 0, 1\}$ 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, which follows directly from the definition of Δ-fork and $\operatorname{div}_\Delta$.

Lemma 2. *Let x and y be characteristic strings in $\{0,1,\perp\}^R$ for which $x \preceq y$. Then 1.) for every fork F, $F \vdash_\Delta x \implies F \vdash_\Delta y$; 2.) for every Δ, $\operatorname{div}_\Delta(x) \leq \operatorname{div}_\Delta(y)$.*

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

Definition 12 (Stochastic dominance). *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 $D_\Delta = \{x \mid \operatorname{div}_\Delta(x) \geq k\}$ which are monotone by Lemma 2. We note that any static adversary that corrupts no more than a $1-\alpha$ fraction of stake is α-dominated, and it follows that $\Pr_{\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}}[\operatorname{div}_\Delta(w) \geq k] \leq \Pr_{\mathcal{D}^f_\alpha}[\operatorname{div}_\Delta(w) \geq k]$. This motivates a particular study of the "dominant" distribution $\mathcal{D}^f_\alpha$.

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 (4) above implicitly defines a family of random variables $\rho_\Delta(w) = x_1 \dots x_\ell \in \{0,1\}^*$, where $w \in \{0,1,\perp\}^R$ is distributed according to $\mathcal{D}^f_\alpha$. Observe that $\ell = R - \#_\perp(w)$ 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.

Lemma 3 (Structure of the induced distribution). *Let $x_1 \dots x_\ell = \rho_\Delta(w)$ where $w \in \{0,1,\perp\}^R$ is distributed according to $\mathcal{D}^f_\alpha$. There is a sequence of independent random variables $z_1, z_2, \dots$ with each $z_i \in \{0,1\}$ so that*

$$\Pr[z_i = 0] = \left(\frac{p_0}{p_0 + p_1}\right) p_\perp^{\Delta-1} \geq \alpha \cdot (1-f)^\Delta, \tag{9}$$

$$\text{and} \qquad x_1 \dots x_{\ell-\Delta} = \rho_\Delta(w_1 \dots, w_R)^{\lceil \Delta} \quad \text{is a prefix of} \quad z_1 z_2 \dots. \tag{10}$$

(Note that while the z_i are independent with each other, they are not independent with w.)

Proof. It simplifies our analysis to treat w as the first R symbols of an infinite string $w_1 w_2 \dots$ of independent random variables with distribution given by Definition 11 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 $b_0 e_1 b_1 e_2 b_2 \dots$, where the (independent) random variables $e_i \in \{0,1\}$ and $b_i \in \{\perp\}^*$ have the probability laws

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

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 $z_1 z_2 \dots = \rho_\Delta(b_0 e_1 b_1 \dots)$—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

$$\Pr[z_i = 0] = \left(\frac{p_0}{p_0 + p_1}\right) p_\perp^{\Delta-1} = \frac{\phi(\alpha)}{f} \cdot (1-f)^\Delta \overset{(6)}{\geq} \alpha \cdot (1-f)^\Delta, \tag{11}$$

where $\overset{\text{(i)}}{\geq}$ follows from equation (i) and we use 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 observe (10):

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

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 $\approx 1 - \alpha(1-f)^\Delta$. □

Divergence for the Dominant Distribution. Our goal is to apply the reduction ρ_Δ, Lemma 1, and Theorem 3 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 3 directly. We resolve this obstacle in the proof of the following theorem.

Theorem 4. *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 $\Pr[\operatorname{div}_\Delta(w) \geq k + \Delta] = 2^{-\Omega(k) + \log R}$.*

Proof. 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 $\operatorname{div}_0(y) \leq \operatorname{div}_0(\check{y}) + s$; 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 1 we conclude that

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

where the last inequality follows because the random variable $\rho_\Delta(w_1 \ldots w_R)$ 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 now follows directly from the assumption that $\alpha(1-f)^\Delta \geq (1+\epsilon)/2$ and Theorem 3. □

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 $\alpha(1-f)^\Delta$.

4.4 Common Prefix, Chain Growth, and Chain Quality

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

Theorem 5 (Common prefix). *Let $k, R, \Delta \in \mathbb{N}$ and $\varepsilon \in (0,1)$. Let $\mathcal{A}$ be an α-dominated adversary against the protocol π_{SPoS} 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_{SPoS} 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_{SPoS} violates the common prefix property with parameter k precisely when the Δ-fork F induced by this execution has $\operatorname{div}_\Delta(F) \geq k$. We have

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

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

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

is monotone; and the last one follows from Theorem 4. (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.) □

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. 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 6 (Chain growth). *Let $k, R, \Delta \in \mathbb{N}$ and $\varepsilon \in (0,1)$. Let $\mathcal{A}$ be an α-dominated adversary against the protocol π_{SPoS} for some $\alpha > 0$. Then the probability that $\mathcal{A}$, when executed in a Δ-semi-synchronous environment, makes π_{SPoS} 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 $\exp(-c\alpha s/(20\Delta) + \ln R\Delta + O(1))$, where c denotes the constant $c := c(f,\Delta) = f(1-f)^\Delta$.*

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 $\mathrm{length}(\mathcal{C}_2) - \mathrm{length}(\mathcal{C}_1) \geq \tau s$.

Let $\hat{sl}_1, \ldots, \hat{sl}_h$ be the increasing sequence of all Δ-right-isolated uniquely honest slots among the slots in $T := \{sl_1 + \Delta, sl_1 + \Delta + 1, \ldots, sl_2 - \Delta\}$. 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 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 $\mathrm{length}(\mathcal{C}_2)$ will be at least as long as this chain. It follows that $\mathrm{length}(\mathcal{C}_2) - \mathrm{length}(\mathcal{C}_1) \geq h$.

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 $x \in \{0,1,\bot\}^R$, we hence have $h = H_T(x)$ for $x \leftarrow \mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$. Furthermore, let $E \triangleq \left\{x \in \{0,1,\bot\}^R \mid H_T(x) < c\alpha s/4\right\}$ 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 $\mu \triangleq \mathbb{E}[X_t] = p_0 p_\bot^{\Delta-1} \geq \alpha f(1-f)^\Delta$ according to Definition 11 and (6), 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 $T_0 \cup \cdots \cup T_{\Delta-1}$. 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

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

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

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

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

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

□

Our chain quality statement of Theorem 7 is a direct consequence of Lemma 4, which observes that a sufficiently long sequence of consecutive blocks in an honest party's chain will most likely contain a block created in a Δ-right-isolated uniquely honest slot.

Lemma 4. *Let $k, \Delta \in \mathbb{N}$ and $\epsilon \in (0,1)$. Let $\mathcal{A}$ be an α-dominated adversary against the protocol π_{SPoS} 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 6, 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 $E = \left\{x \in \{0,1,\bot\}^R \mid g(x) \leq b(x)\right\}$, 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 $\mathcal{D}_{\mathcal{Z},\mathcal{A}}^{f} \preceq \mathcal{D}_{\alpha}^{f}$ implies

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

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 (11), this implies that good slots are sampled with higher probability than bad slots. Therefore, the probability that $g(x) \leq b(x)$ for $x \leftarrow \mathcal{D}_{\alpha}^{f}$ falls exponentially with k. □

Lemma 4 directly implies the following theorem.

Theorem 7 (Chain quality). *Let $k, R, \Delta \in \mathbb{N}$ and $\epsilon \in (0,1)$. Let $\mathcal{A}$ be an α-dominated adversary against the protocol π_{SPoS} for some $\alpha > 0$ satisfying $\alpha(1-f)^{\Delta} \geq (1+\epsilon)/2$. Then the probability that $\mathcal{A}$, when executed in a Δ-semisynchronous environment, makes π_{SPoS} violate the chain quality property with parameters k and $\mu = 1/k$ throughout a period of R slots, is no more than $\exp(\ln R - \Omega(k))$.*

4.5 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 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 8. *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_{SPoS}. Assuming that we have a fixed number C of coins (corresponding to 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 particular coin owned by one of the parties. These random variables provide an alternative view of the slot leader election process: the owner of coin i becomes a slot leader for slot t if $c_{t,i} = 1$. Thanks to the "independent aggregation property" (2), sampling the random variables $c_{t,i}$ yields a distribution on slot leaders equivalent to the method used by π_{SPoS}, 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 11 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, since $\mathcal{A}_3$ is static and corrupts at most $(1-\alpha)$-share of the stake. □

Theorems 5, 6, 7 and 8 together give us the following corollary.

Corollary 1. *Let $\mathcal{A}$ be an adaptive adversary against the protocol Π_{SPoS} that corrupts at most $(1-\alpha)$-fraction of stake. Then the bounds on common prefix, chain growth and chain quality given in Theorems 5, 6, 7 are satisfied for $\mathcal{A}$.*

4.6 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 $\mathcal{F}_{\mathsf{INIT}}^{r}$, which permits the adversary to select the random nonce from a family of r independent and uniformly random nonces. Specifically, $\mathcal{F}_{\mathsf{INIT}}^{r}$ 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, $\mathcal{F}_{\mathsf{INIT}}^{r}$ samples a nonce $\eta \xleftarrow{\$} \{0,1\}^{\lambda}$, 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.

Looking ahead, our reason to introduce the resettable functionality $\mathcal{F}_{\mathsf{INIT}}^{r}$ is to capture the limited grinding capabilities of the adversary. A simple application of the union bound shows that this selection of η from among a set of size r uniformly random candidate nonces can inflate the probability of events during the run of π_{SPoS} by a factor no more than r. We record this as a corollary below.

Corollary 2 (Corollary to Theorems 5, 6, 7). *The protocol Π_{SPoS}, with initialization functionality $\mathcal{F}_{\mathsf{INIT}}^{r}$, satisfies the bounds of Theorems 5, 6, 7 with all probabilities scaled by r.*

5 The Full Protocol

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 values τ and r. First, it leaks to the adversary, up to τ slots prior to the end of an epoch, the beacon value for the next epoch. (Looking ahead, we remark that it is essential that the stake distribution used for sampling slot leaders in the next epoch is determined prior to this leakage.) Second, the adversary can *reset* the value returned by the functionality as many as r times. As expected for a beacon, it reports to honest parties the beacon value only once the next 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 $\mathcal{F}_{\mathsf{INIT}}^{r}$. 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.

Naturally, we do not want to assume the availability of a randomness beacon in the final protocol, 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. This hash function is applied to the concatenation of VRF values that are inserted into each block, using values from all blocks up to and including the middle $\approx 8k$ slots of an epoch that lasts approximately $24k$ slots in entirety. (The "quiet" periods before and after this central block of slots that sets the nonce will ensure that the stake distribution, determined at the beginning of the epoch, is stable, and likewise that the nonce is stable before the next epoch begins.) 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 for determining the nonce for 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: towards this, note that the adversary potentially controls a constant fraction of the $\approx 8k$ slots associated with nonce selection, and this allows him to explore an a priori large space of independent random potential nonces (and, ultimately, select one as the next epoch nonce). The size of this space is however upper-bounded by the number of random oracle queries that the adversary can afford during the sequence of $\approx 8k$ slots. To formalize 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).[5] Assuming that the adversary controls t parties, we obtain a bound equal to $\approx 8qtk$.

5.1 The Dynamic Stake Case with a Resettable Leaky 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 8.

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

$\mathcal{F}_{RLB}^{\tau,r}$ incorporates the diffuse functionality from Section 2.2 and is parameterized by the number of initial stakeholders n and their respective stakes $s_1, \ldots, s_n$, 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:

- In the first round, $\mathcal{F}_{RLB}^{\tau,r}$ operates exactly as $\mathcal{F}_{\mathsf{INIT}}$.
- Upon receiving $(\mathsf{genblock\_req}, sid, U_i)$ from stakeholder U_i it operates as functionality $\mathcal{F}_{\mathsf{INIT}}$ on that message.
- Upon receiving $(\mathsf{epochrnd\_req}, sid, U_i, e_j)$ from stakeholder U_i, if $e_j \geq 2$ is the current epoch, $\mathcal{F}_{RLB}^{\tau,r}$ sends $(\mathsf{epochrnd}, sid, \eta_j)$ to U_i.
- For every epoch e_j, at slot $jR - \tau$, $\mathcal{F}_{RLB}^{\tau,r}$ samples the next epoch's nonce $\eta_{j+1} \xleftarrow{\$} \{0,1\}^\lambda$ and leaks it by sending $(\mathsf{epochrnd\_leak}, sid, e_j, \eta_{j+1})$ 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 $(\mathsf{epochrnd\_reset}, sid, \mathcal{A})$ 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 $\eta_{j+1} \xleftarrow{\$} \{0,1\}^\lambda$ and leaks it by sending $(\mathsf{epochrnd\_leak}, sid, \eta_{j+1})$ to $\mathcal{A}$. Finally, $\mathcal{F}_{RLB}^{\tau,r}$ increments Resets and adds η_{j+1} to $\mathbb{P}$.
- Upon receiving $(\mathsf{epochrnd\_set}, sid, \mathcal{A}, \eta)$ 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 $(\mathsf{epochrnd\_leak}, sid, \eta_{j+1})$ to $\mathcal{A}$.

Fig. 8: Functionality $\mathcal{F}_{RLB}^{\tau,r}$.

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. The protocol also adopts an adaptation of the static $\mathsf{maxvalid}_S$ function, defined so that it narrows selection to those chains which share common prefix. Specifically, it adopts the following rule, parametrized by a prefix length k:

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

The protocol π_{DPoS} is described in Figure 9 and functions in the $\mathcal{F}_{RLB}^{\tau,r}$-hybrid model.

[5] 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.

Protocol π_{DPoS}

The protocol π_{DPoS} is run by stakeholders, initially equal to $U_1, \ldots, U_n$ interacting among themselves and with ideal functionalities $\mathcal{F}_{RLB}^{\tau,r}$ (or $\mathcal{F}_{\mathsf{INIT}}$), $\mathcal{F}_{\mathsf{VRF}}, \mathcal{F}_{\mathsf{KES}}, \mathcal{F}_{\mathsf{DSIG}}, \mathsf{H}$ over a sequence of $L = ER$ slots $S = \{sl_1, \ldots, sl_L\}$ consisting of E epochs with R slots each. Define $T_i^j \triangleq 2^{\ell_{\mathsf{VRF}}} \phi_f(\alpha_i^j)$ as the threshold for 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} denotes the output length of $\mathcal{F}_{\mathsf{VRF}}$, f is the active slots coefficient and ϕ_f is the mapping from Definition 1. Then π_{DPoS} proceeds as follows:

1. **Initialization.** This step is the same as Step 1 in π_{SPoS} except that any messages for $\mathcal{F}_{\mathsf{INIT}}$ are sent to $\mathcal{F}_{RLB}^{\tau,r}$ if it is available instead.
2. **Chain Extension.** After initialization, for every slot $sl \in S$, every online stakeholder U_i performs the following steps:
   (a) This step is the same as Step 2a in π_{SPoS}.
   (b) 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 up to $(j-2)R$ as reflected in $\mathcal{C}$ (where τ parameterizes $\mathcal{F}_{RLB}^{\tau,r}$) and sends $(\mathsf{epochrnd\_req}, sid, U_i, e_j)$ to $\mathcal{F}_{RLB}^{\tau,r}$, receiving $(\mathsf{epochrnd}, sid, \eta_j)$ as answer.
   (c) U_i collects all valid chains received via diffusion 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 $B' = (st', d', sl', B_\pi{}', \rho', \sigma_{j'}) \in \mathcal{C}'$ it holds that the stakeholder who created it is in the slot leader set of slot sl' (by parsing $B_\pi{}'$ as (U_s, y', π') for some s, verifying that $\mathcal{F}_{\mathsf{VRF}}$ responds to $(\mathsf{Verify}, sid, \eta_j \,\|\, sl' \,\|\, \mathtt{TEST}, y', \pi', v_s^{\mathrm{vrf}})$ by $(\mathsf{Verified}, sid, \eta_j \,\|\, sl' \,\|\, \mathtt{TEST}, y', \pi', 1)$, and that $y' < T_s^j$ where T_s^j is the threshold of stakeholder U_s for the epoch e_j to which sl' belongs), that $\mathcal{F}_{\mathsf{VRF}}$ responds to $(\mathsf{Verify}, sid, \eta_j \,\|\, sl' \,\|\, \mathtt{NONCE}, \rho'_y, \rho'_\pi, v_s^{\mathrm{vrf}})$ (where $\rho' = (\rho'_y, \rho'_\pi)$) by $(\mathsf{Verified}, sid, \eta_j \,\|\, sl' \,\|\, \mathtt{NONCE}, \rho'_y, \rho'_\pi, 1)$, and that $\mathcal{F}_{\mathsf{KES}}$ responds to $(\mathsf{Verify}, sid, (st', d', sl', B_\pi{}', \rho'), sl', \sigma_{j'}, v_s^{\mathrm{kes}})$ by $(\mathsf{Verified}, sid, (st', d', sl', B_\pi{}', \rho'), sl', 1)$. U_i computes $\mathcal{C}' = \mathsf{maxvalid}(\mathcal{C}, \mathbb{C})$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\mathrm{head}(\mathcal{C}'))$.
   (d) U_i sends $(\mathsf{EvalProve}, sid, \eta_j \,\|\, sl \,\|\, \mathtt{NONCE})$ to $\mathcal{F}_{\mathsf{VRF}}$, obtaining $(\mathsf{Evaluated}, sid, \rho_y, \rho_\pi)$, Afterwards, U_i sends $(\mathsf{EvalProve}, sid, \eta_j \,\|\, sl \,\|\, \mathtt{TEST})$ to $\mathcal{F}_{\mathsf{VRF}}$, receiving $(\mathsf{Evaluated}, sid, y, \pi)$. U_i checks whether it is in the slot leader set of slot sl with respect to the current epoch e_j by checking that $y < T_i^j$. 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, $B_\pi = (U_i, y, \pi)$, $\rho = (\rho_y, \rho_\pi)$ and σ is a signature obtained by sending $(\mathsf{USign}, sid, U_i, (st, d, sl, B_\pi, \rho), sl)$ to $\mathcal{F}_{\mathsf{KES}}$ and receiving $(\mathsf{Signature}, sid, (st, d, sl, B_\pi, \rho), sl, \sigma)$. U_i computes $\mathcal{C}' = \mathcal{C}|B$, sets $\mathcal{C}'$ as the new local chain and sets state $st = H(\mathrm{head}(\mathcal{C}'))$. Finally, if U_i has generated a block in this step, it diffuses $\mathcal{C}'$.
3. **Signing Transactions.** This step is the same as Step 3 in π_{SPoS}.

Fig. 9: Protocol π_{DPoS}

Lazy players. Note that while the protocol π_{DPoS} in Figure 9 is stated for a stakeholder that is permanently online, this requirement can be easily relaxed. Namely, it is sufficient for an honest stakeholder to join at the beginning of each epoch, determine whether she belongs to the slot leader set for any slots within this epoch (using the Eval interface of $\mathcal{F}_{\mathsf{VRF}}$), and then come online and act on those slots while maintaining online presence at least every k slots. For now we only sketch this variant in Appendix H and defer a formal treatment of this property of our protocol to a later version.

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 accommodate the issues that will arise from the movement of stake throughout protocol execution, we recall the notion of stake shift defined in [KRDO17].

Definition 13. *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.*

Finally, the security of π_{DPoS} is stated below. We slightly abuse the notation from previous sections and denote by $\alpha_{\mathcal{H}}$ a *lower bound* on the honest stake ratio throughout the whole execution.

Theorem 9 (Security of π_{DPoS} with access to $\mathcal{F}_{RLB}^{\tau,r}$). *Fix parameters $k, R, \Delta, L \in \mathbb{N}, \epsilon, \sigma \in (0,1)$ and r. Let $R \geq 16k/(1+\epsilon)$ be the epoch length, L the total lifetime of the system, and*

$$\alpha_{\mathcal{H}}(1-f)^{\Delta} \geq (1+\epsilon)/2 + \sigma \ . \tag{12}$$

The protocol π_{DPoS}, with access to $\mathcal{F}_{RLB}^{\tau,r}$, with $\tau \leq 8k/(1+\epsilon)$ satisfies persistence with parameters k and liveness with parameters $u = 8k/(1+\epsilon)$ throughout a period of L slots of Δ-semisynchronous execution with probability $1 - \exp(\ln L + \Delta + \log(r) - \Omega(k))$ 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 domination of honest stake even after the stake shift is taken into account, Corollary 2 still applies for the whole execution over L slots. The critical observation is the fact that the stakeholder distribution is determined in each epoch $j \geq 2$ by the block that has time stamp up to $(j-2)R$. Since $R \geq 16k/(1+\epsilon)$ and $\tau \leq 8k/(1+\epsilon)$ it holds that at least $R - \tau \geq 8k/(1+\epsilon)$ slots will pass before the leaky beacon releases the random value of the next epoch. By applying the chain growth theorem with $s = 8k/(1+\epsilon)$ we obtain that, except with error $\exp\left(-f(1-f)^{\Delta}\alpha s/(20\Delta) + \ln L\Delta + O(1)\right) = \exp\left(-\Omega(k) + \ln L + \ln \Delta\right)$ the chain will grow by $f(1-f)^{\Delta}\alpha_{\mathcal{H}}/4 \cdot u \geq (1+\epsilon)/8 \cdot u = k$ blocks in that period of slots and thus the stakeholder distribution is completely determined prior to the leaky beacon releasing the random value of the next epoch. Based on the above, we observe that any violation of persistence in the execution with parameter k results in the violation of common prefix with parameter k. Applying a union bound, we obtain an error of $\exp(\ln L + \Delta + \log(r) - \Omega(k))$.

We then examine liveness. Consider any transaction that is provided to the honest parties for a sequence of $u = 8k/(1+\epsilon)$ slots it will follow, as before that by chain growth the chain will grow by k blocks. Then, 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. □

Note that while Theorem 9 (and also Corollary 3 below) formulates the bound (12) in terms of the *overall* upper bound on honest stake ratio $\alpha_{\mathcal{H}}$ and *maximum* stake shift σ over any R-slots interval, one could easily prove more fine-grained statements that would only require inequality (12) to hold for each epoch, with respect to the honest stake ratio and stake shift *in this epoch*.

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 10.

Protocol π_{RLB}

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

- Upon receiving $(\mathsf{epochrnd\_req}, sid, U_i, e_j)$ 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 up to $(j-2)R + 16k/(1+\epsilon)$, concatenate the values ρ' into a value v. Compute $\eta_j = \mathsf{H}(\eta_{j-1}||j||v)$ and return $(\mathsf{epochrnd}, sid, \eta_j)$.

Fig. 10: Protocol π_{RLB}.

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 per corrupted party and there are t corrupted parties.

Lemma 5. *Consider the event of violating one of common prefix, chain quality, chain growth in an execution of π_{DPoS} using sub-protocol π_{RLB} in the $\mathcal{F}_{\mathsf{INIT}}$-hybrid model with adversary $\mathcal{A}$ and environment $\mathcal{Z}$ with the same parameter choices as Theorem 9. We construct an adversary $\mathcal{A}'$ so that the corresponding event happens with the same probability in an execution of π_{DPoS} in the $\mathcal{F}_{RLB}^{\tau,r}$-hybrid world with adversary $\mathcal{A}'$ and environment $\mathcal{Z}$ assuming that $r = 8tqk/(1+\epsilon)$.*

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 $\mathcal{F}_{RLB}^{\tau,r}$ oracle. Specifically, a reset query will be triggered whenever $\mathcal{A}$ queries $\mathsf{H}(\cdot)$ with concatenated valid VRF values $\eta_{j-1} \,\|\, j \,\|\, \rho_i \,\|\, \cdots \,\|\, \rho_{i'}$ 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 between $8k/(1+\epsilon)$ and $16k/(1+\epsilon)$. We observe that by chain growth and chain quality at least one honest block in the middle $8k/(1+\epsilon)$ slots of an epoch will be included in the chain of all honest parties and contribute to the calculation of the hash. Finally, when the epoch e_j reaches an end, $\mathcal{A}'$ will issue $(\mathsf{epochrnd\_set}, w)$ query to $\mathcal{F}_{RLB}^{\tau,r}$ 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 that consists of all the blocks of the epoch that contains blocks produced in the first $16k/(1+\epsilon)$ slots of the epoch. Note that the event that the $\eta_{j-1} \,\|\, j \,\|\, \rho_i \,\|\, \cdots \,\|\, \rho_{i'}$ sequence corresponding to that chain in the common prefix was never queried to $\mathsf{H}(\cdot)$ happens with negligible probability. □

Based on the above lemma, it is now easy to revisit Theorem 9, and show that the same result holds for r in the q-bounded model assuming $r = 8tkq/(1+\epsilon)$ and $\tau \leq 8k/(1+\epsilon)$ which permits to set our epoch length R to $24k/(1+\epsilon)$.

Corollary 3 (Security of π_{DPoS} with subprotocol π_{RLB}). *Fix parameters $k, R, \Delta, L \in \mathbb{N}, \epsilon, \sigma \in (0,1)$. Let $R = 24k/(1+\epsilon)$ 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} using subprotocol π_{RLB} in the $\mathcal{F}_{\mathsf{INIT}}$-hybrid model satisfies persistence with parameters k and liveness with parameters $u = 8k/(1+\epsilon)$ throughout a period of L slots of Δ-semisynchronous execution with probability $1 - \exp(\ln L + \Delta - \Omega(k - \log tkq))$ assuming that σ is the maximum stake shift over R slots.*

References

BGM14. Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. *CoRR*, abs/1406.5694, 2014.

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. *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)*, 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.

Com14. The NXT Community. Nxt whitepaper. `https://bravenewcoin.com/assets/Whitepapers/NxtWhitepaper-v122-rev4.pdf`, July 2014.

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

DP07. Yevgeniy Dodis and Prashant Puniya. Feistel networks made public, and applications. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 534–554. Springer, Heidelberg, May 2007.

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. Updated version at `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.

JKK14. Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253. Springer, Heidelberg, December 2014.

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

KN12. Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. `https://peercoin.net/assets/paper/peercoin-paper.pdf`, August 2012.

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 Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.

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.

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

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, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Heidelberg, May 2017.

RMKQ17. Alexander Russell, Cristopher Moore, Aggelos Kiayias, and Saad Quader. Forkable strings are rare. Cryptology ePrint Archive, Report 2017/241, 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 Definitions

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

A.1 Verifiable Random Functions

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

Definition 14 (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*,*

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

A.2 Digital Signatures and $\mathcal{F}_{\mathsf{DSIG}}$

In Figure 11, we present Functionality $\mathcal{F}_{\mathsf{DSIG}}$ as defined in [Can04], where it is also shown that EUF-CMA signature schemes realize $\mathcal{F}_{\mathsf{DSIG}}$. This functionality will be used to model signatures on transactions.

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

$\mathcal{F}_{\mathsf{DSIG}}$ interacts with a signer U_S and stakeholders $U_1, \ldots, U_n$ as follows:

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

  Output $(\mathsf{Verified}, sid, m, f)$ to U_i.

Fig. 11: Functionality $\mathcal{F}_{\mathsf{DSIG}}$.

A.3 Forward Secure Signatures Schemes

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

Definition 15 (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 16 (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 17 (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 16. 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*.*

B Realizing $\mathcal{F}_{\mathsf{KES}}$

We will follow the proof strategy of [Can04] to show that a construction based on a key evolving signature scheme (Definition 15) realizes $\mathcal{F}_{\mathsf{KES}}$. Our construction π_{KES} is based on a key evolving signature scheme $\mathsf{KES} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update})$. The signature protocol π_{KES} is run between a stakeholder U_S and stakeholders $U_1, \ldots, U_n$, proceeding as follows:

- **Key Generation:** When U_S receives an input $(\mathsf{KeyGen}, sid, U_S)$, it runs $\mathsf{Gen}(1^k, T)$, records the signing key $(\mathsf{sid}, \mathsf{KES.sk}_1,)$, sets counter $\mathsf{k_{ctr}} = 1$ and outputs $(\mathsf{VerificationKey}, sid, \mathsf{KES}.vk)$.
- **Signature and Update:** When U_S receives an input $(\mathsf{Sign}, sid, U_S, m, j)$ for a sid for which it has the signing key $(\mathsf{sid}, \mathsf{U_S}, \mathsf{KES.sk}_{\mathsf{k_{ctr}}},)$ and that $\mathsf{k_{ctr}} \geq j \leq T$. Otherwise, it ignores the input. First, U_S performs the following steps until $\mathsf{k_{ctr}} = j$: run $\mathsf{Update}(\mathsf{KES}.sk_{\mathsf{k_{ctr}}})$ to obtain $\mathsf{KES}.sk_{\mathsf{k_{ctr}}+1}$, securely erase $\mathsf{KES}.sk_{\mathsf{k_{ctr}}}$ and increment $\mathsf{k_{ctr}}$. Next, It runs $\mathsf{Sign}_{\mathsf{KES}.sk_{\mathsf{k_{ctr}}}}(m)$ to obtain $\sigma_{\mathsf{k_{ctr}}}$, runs $\mathsf{Update}(\mathsf{KES}.sk_{\mathsf{k_{ctr}}})$ to obtain $\mathsf{KES}.sk_{\mathsf{k_{ctr}}+1}$, securely erases $\mathsf{KES}.sk_{\mathsf{k_{ctr}}}$, outputs $(\mathsf{Signature}, sid, m, \mathsf{k_{ctr}}, \sigma_{\mathsf{k_{ctr}}})$ and increments $\mathsf{k_{ctr}}$.
- **Verification:** When a stakeholder U_i receives an input $(\mathsf{Verify}, sid, m, j, \sigma_j, \mathsf{KES}.vk')$, it outputs $(\mathsf{Verified}, sid, m, j, \mathsf{Verify}_{\mathsf{KES}.vk'}(m, \sigma_j))$.

Theorem 1 *The* π_{KES} *construction presented above, realizes* $\mathcal{F}_{\mathsf{KES}}$ *with erasures assuming* $\mathsf{KES} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update})$ *is a key evolving signature scheme with forward security as per Definition 15 and Definition 17.*

Proof. We follow the approach of [Can04] by showing that an environment $\mathcal{Z}$ that distinguishes an ideal execution with a prescribed simulator $\mathcal{S}$ and $\mathcal{F}_{\mathsf{KES}}$ from a real execution with an adversary $\mathcal{A}$ and π_{KES} can be used to construct a forger for the underlying key evolving signature scheme σ that is successful in the forward security experiment of Definition 16 with non-negligible probability, thus contradicting the forward security guarantee of σ. We argue that since $\mathcal{Z}$ can distinguish the real world execution from the idea world execution for any $\mathcal{Z}$, it will also succeed for an specific simulator $\mathcal{S}$. Simulator $\mathcal{S}$ runs an internal copy of $\mathcal{A}$, proceeding as follows:

- $\mathcal{S}$ forwards any inputs from $\mathcal{Z}$ to $\mathcal{A}$ and any outputs from $\mathcal{A}$ to $\mathcal{Z}$.
- Upon receiving $(\mathsf{KeyGen}, sid, U_S)$ from $\mathcal{F}_{\mathsf{KES}}$, $\mathcal{S}$ runs $\mathsf{Gen}(1^k, T)$, records the signing key $(\mathsf{sid}, \mathsf{U_S}, \mathsf{KES.sk_1},)$, sets counter $\mathsf{k_{ctr}} = 1$ and sends $(\mathsf{VerificationKey}, sid, U_S \mathsf{KES}.vk)$ to $\mathcal{F}_{\mathsf{KES}}$.
- Upon receiving $(\mathsf{Signature}, sid, U_S, j, m)$ from $\mathcal{F}_{\mathsf{KES}}$, $\mathcal{S}$ checks that $\mathsf{k_{ctr}} \geq j \leq T$. Otherwise, it ignores the request. First, $\mathcal{S}$ performs the following steps until $\mathsf{k_{ctr}} = j$: run $\mathsf{Update}(\mathsf{KES}.sk_{\mathsf{k_{ctr}}})$ to obtain $\mathsf{KES}.sk_{\mathsf{k_{ctr}}+1}$, securely erase $\mathsf{KES}.sk_{\mathsf{k_{ctr}}}$ and increment $\mathsf{k_{ctr}}$. Next, $\mathcal{S}$ runs $\mathsf{Sign}_{\mathsf{KES}.sk_{\mathsf{k_{ctr}}}}(m)$ to obtain $\sigma_{\mathsf{k_{ctr}}}$, runs $\mathsf{Update}(\mathsf{KES}.sk_{\mathsf{k_{ctr}}})$ to obtain $\mathsf{KES}.sk_{\mathsf{k_{ctr}}+1}$, securely erases $\mathsf{KES}.sk_{\mathsf{k_{ctr}}}$, sends $(\mathsf{Signature}, sid, U_S, m, \mathsf{k_{ctr}}, \sigma_{\mathsf{k_{ctr}}})$ to $\mathcal{F}_{\mathsf{KES}}$ and increments $\mathsf{k_{ctr}}$.
- Upon receiving $(\mathsf{Verify}, sid, m, j, \sigma, \mathsf{KES}.vk')$ from $\mathcal{F}_{\mathsf{KES}}$, $\mathcal{S}$ sends $(\mathsf{Verified}, sid, m, j, \mathsf{Verify}_{\mathsf{KES}.vk'}(m, \sigma_j))$ to $\mathcal{F}_{\mathsf{KES}}$.
- When $\mathcal{A}$ corrupts a party U', $\mathcal{S}$ corrupts U' in the ideal world. If $U' = U_S$ (*i.e.* the signer), $\mathcal{S}$ reveals the signing key $\mathsf{KES}.sk_{\mathsf{k_{ctr}}}$ and the internal state of Sign (if there's any) as the state of $U_\prime$.

Given an environment $\mathcal{Z}$ that distinguishes an ideal execution with simulator $\mathcal{S}$ and $\mathcal{F}_{\mathsf{KES}}$ from a real execution with $\mathcal{A}$ and π_{KES}, we will construct a forger $F = (F_{\mathsf{cma}}, F_{\mathsf{forge}})$ for the underlying key evolving signature scheme σ. Our forger F will run an internal copy of $\mathcal{Z}$, simulating for $\mathcal{Z}$ the interactions with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{KES}}$ (*i.e.* acting as $\mathcal{S}$ and $\mathcal{F}_{\mathsf{KES}}$ towards $\mathcal{Z}$). Moreover, F will run an internal copy of $\mathcal{A}$ as in the case of $\mathcal{S}$.

Forger F starts execution as F_{cma} in the first phase of the forward security game (as per Definition 16), where it is given a verification key $\mathsf{KES}.vk$ for total number of time periods T and has access to signing oracle. When F is activated, it gives $\mathcal{A}$ the verification key $\mathsf{KES}.vk$ obtained in the forward security experiment. Whenever F has to generate a signature as $\mathcal{S}$, it calls its signing oracle with the given message and current $\mathsf{k_{ctr}}$, which are added the set of queried message and time period pairs CM. The signatures obtained from the oracle are used in the answers of the simulated $\mathcal{S}$ and added the set of signatures $sign(CM)$ corresponding to CM. Whenever the internal $\mathcal{Z}$ activates an uncorrupted party with $(\mathsf{Verify}, sid, m, j, \sigma, \mathsf{KES}.vk')$, F tests whether $(m, j) \notin CM$, $\mathsf{Verify}_{\mathsf{KES}.vk'}(m, \sigma_j)$ and $j < \mathsf{k_{ctr}}$. If this condition is met and the signer is not corrupted, the triple (m, j, σ) can be used by F to succeed in the forward security experiment. F outputs CM, $sign(CM)$ and $\mathsf{k_{ctr}}$ ending the first phase of the forward security game. Next, acting as F_{forge} in the forward security game, it outputs (m, σ_j), succeeding in the experiment. If $\mathcal{A}$ corrupts the signer, F outputs $\perp$ and halts.

Following the same reasoning as in [Can04], we argue that given the correctness property of σ, the internal environment $\mathcal{Z}$ would not issue a query $(\mathsf{Verify}, sid, m, j, \sigma, \mathsf{KES}.vk')$, F such that $(m, j) \notin CM$, $\mathsf{Verify}_{\mathsf{KES}.vk'}(m, \sigma_j)$ and $j < \mathsf{k_{ctr}}$ in the case that the signer is not corrupted. If this query indeed does not happen, $\mathcal{Z}$ would not be able to distinguish between an ideal and a real executions. However, this query happens with non-negligible probability since we assume that $\mathcal{Z}$ does distinguish real from ideal executions. Moreover, notice that the interactions with $\mathcal{F}_{\mathsf{KES}}$ from the point of view of $\mathcal{A}$ and $\mathcal{Z}$ are the same in a real world execution with π_{KES}, guaranteeing that a forgery is obtained. □

C Realizing $\mathcal{F}_{\mathsf{VRF}}$

We provide an implementation of $\mathcal{F}_{\mathsf{VRF}}$ in the random oracle model using two hash functions H, H' with ranges $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and $\langle g \rangle$ respectively with $|\langle g \rangle| = q$. The implementation is based on the 2-Hash-DH verifiable oblivious PRF construction of [JKK14]. Specifically, the public-key is equal to $v = g^k$ and the output is $y = H(m, u)$ where $u = H'(m)^k$, while the proof is set to $\pi = (u, \mathrm{EQDL}(k : \log_{H'(m)}(u) = \log_g(v); m, v))$. The verification of (m, y, π, v) proceeds as follows. First it parses π as (u, π') where π' is a proof of equality of discrete logarithms. It verifies

$y = H(m, u)$ as well as the proof π' and returns 1 if and only if both tests pass. The proof notation $\text{EQDL}(k : \log_{H'(m)}(u) = \log_g(v); m, v)$ stands for the string (c, s) where $c = H(m, v, g^r, H'(m)^r)$, $s = r + kc \bmod q$, while the verification of (c, s) on context m, v is performed by checking the equality

$$c = H(m, v, g^s v^{-c}, H'(m)^s u^{-c}) .$$

Theorem 2 *The 2Hash-DH construction presented above, realizes $\mathcal{F}_{\mathsf{VRF}}$ in the random oracle model assuming the CDH.*

Proof. We describe a simulator $\mathcal{S}$ that controls the random oracles H, H' and operates in the following manner.

1. Upon receiving a message $(\mathsf{KeyGen}, sid, U_i)$ from $\mathcal{F}_{\mathsf{VRF}}$, a new value $v = g^k$ is selected for a random k and $\mathcal{S}$ inserts (U_i, v) in its internal registry of keys; in case the key exists already, $\mathcal{S}$ returns FAIL and terminates. It returns to $\mathcal{F}_{\mathsf{VRF}}$ the message $(\mathsf{VerificationKey}, sid, U_i, v)$.
2. Upon receiving a message $(\mathsf{EvalProve}, sid, U_i, m)$ from $\mathcal{F}_{\mathsf{VRF}}$, this is matched to the verification key v of U_i and is checked whether m has been queried before. In such case, the value u that corresponds to m in the table for v is recovered. In case m was not queried before, it is checked whether $H'(m)$ is defined. In such case the entry (base, m, t) is recovered, the value u is set to v^t and the triple (v, m, u) is stored for future reference. In case the value $H'(m)$ is undefined $\mathcal{S}$ selects t at random, stores (base, m, t) and sets $H'(m) = g^t$. Subsequently random c, s values are selected by $\mathcal{S}$; the pair $((m, v, g^s v^{-c}, H'(m)^s u^{-c}), c)$ is inserted to the table of the random oracle H. In case this is not feasible (because that would make the table inconsistent), $\mathcal{S}$ outputs FAIL and terminates. Finally, π is set to $(u, (c, s))$ and the message $(\mathsf{Eval}, sid, m, \pi)$ is returned to $\mathcal{F}_{\mathsf{VRF}}$.
3. Upon receiving $(\mathsf{Verify}, sid, m, y, \pi, v')$ from $\mathcal{F}_{\mathsf{VRF}}$, parse π as $(u', (c, s))$ and verify the proof (c, s) as a proof of equality of discrete logarithms, $\log_g(v') = \log_{H'(m)}(u')$, to obtain a bit b. Now observe that (base, m, t) must be recorded, and set $b' = ((u')^{1/t} = v') \wedge (H(m, u) = y)$. If $b \neq b'$ output FAIL and terminate. In any other case, return $(\mathsf{Verified}, sid, m, y, \pi, v')$ to $\mathcal{F}_{\mathsf{VRF}}$.
4. Upon receiving a query m for the random oracle H', select t at random, store (base, m, t) and return g^t.
5. Upon receiving a query for the random oracle H of the form (m, u), and the value (base, m, t) is stored previously, $\mathcal{S}$ performs the following. First, if $v = u^{1/t}$ is not registered as a public-key it registers $(\mathsf{KeyGen}, sid, v)$ with $\mathcal{F}_{\mathsf{VRF}}$. Then it submits $(\mathsf{Eval}, sid, u^{1/t}, m)$ to the $\mathcal{F}_{\mathsf{VRF}}$. If $\mathcal{F}_{\mathsf{VRF}}$ ignores the request $\mathcal{S}$ terminates with FAIL. Else it obtains the response y which is set as the random oracle output to the query (m, u). In case (base, m, t) is not stored, then perform the step that corresponds to the query $H'(m)$ above and repeat the current step.
   Other queries to H are handled by returning random elements of $\{0, 1\}^{\ell_{\mathsf{VRF}}}$.

We observe that unless $\mathcal{S}$ outputs FAIL the simulation of protocol 2Hash-DH is perfect. We next argue that the probability to output FAIL is negligible. $\mathcal{S}$ outputs FAIL in the case that $\mathcal{F}_{\mathsf{VRF}}$ ignores a request $(\mathsf{Eval}, sid, u^{1/t}, m)$. This means that the key $v = u^{1/t}$ is registered with an honest party that has not evaluated m. It follows that the event an adversary that produces such u can be turned to solver for the CDH assumption. The other cases where $\mathcal{S}$ produces FAIL, specifically, step 1 and step 3 can be easily seen to be negligible probability events. □

D 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.

D.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.

D.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 $\mathsf{maxvalid}$ 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], $\mathsf{maxvalid}$ 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 $\Delta \geq 2$.

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 10 (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 11 ([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 10 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 From Executions to Forks

We define the useful notion of an *execution tree* that captures the structure formed by all chains that are observed by any honest party during an execution.

Definition 18 (Execution tree). *Consider an execution $\mathcal{E}$ of the hybrid experiment. The* execution tree *for this execution is a directed, rooted tree $T_{\mathcal{E}} = (V, E)$ with a labeling $\ell : V \to \mathbb{N}_0$ that is constructed during the execution as follows:*

(i) At the beginning, $V = \{r\}$, $E = \emptyset$ and $\ell(r) = 0$.
(ii) Every chain C' that is input to maxvalid *as a part of $\mathbb{C}$ in Step 2b or created as a new local chain in Step 2c of π_{SPoS} run by any honest party is immediately processed block-by-block from the genesis block to* $\text{head}(C')$*. For every block $B = (st', d', sl', B_\pi{}', \sigma_{j'})$ processed for the first time:*
  - *a new vertex v_B is added to V;*
  - *a new edge (v_{B^-}, v_B) is added to E where B^- is the unique block such that $H(B^-) = st'$;*
  - *the labeling ℓ is extended by setting $\ell(v_B) = sl'$.*

We now observe that as desired, the execution tree of every execution is among the forks for the characteristic string of that execution. Note that technically, this conclusion is conditioned on no collisions in the random oracle outputs. For the sake of improved readability, we neglect the possibility of such collisions in our further considerations.

Lemma 6. *For any execution $\mathcal{E}$ of the hybrid experiment we have $T_{\mathcal{E}} \vdash_{\Delta} w_{\mathcal{E}}$, unless a collision in the responses of the random oracle occurs.*

Proof (sketch). Given an execution $\mathcal{E}$ and the resulting execution tree $T_{\mathcal{E}}$, we need to prove that it satisfies the properties (i)–(v) of Definition 6 with respect to $w_{\mathcal{E}}$. Property (i) follows directly from the definition of $T_{\mathcal{E}}$, while (ii) follows from the order in which vertices in $T_{\mathcal{E}}$ are created: every edge is directed from an older vertex to a newer one and the root is the first vertex created. Property (iii) is satisfied due to the requirement in Definition 2 that the sequence of slots in a valid blockchain is strictly increasing (otherwise, the chain is rejected by π_{SPoS} as invalid). To see property (iv), note that by π_{SPoS}, every uniquely honest slot leader will create a block and this is immediately processed, resulting in a vertex with the respective label in $T_{\mathcal{E}}$. Finally, property (v) is satisfied as every uniquely honest slot leader will be aware of any other such slot leader's block created at least Δ slots ago due to the guarantees provided by DDiffuse, and will hence extend a chain that is at least as long as the one containing this block. Note that several of our arguments above assume no random oracle collisions. □

Given Lemma 6, we can later focus on investigating the properties of the distribution $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$. Roughly speaking, if we prove that a characteristic string sampled from $\mathcal{D}^f_{\mathcal{Z},\mathcal{A}}$, with overwhelming probability, does not allow for *any* "harmful" forks for it, then this also implies that a random execution with overwhelming probability results in a "harmless" execution tree.

G Further Details on Forks, Forkability and Divergence

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

Definition 19 (Height and tine 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 20 (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 12 ([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 $\Pr[w \text{ is forkable}] = 2^{-\Omega(k)}$. 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 13 ([KRDO17]). *Let $w \in \{0,1\}^*$ with $\operatorname{div}_0(w) \geq k$. Then there is a forkable substring $\breve{w}$ of w with $|\breve{w}| \geq k$.*

An immediate conclusion of Theorems 12 and Theorem 13 is the following bound on the probability that a synchronous characteristic string drawn from the binomial distribution has large divergence.

Theorem 14 (Restatement of Theorem 3). *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 used in [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 3 relying on the results of [RMKQ17].

Proof (of Theorem 3). It follows from Theorem 13 that if $\operatorname{div}_0(w) \geq k$, there is a forkable substring $\breve{w}$ of length at least k. Thus

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

According to Theorem 12 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

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

as desired. □

H A Lazy Variant of $\pi_{\mathbf{DPoS}}$

This section sketches how the protocol π_{DPoS} can be easily modified to significantly weaken the requirement of all honest parties being online all the time. In the modified protocol $\pi^{\mathsf{lazy}}_{\mathrm{DPoS}}$ given in Figure 12, every stakeholder comes online only at the beginning of each epoch, determining for which of the slots in that epoch he belongs into the slot leader set. Consequently, the stakeholder can only come online in those slots and create blocks as usually. Additionally, the stakeholder also makes sure that it is not offline for k or more slots.

Note that this requires a slightly different mechanism of distributing existing chains, as the lazy stakeholder needs a way to obtain a copy of the current chain the moment he comes online. In theory, this can be achieved by letting all stakeholders diffuse their current chain even if they haven't added a block to it; in practice this would be handled by a request mechanism where a stakeholder can ask for chains from other nodes upon coming online.

Protocol $\pi^{\mathsf{lazy}}_{\mathrm{DPoS}}$

The protocol $\pi^{\mathsf{lazy}}_{\mathrm{DPoS}}$ is run in the same setting as π_{DPoS}, with the same definition of T_i^j. It proceeds as follows:

1. **Initialization.** This step is the same as Step 1 in π_{DPoS}. Additionally, U_i initializes an empty set $\mathcal{L}$.
2. **Chain Extension.** After initialization, every stakeholder U_i acts as described below. If none of the conditions below occurs, U_i can go online and offline arbitrarily, with the requirement that it is never offline for k or more consecutive slots.
   - **upon a new epoch:** U_i performs Step 2b of π_{DPoS}. Afterwards, for every slot $\hat{sl}$ of the upcoming epoch e_j, U_i sends $(\mathsf{Eval}, sid, \eta_j \parallel \hat{sl} \parallel \mathtt{TEST})$ to $\mathcal{F}_{\mathsf{VRF}}$, receiving $(\mathsf{Evaluated}, sid, y)$. U_i checks whether it is in the slot leader set of slot $\hat{sl}$ with respect to the current epoch e_j by checking that $y < T_i^j$. If yes, it adds $\hat{sl}$ to a set $\mathcal{L}$.
   - **upon a slot in $\mathcal{L}$:** U_i performs the Steps 2c and 2d of π_{DPoS}.
3. **Upon being online:** Whenever U_i is online, it performs Steps 2a and 3 of π_{DPoS}.

Fig. 12: Protocol $\pi^{\mathsf{lazy}}_{\mathrm{DPoS}}$.