Proof-of-Stake Blockchain Protocols with Near-Optimal Throughput

Matthias Fitzi*, Peter Gaži*, Aggelos Kiayias*†, Alexander Russell*‡
*IOHK Research
†University of Edinburgh
‡University of Connecticut

***Abstract*—One of the most significant challenges in the design of blockchain protocols is increasing their transaction-processing throughput. In this work we put forth for the first time a formal execution model that enables to express transaction throughput while supporting formal security arguments regarding persistence and liveness. We then present a protocol in the proof-of-stake setting achieving near-optimal throughput under adaptive active corruption of any minority of the stake.**

I. Introduction

Permissionless blockchain protocols inspired by Bitcoin [28] exhibit an inherent speed-security trade-off that depends on the latency of the communication network. This was first formally illustrated for the proof-of-work (PoW) setting in [18] by expressing how block production per round of message-passing must be relatively small for security theorems to be meaningful. This point was further refined in [29] where block production per unit of time was restricted with respect to $1/\Delta$, where Δ is the network delay, while for arbitrary delays a generic attack was formulated. A conclusion of these results, that also carries over to Nakamoto-style proof-of-stake (PoS) protocols, is that in order to achieve safety, blockchain protocols must be operated at a relatively slow pace with respect to network latency.

This issue severely impacts the throughput of conventional protocols since long periods of silence between blocks result in an underutilization of the available network bandwidth.

Later efforts to improve throughput of permissionless protocols focused mostly on PoW-based ledgers, and lead to a number of advances in protocol design that adopted one of two possible approaches. The first one delegates block creation to a temporary leader or committee, thus providing better throughput at least in optimistic environments – exemplified by works such as Bitcoin-NG [14], Hybrid Consensus [30], Thunderella [32], or Mir-BFT[39]. The second approach redefines the underlying blockchain protocol by deviating from the original single-chain structure – exemplified by "block-tree" or "DAG" protocol proposals such as GHOST [35], Spectre [34], Phantom [37], GraphChain [7], Tangle [33], Conflux [24], and Swirlds [4].

Despite the above exciting development, there are still clear gaps in the current state of the art. First, we lack a detailed formal execution model capable of capturing the theoretical throughput of blockchain protocols. Second, most of the above work focuses on the PoW paradigm, with the promising alternative approach of PoS being left behind. This paper addresses both these deficiencies.

Our results. We propose a new formal model that captures a concept of throughput for distributed ledger protocols. This is achieved by a novel, fine-grained network-diffusion functionality that reflects network communication with greater precision compared to previous work, tailored towards capturing protocol throughput. Based on this model, we propose a PoS protocol provably achieving both security and (near) optimal throughput against any adaptive active adversary controlling a minority of stake in the system.

Network and throughput modeling. Our network functionality abstracts the uplink, transit queue as well as the downlink of each single party in the network. Whenever a message is transmitted, it is placed via a party's uplink into the transit queue for all recipients. The uplink capacity of each party is by definition one message per unit of time (we use the term "slot"), and messages are restricted to a fixed packet length and as such may be insufficient to carry all the transactions that are available in a party's pool of pending transactions. The transit queue for each recipient can be arbitrarily manipulated by the adversary by placing messages into a party's downlink and making them become available to the party, subject to an upper bound on the number of messages that may be accessed from the downlink per slot. Note that we do not put any restriction on maximum delivery time into the model, rather condition our security statements on such a bound.

Using this network abstraction, we introduce the following notion for block-based distributed-ledger protocol throughput: it is the cardinality of a largest set of disjoint (with respect to the transactions they carry) honest blocks added into the *stable* ledger divided by the number of slots – during any slot interval of some minimal length within the protocol execution. A protocol is defined to achieve (near) throughput-optimality if it can be parameterized such that this ratio converges to the fraction of stake controlled by the honest participants.[1]

Throughput-optimal PoS protocol. As a basis for our construction, we consider a Nakamoto-style PoS protocol with suboptimal throughput, such as [20], [6], [12], [2]. In a nutshell, we execute m instances of it in parallel, producing m chains while any given transaction is eligible to be added

[1]For simplicity, we will drop the term "near" in the sequel.

to exactly one of the chains, e.g., determined by the hash of the transaction.

The staking lottery (deciding eligibility to append new blocks) is evaluated independently for each of these chains, but for each chain is based on the joint stake distribution comprising all the stake in the system. At any given moment, the contents of the m chains in the view of an honest party can be organized into a single ledger state by ordering the transactions according to the index of the block and then the rank of the blockchain it belongs to (from 1 to m).

So far, the sketched extension gives the same security guarantees as the underlying single-chain protocol, and turns out to already achieve optimal throughput in presence of a fail-stop adversary who can adaptively shut down parties but cannot make them misbehave arbitrarily. However, achieving throughput optimality against fully adversarial behavior poses additional challenges: the adversary might for example apply a variant of the selfish-mining attack [15] to unbalance the proportion of honest blocks in the ledger, and only create empty blocks to harm throughput. To mitigate such problems, we borrow an idea from the PoW space and adapt it to the PoS setting: specifically we adapt the Conflux block-inclusion rule from [24], based on the inclusive-blockchain idea from [23]: all blocks pruned by the chain-selection rule are re-included to the ledger, ensuring that no honest blocks ever get lost.

Note the following difference to standard DAG protocols from the literature: those protocols improve throughput by increasing the block growth while all blocks are added to the same block graph and forks are frequent. Our protocol maintains separate graphs at low growth and forks are as rare as in the underlying slow protocol, while throughput optimality is achieved by parallelism. This separation of graphs gives the protocol more structure and makes it easier to analyze.

We make our construction explicit by applying the above technique to the PoS protocol Ouroboros Praos [12]. This protocol provides security against fully adaptive corruptions, which is inherited by our construction.

We formally prove both the security and the throughput guarantees of the construction. Note that the latter would be impossible (for *any* protocol!) without our formal throughput model, which we see as an important independent contribution that we hope to prove useful for analyzing security of other protocols in the future.

Analysis vs. simulation. It is often debated whether throughput behavior should be proven experimentally or analytically. We consider both alternatives justified with their advantages over the other.

The experimental approach makes use of real networks but must typically be based on simplified assumptions over the environment, and in particular, does not capture worst-case adversarial behavior.

In contrast, the analytical approach must be based on a simplified (idealized) network model but gives provable guarantees under worst-case adversarial behavior in this model. In fact, this paper analytically proves that asymptotic throughput optimality is achievable by PoS protocols – a property that would be hard to demonstrate experimentally.

Related work. The hybrid approach that decouples (fast) transaction processing from (slow) blockchain protocol maintenance is intrinsic in the proposal by Eyal et al. [14] (Bitcoin-NG), however this protocol cannot withstand adaptive corruption. Generalizing the chain structure of [28] to more general graphs was first considered by Sompolinsky et al. [36] and Lerner [22]. The former generalization (GHOST) still extracts a single chain to represent the ledger but the blocks of the complete block tree (including all forks) remain to contribute to this selection process to preserve the PoW of the blocks outside the main chain. The latter suggests to organize the blocks in a directed acyclic graph (DAG) where a single block references multiple previous blocks. The first DAG protocols involving some type of formal analysis were proposed by Boyen et al. [7], Sompolinsky et al. [34], [37], and Bentov et al. [5]. However, these proposals only involve qualitative arguments without giving a full formal model for arguing about throughput. Other examples of DAG protocols are described by Popov [33] (IOTA) and Baird [4] (Hashgraph) — both not thoroughly analyzed in the Byzantine permissionless setting. In [24], Li et al. proposed Conflux, the first DAG protocol that directly allows for a detailed analysis. As their protocol is an extension of the GHOST rule, maximal respective throughput is limited by the conditions under which the GHOST protocol can safely operate.

Our work first appeared as part of [16]. While combining multiple chains in parallel has been examined before in [25], [21], [17], and concurrently in [41], none of them demonstrated throughput optimality. Similarly, the PoS protocol in [39] delegates block creation to a PBFT committee wherein the participants propose blocks in parallel; achieving throughput superiority over the other committee-based protocols as demonstrated by experimental results. Note that involving a committee inherently limits robustness to less than one third of actively corrupted stake – while our solution tolerates any minority of actively corrupted stake.

Concurrently to our work, Bagaria et al. [3] published a throughput-optimal protocol using the input-endorsers [20] or fruits [31] technique originating in [18] – but restricted to the PoW setting and analyzed in a simpler network model. They achieve throughput-optimality under active corruption while our initial protocol, though actively secure, only achieved throughput-optimality under fail-stop corruption. Throughput-optimality under active corruption was added to our protocol later.

In the realm of PoS, Algorand [26] improves over the latency of eventual-consensus protocols, but does not address throughput. Thunderella [32] provides improved throughput under optimistic conditions (at least 3/4 of honest stake and a designated party is honest), but downgrades to single-chain throughput otherwise, e.g. after a single adaptive corruption of the designated party.

II. Preliminaries

Basic Notation. For $n \in \mathbb{N}$ we use the notation $[n]$ to refer to the set $\{1, \ldots, n\}$. For brevity, we often write $\{x_i\}_{i=1}^{n}$ and $(x_i)_{i=1}^{n}$ to denote the set $\{x_1, \ldots, x_n\}$ and the tuple $(x_1, \ldots, x_n)$, respectively.

Execution Model. We divide time into discrete units called *slots*. Parties are equipped with (roughly) synchronized clocks that indicate the current slot: we assume that any clock drift is subsumed in the slot length. Each slot is indexed by an integer $t \in \{1, 2, \ldots\}$. For an interval of slots $I = [t_1, t_2]$ we denote its length as $|I| \triangleq t_2 - t_1 + 1$.

We consider a UC-style [8] execution of a protocol Π, involving an environment $\mathcal{Z}$, a number of parties P_i, functionalities that these parties can access while running the protocol (such as the functionality $\mathcal{F}_{\mathsf{diff}}$ used for communication), and an adversary $\mathcal{A}$. All these entities are interactive algorithms. The environment controls the execution by activating parties via inputs it provides to them. The parties, unless corrupted, respond to such activations by following the protocol Π and invoking the available functionalities as needed. We denote by $\mathsf{Exec}(\Pi, \mathcal{A}, \mathcal{Z})$ the random variable containing the full transcript of an execution of the protocol Π with adversary $\mathcal{A}$ and environment $\mathcal{Z}$.

Adaptive Byzantine Corruptions. The adversary influences the protocol execution by interacting with the available functionalities, and by corrupting parties. To corrupt a party, the adversary has to first ask the environment $\mathcal{Z}$ for permission. If the corruption is approved by $\mathcal{Z}$ (via a special message from $\mathcal{Z}$ to $\mathcal{A}$), the adversary corrupts P_i immediately. A corrupted party P_i will relinquish its entire state to $\mathcal{A}$; from this point on, the adversary will be activated in place of the party P_i. We call these *Byzantine* or *active* corruptions.

Semi-synchronous Communication. Our model allows for semi-synchronous (sometimes called *non-lock-step synchronous*) communication among honest parties, where messages are guaranteed to be delivered within Δ slots. This is formally captured by the functionality $\mathcal{F}_{\mathsf{diff}}$ that also models bandwidth-restricted communication, we describe it in detail in Section III as one of our main contributions. We do not make restriction on maximum delivery time an explicit part of $\mathcal{F}_{\mathsf{diff}}$, we rather condition our security statements on maximum observed delays. Note that the value Δ is unknown to the protocol itself.

Blockchains and Ledgers. A *blockchain* (or a *chain*) (denoted e.g. C) is a sequence of blocks where each block is connected to the previous one by containing its hash. The first block of a chain C is called its *genesis block* and denoted by $\mathcal{G}$. The last block of C is denoted by $\mathsf{head}(\mathsf{C})$. By $\mathsf{C}^{\lceil k}$ we denote C where the last k blocks have been removed. We denote the extension of a chain C by a block B as $\mathsf{C}' = \mathsf{C} \,\|\, B$. For simplicity, we write $B \in \mathsf{C}$ iff C contains block B. By $\mathsf{len}(\mathsf{C})$ we denote the number of blocks in C.

A *ledger* (denoted in bold-face, e.g., $\mathbf{L}$) is a mechanism for maintaining a sequence of transactions, often stored in the form of a blockchain. We slightly abuse the language by letting $\mathbf{L}$ (without further qualifiers) interchangeably refer to the algorithms used to maintain the sequence, and all the views of the participants of the state of these algorithms when being executed. For example, the (existing) ledger Bitcoin consists of the set of all transactions that ever took place in the Bitcoin network, the current UTXO set, as well as the local views of all the participants.

In contrast, we call a *ledger state* a concrete transaction sequence $\mathrm{tx}_1, \mathrm{tx}_2, \ldots$ stored in the *stable* part of a ledger $\mathbf{L}$, typically as viewed by a particular party. Hence, in every blockchain-based ledger $\mathbf{L}$, every fixed chain C defines a concrete ledger state by applying the interpretation rules given as a part of the description of $\mathbf{L}$ (e.g., the ledger state is obtained from the blockchain by dropping the last k blocks and serializing the valid transactions in the remaining blocks). We denote by $\mathbf{L}^{\mathsf{P}}[t]$ the ledger state of a ledger $\mathbf{L}$ as viewed by a party P at the end of a time slot t. We say that a party P reports a transaction tx as *stable* during time slot t iff $\mathrm{tx} \in \mathbf{L}^{\mathsf{P}}[t]$. By $\tilde{\mathbf{L}}^{\mathsf{P}}[t]$ we denote the ledger that additionally contains any transactions that are valid but still pending according to the view of party P. For any t, P, it holds that $\mathbf{L}^{\mathsf{P}}[t] \preceq \tilde{\mathbf{L}}^{\mathsf{P}}[t]$ where $\preceq$ denotes the prefix relation.

Finally, we introduce a specific class of transaction-ledger protocols that have the following structure: transactions are grouped into blocks, and the stable ledger state is determined by the protocol at any time t by selecting which of the blocks seen so far are *stable*, ordering them according to some protocol-specific rule, interpreting them as a sequence of the transactions they contain, and sequentially removing all transactions from this sequence that would not be valid with respect to the remaining transactions that are preceding them. We call such transaction ledger protocols *block-based* and remark that almost all protocols considered in this paper are block-based. Note that for a block-based ledger protocol $\mathbf{L}$, it is well-defined to treat a ledger state $\mathbf{L}^{\mathsf{P}}[t]$ as the sequence of the stable blocks that form it, allowing for notational shorthands such as for example $|\mathbf{L}^{\mathsf{P}}[t]|$ denoting the number of stable blocks for P at time t, and $B \in \mathbf{L}^{\mathsf{P}}[t]$ to indicate that B is a stable block at time t for a party P.

Recall the definitions of persistence and liveness of a robust public transaction ledger adapted from the most recent version of [18]:

Definition 1. *A protocol Π implements a* robust transaction ledger (with respect to parameter u) *iff it organizes the ledger as a sequence of transactions by satisfying the following properties:*

Persistence. *For any two honest parties $\mathsf{P}_1, \mathsf{P}_2$ and time slots $t_1 \leq t_2$, it holds that $\mathbf{L}^{\mathsf{P}_1}[t_1] \preceq \tilde{\mathbf{L}}^{\mathsf{P}_2}[t_2]$.*

Liveness. *If all honest parties in the system attempt to include a transaction then, at any slot t after u slots (called the liveness parameter), any honest party P, if queried, will report $\mathrm{tx} \in \mathbf{L}^{\mathsf{P}}[t]$.*

When a ledger is implemented by a single blockchain, each party P at time t holds a single chain C as its state. If we want to make P and t explicit, we denote this chain as $\mathsf{C}^{\mathsf{P}}[t]$. For

such blockchain-based ledger protocols, robustness is known to be implied by the following blockchain properties:

Common Prefix (CP)**.** A blockchain protocol achieves *common-prefix with parameter* $k \in \mathbb{N}$ if for any pair of honest parties $\mathsf{P}_0, \mathsf{P}_1$ and time slots t_0, t_1 such that $t_0 \leq t_1$, it holds that $\mathsf{C}^{\mathsf{P}_0}[t_0]^{\lceil k} \preceq \mathsf{C}^{\mathsf{P}_1}[t_1]$.

Chain Growth (CG)**.** A blockchain protocol achieves *chain growth with parameters* $\tau \in \mathbb{R}$ *and* $s \in \mathbb{N}$ if for any honest party P, for slots t_1, t_2 such that $t_1 + s \leq t_2 \leq t$ the chain $\mathsf{C}^{\mathsf{P}}[t]$ contains at least $\tau \cdot s$ blocks from the time interval $(t_1, t_2]$.

Chain Quality (CQ)**.** A blockchain protocol achieves *chain quality with parameters* $\mu \in \mathbb{R}$ *and* $\ell \in \mathbb{N}$ if for any honest party P and any time slot t, it holds that among any ℓ consecutive blocks in $\mathsf{C}^{\mathsf{P}}[t]$ the ratio of adversarial blocks is at most $1 - \mu$.

III. Modeling Bandwidth Limitations

Our communication model is based on the model from [12], but extends it to capture bandwidth limitations of the protocol participants and the efficiency of the protocol in terms of throughput. The network model is formally captured by the "diffuse" functionality $\mathcal{F}_{\mathsf{diff}}$ given in Figure 1.

We start by fixing a block size b and defining a *slot* to be the time interval in which a party can "push" a block of this size to the network, given its bandwidth (we assume the bandwidth limitations to be uniform for all parties).

We now allow parties to diffuse messages to all other parties at the maximum rate of one block per slot. If a party attempts to diffuse more than one block in a given slot, these blocks are queued locally and sent one by one. After being sent, the messages are delayed arbitrarily (and independently for each recipient) by the adversary, and upon delivery they are pushed into the recipient's inbox queue, from which they can be fetched by the recipient at the maximum speed of $\mu \in \mathbb{N}$ blocks per slot. Here μ is the *bandwidth-asymmetry parameter* of our model. Unless stated otherwise, we consider the symmetric case $\mu = 1$.

The adversary is allowed to submit arbitrary additional messages for diffusion. These messages do not suffer from initial queuing (we assume the adversary has an unlimited uplink), but will queue on the recipients' side. Importantly, also the adversarial messages are sent to all participating parties – this reflects the nature of peer-to-peer gossip protocols where even adversarial messages, once entering the system, are propagated to all parties. In line with the UC communication model we also allow the adversary to send direct messages to parties, but as parties can see that these messages were not received via $\mathcal{F}_{\mathsf{diff}}$, they drop them as no such messages are expected by the protocol. This again reflects the behavior of peer-to-peer gossip networks underlying permissionless protocols.

Of course, we need to put some restrictions on the adversarial capability of delaying messages, as blockchain protocols are known to be insecure in settings with unbounded delays, cf. [29]. In any fixed execution transcript T (that implicitly specifies a protocol, an adversary, and an environment) and a message M with corresponding message identifier mid diffused in T via the functionality $\mathcal{F}_{\mathsf{diff}}$, we denote by $\mathsf{delay}_T(\mathsf{mid})$ the amount of slots that passed between M being submitted for diffusion and being fetched from the inbox by all its honest recipients (or the end of the execution, whichever comes first), decreased by 1. Furthermore, we let $\mathsf{maxdelay}(T) \triangleq \max_{\mathsf{mid} \in T} \mathsf{delay}_T(\mathsf{mid})$, where the maximum is taken over all message identifiers mid diffused by honest parties in T.

Fig. 1. **Functionality** $\mathcal{F}_{\mathsf{diff}}$.

The functionality $\mathcal{F}_{\mathsf{diff}}$ is parameterized by $\mu \in \mathbb{N}$. It keeps rounds, executing one round per slot. $\mathcal{F}_{\mathsf{diff}}$ interacts with the environment $\mathcal{Z}$, stakeholders $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and an adversary $\mathcal{A}$, working as follows for each round:

1) $\mathcal{F}_{\mathsf{diff}}$ maintains three lists of messages for each party P_i that participates: outbox_i, $\mathsf{delayed}_i$ and inbox_i, and one message list $\mathsf{outbox}_{\mathcal{A}}$ for the adversary; initially all empty. Each outbox_i and inbox_i for $i \neq \mathcal{A}$ operate as queues, keeping the messages in the order in which they were added.
2) In each round, for each party P_i at most one message is marked as *pending*, and keeps this status until the round's end: If outbox_i is non-empty at the beginning of the round then the first message it contains is marked as pending, otherwise it is the first message P_i submits for diffusion (see Item 3) in this round.
3) Whenever activated, a party P_i is allowed to submit for diffusion a sequence of messages of size up to b bits, each. Each message is assigned a unique message identifier mid. These messages are appended to outbox_i in order of the sequence.
4) When the adversary $\mathcal{A}$ is activated, it is allowed to:
   - Read all lists $\mathsf{delayed}_i$ and inbox_i and all pending messages, with corresponding mid values.
   - Create arbitrary messages and submit them for diffusion, these messages are put into $\mathsf{outbox}_{\mathcal{A}}$. This list is not a queue, all messages in it are immediately considered to be pending.
   - For any $j \in \{1, \ldots, n, \mathcal{A}\}$ and any pending message mid still in outbox_j, $\mathcal{A}$ can remove the corresponding message M from outbox_j and put it into $\mathsf{delayed}_i$ for all $i \in [n]$.
   - For any party P_i, $\mathcal{A}$ can remove any message from the list $\mathsf{delayed}_i$ and push it to the queue inbox_i.
5) At the end of each round, the functionality ensures that every pending message still present in outbox_j for $j \in \{1, \ldots, n, \mathcal{A}\}$ is removed from outbox_j and pushed into $\mathsf{delayed}_i$ for all $i \in [n]$.
6) A party is once per round (if activated) allowed to pop at most μ oldest messages from inbox_i. These messages are given to P_i and removed from inbox_i.

Definition 2 ($\mathsf{MaxDelay}_\Delta$)**.** *For a protocol* Π*, an adversary* $\mathcal{A}$*, and an environment* $\mathcal{Z}$*, we define the event* $\mathsf{MaxDelay}_\Delta(\Pi, \mathcal{A}, \mathcal{Z})$ *(or just* $\mathsf{MaxDelay}_\Delta$ *for brevity) to be the event that* $\mathsf{maxdelay}(\mathsf{Exec}(\Pi, \mathcal{A}, \mathcal{Z})) \leq \Delta$.

Looking ahead, our security analysis will provide meaningful guarantees in adversarial settings where the probability of the event $\neg\mathsf{MaxDelay}_\Delta$ remains negligible for some Δ (which may not be necessarily known explicitly to the participants). Note that this for example excludes adversaries that would try to attack the protocol by a DoS type of attack clogging the network, as these network-level attacks are outside of the

scope of our model.

However, note that since our model bounds the downlink throughput of the parties, some protocols may cause message delays even when operating in a non-adversarial environment. Definition 3 captures this; to present it we first need to formalize such non-adversarial executions. For this, we denote by $\mathcal{A}_\perp$ the *dummy adversary* that does not corrupt any parties, and does not delay or inject any messages, i.e., the adversary pushes pending outbox messages directly into the inbox_i of party P_i for all i. Now, a protocol is κ-bounded if its operations do not cause any delay beyond κ when assuming no (adversarial) propagation delays:

Definition 3 (κ-Boundedness)**.** *A protocol Π is (κ, ε)-*bounded *if for the adversary $\mathcal{A}_\perp$ and for any environment $\mathcal{Z}$ we have $\Pr[\neg\mathsf{MaxDelay}_\kappa(\Pi, \mathcal{A}, \mathcal{Z})] \leq \varepsilon$.*

Finally, we describe how to quantify the throughput of a block-based transaction ledger protocol in this model. Our definition is akin to the standard definition of the chain growth property, applied to a ledger instead of a single blockchain, and restricting our counting to non-intersecting honest blocks.

Definition 4 (Productive blocks)**.** *A set of stable blocks generated by a block-based ledger protocol is called* productive *iff they were all generated by honest parties (at the time of block generation) and all blocks' included transaction sets are pairwise disjoint. The set is called* productive with respect to slot interval I *iff its blocks were generated within the slot interval I.*

Definition 5 (Throughput (TP))**.** *A block-based transaction-ledger protocol achieves (θ, L_0)-*throughput *for parameters $\theta \in [0, 1]$ and $L_0 \in \mathbb{N}$ iff for any honest party P, for any $L \geq L_0$, and for any slots t_1, t_2 such that $t_1 + L \leq t_2 \leq t$, the ledger $\mathbf{L}^{\mathsf{P}}[t]$ contains a set of at least $\theta \cdot (t_2 - t_1)$ productive blocks with respect to the slot interval $(t_1, t_2]$.*

Definition 5 captures "productive" growth of the stable ledger as a fraction of the theoretical maximum growth of μ blocks per slot. With an adversarial corruption of an $(1 - \alpha)$-fraction of all stake, the best achievable throughput is for θ to approach $\mu \cdot \alpha$, and this is what we refer to as throughput optimality.

As an illustration, we use our definitional framework to give a quantitative assessment of several natural throughput-scaling proposals in the following section.

IV. Assessing Some Throughput-Scaling Proposals

As a warm-up, we use our definitional framework to give a quantitative assessment of several natural scaling proposals. We mostly consider the PoW setting here as throughput optimization has hardly been addressed in the PoS space yet, and the respective ideas similarly apply to the PoS case. The purpose is to familiarize the reader with our model and illustrate its expressiveness, hence for simplicity this section only provides informal claims without proofs and focuses on throughput in the non-adversarial setting (with $\mathcal{A}_\perp$).

a) Plain Bitcoin: First, let us consider Bitcoin with blocks of size b, i.e., such that a block can be submitted for diffusion by a party in a single slot. In a typical Bitcoin execution without adversarial interference, a successful miner of a block B spends a single slot diffusing B, this block is then delivered within Δ slots to the future miner of the next block (along with all other miners), and finally this miner spends d slots mining for the successor of B. During these $\Delta + d$ slots, the "available bandwidth" is not used productively, contributing to the rather unsatisfactory throughput of $\approx 1/(1 + \Delta + d)$ with $1 \ll \Delta \ll d$. If a block is considered stable after k blocks are mined on top of it, the expected confirmation time (latency) of this protocol is $\approx k \cdot (1 + \Delta + d)$.

b) Nakamoto-style PoS Protocols: The above analysis applies also to eventual-consensus ("Nakamoto-style") proof-of-stake protocols such as [20], [6], [12], [2], [1] with one difference: the quantity d does not capture the expected delay caused by mining, but rather by the leader-selection algorithm that replaces it in these PoS protocols. Nonetheless, given the probabilistic nature of both Δ and d, the protocol's resilience against incidental forks still requires a parameterization of the leader-selection algorithm resulting in $\Delta \ll d$ and hence limiting the achievable throughput.

c) Larger Blocks: Consider the Bitcoin protocol with each "logical" block consisting of s network-level blocks, i.e., a block that can be submitted for diffusion in s slots (we neglect the details of the mechanism for splitting the block into parts). Similarly to above, we arrive at an expected throughput of $\approx s/(s+\Delta+d)$ and latency $\approx k\cdot(s+\Delta+d)$. However, note that $s + \Delta \ll d$ is still necessary to maintain a low forking rate even in the non-adversarial setting, limiting the achievable throughput.

d) Shorter Block Intervals: As above, the effect of shortening block interval $s + \Delta + d$ (by lowering the mining difficulty and hence d) is limited due to the condition $s + \Delta \ll d$ necessary to prevent accidental forks.

e) Conflux: Bitcoin (or another PoW or PoS Nakamoto-style protocol) is run whereby "pruned" blocks are eventually re-included in the ledger. Still, the block interval cannot be substantially reduced for the same argument as in the case of "shorter block intervals" above.

f) Bitcoin-NG: This proposal exploits a part of the available throughput between the diffusion of ("key") blocks to produce and diffuse so-called microblocks created by the miner of the last key block. Therefore, out of the $1+\Delta+d$ slots that represent the time interval between two consecutive key blocks, only the bandwidth of the last Δ slots are wasted, as the microblocks produced in these slots will not be received in time by the miner of the next key block. This results in a throughput of $\approx (1 + d)/(1 + \Delta + d)$, with a latency of $\approx k\cdot(1+\Delta+d)$. Hence, increasing d improves throughput but hurts latency. Also, increasing d leads to greater centralization (due to less frequent key blocks and hence more variance in rewards) and weaker adaptive security guarantees (each leader is a target for corruption—such as a DoS attack—until the next key block is selected).

V. Our Protocol

For concreteness, we implement our protocol on top of Ouroboros Praos [12] as our underlying PoS protocol. Still, note that similar protocols (e.g. [20], [6], [2]) could be easily used instead to obtain similar results. We first provide a self-contained overview of Ouroboros Praos (Section V-A) and its inclusive extension (Section V-B) as a reference for our exposition. The descriptions in these sections are not fully formal to avoid duplicity, as the detailed outline of our new protocol (Section V-C) will capture all of the mentioned building blocks in a formal manner.

A. Ouroboros Praos

The protocol, denoted Π_{Praos} here, operates (and was analyzed) in the semi-synchronous model with fully adaptive corruptions, described in Section II.

It operates as follows: In each slot, each of the parties determines whether she qualifies as a so-called *slot leader* for this slot. This is done by locally evaluating a verifiable random function (VRF, [13]) using the secret key associated with their stake, and providing as inputs to the VRF both the slot index and so-called *epoch randomness* (we will discuss shortly where this randomness comes from). If the VRF output is below a certain threshold that is proportional to the party's stake, then the party is an eligible slot leader and she creates, signs, and broadcasts a block for that slot (containing transactions that move stake among stakeholders). She also includes into the block she creates the above VRF output and its proof, to certify her eligibility to act as a slot leader. Additionally, she also inserts another, independent VRF value with a proof, these will be used for determining next epoch randomness (see below). Note that the event of a particular party becoming a slot leader is independent for two different slots and two different parties, leading to some slots with no, or several, slot leaders. Parties participating in the protocol are collecting such valid blocks and always update their current state to reflect the longest chain they have seen so far that did not fork from their previous state by too many blocks into the past (as captured by procedure $\mathsf{selectchain}$ in Fig. 3, called $\mathsf{maxvalid}$ in [12]).

Multiple slots are combined into *epochs*, each of which contains $R \in \mathbb{N}$ slots. The epochs are indexed by $j \in \mathbb{N}$. During epoch j, slot-leader election is based on the stake distribution recorded in the ledger up to the last block of epoch $j-2$. Additionally, the *epoch randomness* for epoch j is derived as a hash of those additional VRF-values that were included into blocks in the first two thirds of epoch $j-1$ for this purpose by the respective slot leaders.

In greater detail, the threshold T_i^j for a party P_i, which her VRF output must fall below for P_i to become a slot leader in a fixed slot of epoch j is

$$T_i^j \triangleq 2^{\ell_{\mathsf{VRF}}} \phi_f(\alpha_i^j) \tag{1}$$

where $\alpha_i^j \in [0,1]$ is the relative stake of stakeholder P_i in the stake distribution $\mathbb{S}_j$, ℓ_{VRF} denotes the output length of the VRF (formally, of the functionality $\mathcal{F}_{\mathsf{vrf}}$), $f \in (0,1)$ is the so-called *active slots coefficient*, and ϕ_f is the mapping

$$\phi_f(\alpha) \triangleq 1 - (1-f)^\alpha \,. \tag{2}$$

Finally, the protocol uses *key-evolving signatures* (KES, formally modeled by the functionality $\mathcal{F}_{\mathsf{kes}}$) for block signing, and in each slot the honest parties update their private key, contributing to their resilience to adaptive corruptions.

To accommodate the described protocol operation, every block in the Ouroboros Praos blockchain has the format $B = (st, d, sl, \pi_B, \rho, \sigma)$: Here st is the hash of the previous block; d denotes the transaction-carrying data; and sl is the index of the slot this block belongs to. π_B is a triple (P_i, y, π) where (y, π) is the VRF output-proof pair certifying that P_i is an eligible slot leader for slot sl; $\rho = (y_\rho, \pi_\rho)$ is an independent VRF output-proof pair used for randomness generation. Finally, σ is a key-evolving signature of the block by its creator.

We now note the security statement about Π_{Praos} from [12], expressed in our model. Here *honest stake ratio* denotes the stake (as recorded in the current view of the ledger) controlled by uncorrupted parties as a proportion of the total stake. Moreover, *stake shift* between two slots $t_1 < t_2$ denotes the statistical distance of the stake distributions corresponding to t_1 and t_2. Since the stake distribution used for sampling slot leaders in Π_{Praos} is at most 2 epochs old, in all statements below it is necessary to consider $\alpha_{\mathcal{H}} - \nu$ as the lower bound on effective honest stake ratio, see [20], [12], [40] for a more detailed discussion of stake shift relevance.

Theorem 1 (Security of Ouroboros Praos [12])**.** *Fix parameters $k, R, \Delta, L \in \mathbb{N}$; $\xi_1, \nu \in (0,1)$. Let $R = 24k/f$ be the epoch length, let L denote the total lifetime of the system. Let $\mathcal{A}$ be a Byzantine adversary, let $\alpha_{\mathcal{H}}$ be a lower bound on the honest stake ratio throughout the execution. Let ν and q be upper bounds on the maximum stake shift over $2R$ slots, and the number of adversarial random oracle queries, respectively. If $(\alpha_{\mathcal{H}} - \nu)(1-f)^\Delta \geq (1+\xi_1)/2$ then Π_{Praos} implements a robust transaction ledger against $\mathcal{A}$ with liveness parameter $u_{\mathsf{Praos}} = 8k/f$ throughout a period of L slots except with probability $\varepsilon_{\mathsf{Praos}} = \exp(\ln L + \Delta - \Omega(k - \log qk)) + \Pr[\neg \mathsf{MaxDelay}_\Delta]$.*

B. An Inclusive Variant

As discussed in Section I, an active adversary can invalidate a substantial fraction of honest blocks in a single-chain protocol by a variant of selfish mining [15].

To prevent this, Conflux [24] applies an inclusive-chain rule [23] to re-include all blocks that got pruned off the main chain. Note that this remedy alone does not achieve optimal throughput, see Section IV for a discussion.

To achieve throughput optimality of our final parallel-chains protocol under *active* corruption, we first adapt Ouroboros Praos to apply the Conflux inclusion rule (in a slightly modified way) before moving to its parallel execution.

The inclusion is performed by adding to each block, on top of the link used to append a block to an existing chain, also

additional distinguished references to pick up off-chain blocks. To disambiguate, we call the former *chain links* and the latter *inclusion references*. For chain selection, inclusion references are ignored.

> **The inclusion rule:** When a slot leader creates a block to extend a chain C, they also add inclusion references to all tips of valid chains forking from C that are not yet referenced from previous blocks in C.

Given a stable chain $\mathsf{C} = B_1, B_2, \ldots, B_\ell$ of blocks B_i, the final sequence of blocks in the ledger is determined by including the additionally referenced blocks outside C as follows: each B_i is directly preceded by all blocks in $\mathsf{past}(B_i) \setminus \mathsf{past}(B_{i-1})$ sorted topologically (with tie-breaking by block hash), where following [34], [38], [24], $\mathsf{past}(B)$ denotes all blocks related to B in the transitive closure of the union of both reference relations (chain links and inclusion references).

Note that, in contrast to original Conflux, our chain selection is by the longest-chain rule instead of GHOST [36], which adapts more readily to Ouroboros Praos. It is easy to see that the stabilization of block B_i in the main chain implies the stabilization of the full history implied by B_i.

As a side remark, the inclusion of off-chain blocks must be managed properly to prevent protocol-level DoS attacks: to this end, our protocol only stores (and later includes) the first received block per slot and slot leader.

C. The Full Construction: Parallel Chains

We denote our protocol Π^m_{pc}, where pc stands for "parallel chains". As mentioned, on a high level the protocol Π^m_{pc} consists of a parallel execution of m copies of a slightly adapted Inclusive Ouroboros Praos protocol such that these copies together maintain a joint ledger (describing a joint distribution of stake). We sometimes omit m from the notation and simply write Π_{pc}.

The protocol Π_{pc} is spelled out in detail in Figures 2 and 3; we now focus on describing the differences from a single-chain execution of Inclusive Ouroboros Praos. The protocol assumes the availability of functionalities $\mathcal{F}_{\mathsf{init}}$, $\mathcal{F}_{\mathsf{diff}}$, $\mathcal{F}_{\mathsf{vrf}}$, $\mathcal{F}_{\mathsf{sig}}$, $\mathcal{F}_{\mathsf{kes}}$, $\mathcal{F}_{\mathsf{ro}}$ as described in [12] and surveyed in Appendix B for completeness.

We treat the protocol in the symmetric-bandwidth model with $\mu = 1$ (cf. Section III). We remark that it can be trivially generalized to a different bandwidth profile (with better downlink bandwidth $\mu > 1$) by involving an increased number of chains $m_\mu \triangleq \mu \cdot m$ to again achieve optimal throughput as long as the block producers would not oversaturate their uplinks.

In Π_{pc} we extend the block format of Π_{Praos} to also contain the index of a chain it belongs to, denoted $c \in [m]$; and a list $\mathcal{I}$ of references to blocks included according to the rule from Section V-B. Formally, every block in Π_{pc} has the format $B = (c, st, \mathcal{I}, d, sl, \pi_B, \rho, \sigma)$. We sometimes call a block c-*block* to refer to its chain index. The c-th chain only contains c-blocks (except the genesis block) and is referred to as the c-chain. Since each block B contains its slot number, we denote it by $\mathsf{slot}(B)$, and $\mathsf{C}^{\lfloor t}$ denotes the chain C truncated to only contain blocks up to slot t. Overloading this notation, $\mathsf{C}^{\lfloor j}$ denotes the chain C truncated up to the end of epoch j, and similar filtering can also be applied to block sets.

Formally, let $E_{\mathsf{st}}, E_{\mathcal{I}}$ be the binary relations on blocks such

Fig. 2. **Protocol** Π^m_{pc}, part 1.

The protocol Π_{pc} is run by stakeholders, initially equal to $\mathsf{P}_1, \ldots, \mathsf{P}_n$, interacting among themselves and with ideal functionalities $\mathcal{F}_{\mathsf{init}}$, $\mathcal{F}_{\mathsf{diff}}$, $\mathcal{F}_{\mathsf{vrf}}$, $\mathcal{F}_{\mathsf{sig}}$, $\mathcal{F}_{\mathsf{kes}}$, $\mathcal{F}_{\mathsf{ro}}$ over a sequence of $L = ER$ slots $S = (sl_1, \ldots, sl_L)$ consisting of E epochs with R slots each. Let $T_i^j \triangleq 2^{\ell_{\mathsf{VRF}}} \phi_f(\alpha_i^j)$. Then Π_{pc} proceeds as follows for each stakeholder P_i:

1) **Initialization.**
   a) P_i sends $(\mathsf{KeyGen}, sid, \mathsf{P}_i)$ to $\mathcal{F}_{\mathsf{vrf}}$, $\mathcal{F}_{\mathsf{kes}}$ and $\mathcal{F}_{\mathsf{sig}}$; receiving $(\mathsf{VerificationKey}, sid, v_i)$ for $v_i \in \{v_i^{\mathrm{vrf}}, v_i^{\mathrm{kes}}, v_i^{\mathrm{sig}}\}$, respectively. If this is the first round, P_i sends $(\mathsf{ver\_keys}, sid, \mathsf{P}_i, v_i^{\mathrm{vrf}}, v_i^{\mathrm{kes}}, v_i^{\mathrm{sig}})$ to $\mathcal{F}_{\mathsf{init}}$ to claim stake from the genesis block. In any case, it terminates the round by returning $(\mathsf{P}_i, v_i^{\mathrm{vrf}}, v_i^{\mathrm{kes}}, v_i^{\mathrm{sig}})$ to $\mathcal{Z}$.
   b) In the next round, P_i sends $(\mathsf{genblock\_req}, sid, \mathsf{P}_i)$ to $\mathcal{F}_{\mathsf{init}}$, receiving $(\mathsf{genblock}, sid, \mathbb{S}_0, \eta)$. If P_i is initialized in the first round, it sets the local blockchains $\mathcal{C} \triangleq (\mathsf{C}_c)_{c=1}^m$ to $\mathsf{C}_c := \mathcal{G} \triangleq (\mathbb{S}_0, \eta)$ and block sets $\mathcal{B} \triangleq (\mathbb{B}_c)_{c=1}^m$ to $\mathbb{B}_c = \{\mathcal{G}\}$ for each $c \in [m]$, otherwise it receives the local blockchains $\mathcal{C} = (\mathsf{C}_c)_{c=1}^m$ and sets $\mathcal{B} = (\mathbb{B}_c)_{c=1}^m$ from the environment.

After initialization, in each slot $sl^* \in S$ (of epoch e_{j^*}), P_i performs the following:

2) **Epoch Update.** If a new epoch e_{j^*} with $j^* \geq 2$ has started, P_i computes $\mathbb{S}_{j^*}$ and η_{j^*} as follows:
   a) $\mathbb{S}_{j^*}$ is the stake distribution recorded in the state $\mathsf{GetValidTX}\left(\mathcal{C}^{\lfloor j^*-2}, \mathcal{B}^{\lfloor j^*-2}\right)$ where the inputs are the currently held chains and block sets $(\mathcal{C}, \mathcal{B})$ truncated up to the last slot of epoch $j^* - 2$.
   b) To compute η_{j^*}, collect the blocks $B = (c, st, \mathcal{I}, d, sl, \pi_B, \rho, \sigma) \in \mathsf{C}_c$ belonging to epoch e_{j^*-1} up to the slot with timestamp $(j^* - 2)R + 2R/3$ in any of the currently held c-chains C_c for $c \in [m]$, concatenate the values y_ρ (from each ρ) into a value v in some fixed predetermined order, and let $\eta_{j^*} = \mathsf{H}(\eta_{j^*-1} \,\|\, j^* \,\|\, v)$.
3) **Chains Update.** For all $c \in [m]$, P_i performs the following steps:
   a) P_i processes every new c-block $B = (c, st, \mathcal{I}, d, sl, \pi_B, \rho, \sigma)$ with $\pi_B = (\mathsf{P}_s, y, \pi)$, $\rho = (y_\rho, \pi_\rho)$, and sl belonging to some epoch e_j; received via diffusion as follows: The block is added to $\mathbb{B}_c$ if all the following conditions are satisfied, otherwise it is dropped:
      (i) $sl \leq sl^*$, $y < T_s^j$, and there is no block with the same (P_s, sl) in $\mathbb{B}_c$,
      (ii) $\mathcal{F}_{\mathsf{vrf}}$ replies $(\mathsf{Verified}, sid, \eta_j \,\|\, c \,\|\, sl \,\|\, \mathtt{TEST}, y, \pi, 1)$; to $(\mathsf{Verify}, sid, \eta_j \,\|\, c \,\|\, sl \,\|\, \mathtt{TEST}, y, \pi, v_s^{\mathrm{vrf}})$
      (iii) $\mathcal{F}_{\mathsf{vrf}}$ replies $(\mathsf{Verified}, sid, \eta_j \,\|\, c \,\|\, sl \,\|\, \mathtt{NONCE}, y_\rho, \pi_\rho, 1)$ to $(\mathsf{Verify}, sid, \eta_j \,\|\, c \,\|\, sl \,\|\, \mathtt{NONCE}, y_\rho, \pi_\rho, v_s^{\mathrm{vrf}})$;
      (iv) $\mathcal{F}_{\mathsf{kes}}$ replies $(\mathsf{Verified}, sid, (c, st, d, sl, \pi_B, \rho), sl, 1)$ to $(\mathsf{Verify}, sid, (c, st, \mathcal{I}, d, sl, \pi_B, \rho), sl, \sigma, v_s^{\mathrm{kes}})$.
   b) P_i determines $\mathbb{V}_c$ as the set of all *connected* blocks in $\mathbb{B}_c$ and collects all c-chains that can be constructed (respecting the previous-block hashes st) from the blocks in $\mathbb{V}_c$ into a set $\mathbb{C}_c$.
   c) P_i computes $\mathsf{C}_c := \mathsf{selectchain}(\mathsf{C}_c, \mathbb{C}_c)$, and updates $\mathcal{C} = (\mathsf{C}_c)_{c=1}^m$.

Fig. 3. **Protocol** Π_{pc}^m, part 2.

4) **Chains Extension.** P_i receives from the environment the transaction data $d^* \in \{0,1\}^*$ to be inserted into the ledger. For all $c \in [m]$, P_i performs the following steps:
   a) Send $(\mathsf{EvalProve}, sid, \eta_j \,\|\, c \,\|\, sl^* \,\|\, \mathtt{NONCE})$ to $\mathcal{F}_{\mathsf{vrf}}$, get $(\mathsf{Evaluated}, sid, y_\rho^c, \pi_\rho^c)$.
   b) Send $(\mathsf{EvalProve}, sid, \eta_j \,\|\, c \,\|\, sl^* \,\|\, \mathtt{TEST})$ to $\mathcal{F}_{\mathsf{vrf}}$, get $(\mathsf{Evaluated}, sid, y^c, \pi^c)$.
   c) P_i checks whether $y^c < T_i^j$. If yes, it chooses a maximal sequence d' of c-transactions in d^* that can be appended to $\mathsf{GetValidTX}\,(\mathcal{C}, \mathcal{B})$ without invalidating it and fit into a block, and attempts to include d' into C_c as follows: It generates a new block $B = (c, st_c, \mathcal{I}, d', sl^*, \pi_B, \rho, \sigma)$ where $st_c = H(\mathsf{head}(\mathsf{C}_c))$, $\mathcal{I}$ is a list of hash references to all leaf blocks (referenced neither by chain links nor by inclusion references) in $\mathbb{V}_c$ that are not in $\mathsf{past}(\mathsf{head}(\mathsf{C}_c))$, $\pi_B = (\mathsf{P}_i, y^c, \pi^c)$, $\rho = (y_\rho^c, \pi_\rho^c)$ and σ is a signature obtained by sending $(\mathsf{USign}, sid, \mathsf{P}_i, (c, st_c, \mathcal{I}, d', sl^*, \pi_B, \rho), sl^*)$ to $\mathcal{F}_{\mathsf{kes}}$ and receiving the repsonse $(\mathsf{Signature}, sid, (c, st_c, \mathcal{I}, d', sl^*, \pi_B, \rho), sl^*, \sigma)$. P_i computes $\mathsf{C}_c := \mathsf{C}_c \,\|\, B$, sets C_c as the new local c-chain and diffuses B.
5) **Signing Transactions.** Upon receiving $(\mathsf{sign\_tx}, sid', tx)$ from the environment, P_i sends $(\mathsf{Sign}, sid, \mathsf{P}_i, tx)$ to $\mathcal{F}_{\mathsf{sig}}$, receiving $(\mathsf{Signature}, sid, tx, \sigma)$. Then, P_i sends $(\mathsf{signed\_tx}, sid', tx, \sigma)$ back to the environment.

Procedure $\mathsf{selectchain}(\mathsf{C}, \mathbb{C})$**:**
1) Drop all chains C' from $\mathbb{C}$ that fork from C more than k blocks (i.e., more than k blocks of C would be discarded if C' was adopted).
2) Return the longest of the remaining chains. If multiple such chains remain, return either C if this is one of them, or return the one that is listed first in $\mathbb{C}$.

Procedure $\mathsf{GetValidTX}(\mathsf{C}_1, \ldots, \mathsf{C}_m, \mathbb{B}_1, \ldots, \mathbb{B}_m)$**:**
1) Take all blocks in chains $\mathsf{C}_1, \ldots, \mathsf{C}_m$ and order these blocks in an increasing order according to their slot index sl, breaking ties using the chain number c, obtaining a sequence $B_1, \ldots, B_\ell$.
2) For each $i \in [\ell]$, prepend block B_i in this sequence by all blocks from $\bigcup_{i \in [m]} \mathbb{B}_i \cap (\mathsf{past}(B_i) \setminus \mathsf{past}(B_{i-1}))$ sorted topologically, breaking ties by block hash.
3) Take all transactions from the resulting sequence of blocks, in the order as they appear. Remove all transactions that are invalid with respect to the ledger state formed by all the preceding transactions.

that $B_1 E_{\mathsf{st}} B_2$ if and only if B_2 contains $H(B_1)$ as its value st; and $B_1 E_{\mathcal{I}} B_2$ if and only if B_2 contains $H(B_1)$ in its list of references $\mathcal{I}$. Let $\mathsf{past}(B) \triangleq \{B' : B'(E_{\mathsf{st}} \cup E_{\mathcal{I}})^+ B\}$, where R^+ denotes the transitive closure of a relation R. At any point during the execution, a party considers a block B she has seen as *connected* if the party has already seen all the blocks in $\mathsf{past}(B)$, while the genesis block $\mathcal{G}$ is always connected. Accordingly, a received block is only considered once it is connected.

To prevent concurrent block production across the m chains from including the same transactions, we partition the transaction set along the lines of "sharding" [11], [10]: We assume a fixed function $\mathsf{chain}(\mathrm{tx})$ that for each transaction tx returns a chain index $c \in [m]$ of the only chain in which tx can be inserted. We refer to a tx such that $\mathsf{chain}(\mathrm{tx}) = c$ as a c-transaction, and assume that $\mathsf{chain}(\cdot)$ is reasonably balanced (consider for example $\mathsf{chain}(\mathrm{tx}) = H(\mathrm{tx}) \mod m$).

Each party P_i executing Π_{pc} starts by requesting its verification keys for a VRF, a KES, and an ordinary signature scheme, just like in the single-chain protocol. She claims its stake in the genesis block $\mathcal{G}$ (if she participates from the first round) and initializes all m chains to either consist of $\mathcal{G}$ only (if this is the first round), or adopts chains received from the environment.

The sequences of transactions stored in the m parallel executions are merged using the procedure $\mathsf{GetValidTX}$ given in Figure 3. Any party P_i obtains the ledger state $\mathbf{L}^{\mathsf{P}_i}[t]$ of the stable transactions at time t as follows: given its current view of chains $\mathcal{C} = (\mathsf{C}_c)_{c=1}^m$ and respective blocks $\mathcal{B} = (\mathbb{B}_c)_{c=1}^m$, she computes t_{stable} as the maximum slot up to which all m chains are stable, i.e., the minimum slot index of a k-blocks-deep block in any of the chains (this only depends on $\mathcal{C}$, not $\mathcal{B}$). $\mathbf{L}^{\mathsf{P}_i}[t]$ is then obtained by merging the chains' (inclusive) block sequences together with the included blocks up to this slot using $\mathsf{GetValidTX}$. Formally,

$$t_{\mathsf{stable}} \triangleq \min_{c \in [m]} \mathsf{slot}\left(\mathsf{head}\left(\mathsf{C}_c^{\lceil k}\right)\right)$$

and

$$\mathbf{L}^{\mathsf{P}_i}[t] \triangleq \mathsf{GetValidTX}\left(\mathcal{C}^{\lfloor t_{\mathsf{stable}}}, \mathcal{B}^{\lfloor t_{\mathsf{stable}}}\right) ,$$

where the truncation is performed on every element of the sets $\mathcal{C}, \mathcal{B}$ individually.

The stake distribution $\mathbb{S}_j$ and epoch randomness η_j for determining slot leaders in epoch j are derived in conceptually the same way as in the single-chain case. Namely, the stake distribution used is taken from the end of epoch $j-2$, by looking at all chains and included blocks up to that point and merging them using $\mathsf{GetValidTX}$. The randomness η_j is obtained by hashing together all the VRF values y_ρ included in all m chains in all blocks in epoch $j-1$ up to its slot $2R/3$. Slot leadership is determined independently for each party, each slot, *and each chain*, via the same VRF-based threshold mechanism as in Ouroboros Praos, and verified in the same way. Eligible slot leaders (for a particular chain C_c) simply create a block for this chain, and include pending c-transactions that are consistent with the current view of the chains, as well as all block references $\mathcal{I}$ according to the inclusive rule. Validity of a chain received from the network is determined as in the single-chain case, except that the transactions themselves are not validated at this point. Their validity can only be determined based on the state of the other chains.

VI. Analysis

In this section we formally prove the robustness and throughput guarantees of Π_{pc}.

A. Robustness

We first establish the security of Π_{pc}^m by proving that it implements a robust transaction ledger against Byzantine

adversaries with the same liveness as Π_{Praos} and the error probability scaled by the number of chains m.

Theorem 2 (Robustness)**.** *Fix parameters $k, \Delta \in \mathbb{N}$; $\xi_1 \in (0,1)$; let f denote the active slot coefficient in Π^m_{pc}, let $R = 24k/f$ be the epoch length, let L denote the total lifetime of the system. Let $\mathcal{A} \in \mathbb{A}_{\mathsf{b}}$ be a Byzantine-corruption adversary, let $\alpha_{\mathcal{H}}$ be a lower bound on the honest stake ratio throughout the execution. Let ν and q be upper bounds on the maximum stake shift over $2R$ slots, and the number of adversarial random oracle queries, respectively. Let u_{Praos} and $\varepsilon_{\mathsf{Praos}}$ be as in Theorem 1. If $(\alpha_{\mathcal{H}} - \nu)(1-f)^{\Delta} \geq (1+\xi_1)/2$ then the protocol Π^m_{pc} implements a robust transaction ledger with liveness parameter u_{Praos} against $\mathcal{A}$ except with error probability at most $\varepsilon_{\mathsf{rob}} = m \cdot \varepsilon_{\mathsf{Praos}} + \Pr[\neg \mathsf{MaxDelay}_{\Delta}]$.*

Proof sketch. Observe that each of the m chains produced during the execution of Π^m_{pc} can be seen as an outcome of an independent execution of Π_{Praos}, except that the leader selection in epoch j is based on:

- the stake distribution $\mathbb{S}_j$ taken from the state of *all* m chains at the end of epoch $j-2$;
- the epoch randomness η_j derived from VRF-values y_ρ collected from appropriate blocks of *all* m chains.

In particular, note that the chain construction and chain selection (and hence also the consensus itself) is completely unaffected by the "inclusive" changes described in Section V-B.

An analysis of the robustness argument for single-chain Π_{Praos} given in [12] shows that the argument is not violated by the above changes, the error probability merely needs to be adjusted by an additional factor m to account for a union bound over the failure probability for each of the chains. Since the analysis in [12] is performed in a model where all message delays are bounded by Δ, an additional additive term $\Pr[\neg \mathsf{MaxDelay}_{\Delta}]$ accounts for the occurrence of longer delays in our model.

More concretely, the robustness of the single-chain Π_{Praos} is derived in [12] from the simpler chain-specific properties of common prefix (CP), chain growth (CG), and chain quality (CQ), as defined in Section II. First, CP, CG and CQ are established for a single-epoch execution of the protocol with a static-corruption adversary. As these arguments assume a fixed stake distribution and perfect epoch randomness, they can also be directly applied to a single-epoch execution of each of the m chains of Π^m_{pc} under these assumptions. Second, it is shown that any adaptive adversary is dominated by a particular static adversary, hence extending the single-epoch guarantees to adaptive adversaries as well: this argument can also be applied to Π_{pc} without modification. Finally, the analysis is extended to multiple epochs by analyzing the subprotocol for updating the stake distribution and randomness used for leader sampling. This analysis relies on the single-epoch bounds on CP, CG and CQ violations obtained above. Since in Π^m_{pc} this subprotocol depends on all m chains, we need to assume that single-epoch CP, CG and CQ are maintained on each of them, leading to the additional factor m in the final security bound. □

B. κ-Boundedness

We now investigate the delays caused by non-adversarial message queuing for our construction. Namely, we show that Π^m_{pc} is κ-bounded except with probability negligible in κ, as long as the number of chains m does not exceed a particular threshold. Intuitively, exceeding it would cause a block creation rate in the system to go too close to (or exceed) 1 and hence overload the recipients' bandwidth – even in absence of an adversary.

Theorem 3. *[κ-Boundedness] Let $\kappa \in \mathbb{N}$ and $\xi_2 \in (0, 1/2)$, let f denote the active slot coefficient in Π^m_{pc} and let $m \leq (1-\xi_2)(1-f)/f$. The probability that, during an L-round execution of Π^m_{pc} with $\mathcal{A}_{\perp}$, there is a block that incurs a delay of at least κ rounds, is bounded by*

$$\varepsilon_{\kappa} \leq 2L \cdot \exp\left(-\frac{\xi_2^2(\xi_2\kappa - 3)}{9}\right) .$$

Hence, the protocol Π^m_{pc} executed over a period of L slots is $(\kappa, \varepsilon_{\kappa})$-bounded.

Towards proving Theorem 3, we first establish a general queuing lemma that will be used in our argument.

Lemma 1. *Assume an L-slot execution of $m \geq 1$ queues each with service rate 1 per slot wherein, during any r slots, at most r overall messages get added to the queues. Then there is no slot where, at its very beginning, all queues together contain $r+1$ or more messages.*

Proof. Consider the single-queue case — whereas the multi-queue case directly follows as the overall service is at least as large as in the single-queue case. For the sake of contradiction, consider the first slot t where the queue contains at least $r+1$ messages at its very beginning. Consider the oldest one of these $r+1$ messages and consider the first slot t' where this message was present at the beginning of the slot. It must hold that $t' \geq t-r+1$ as, otherwise, there were $r+1$ messages in the queue at the beginning of slot $t' < t$ where t is assumed to be the first such slot. This implies that all $r+1$ messages must have arrived within the r slots $t-r, \ldots, t-1$, in contradiction to the assumption. □

Now we apply Lemma 1 to upper-bound the number of messages waiting in the outbox and inbox queues at any point during an execution of Π_{pc} with $\mathcal{A}_{\perp}$, this is done respectively in Lemmas 2 and 3.

Lemma 2. *Let $\kappa \in \mathbb{N}$ and $\xi_2 \in (0, 1/2)$, let f denote the active slot coefficient in Π^m_{pc}. For any*

$$m \leq (1-\xi_2)(1-f)/f \ , \tag{3}$$

the probability that during an L-slot execution of Π^m_{pc} with $\mathcal{A}_{\perp}$ there is a slot where all parties' outbox queues together contain at least $\xi_2\kappa/3$ blocks at the beginning of the slot is bounded by

$$P_{\mathsf{out}} \leq L \cdot \exp\left(-\frac{\xi_2^2(\xi_2\kappa - 3)}{9}\right) .$$

Proof. For any fixed slot, let $(\alpha_i)_{i=1}^n$ denote the relative stakes of all currently participating parties, as recorded in the stake distribution used for determining slot leaders for this slot. We first note that the expected number of blocks created in this slot (by all parties on all chains) is

$$m \cdot \sum_{i=1}^{n} \phi_f(\alpha_i) = m \cdot \sum_{i=1}^{n} (1-(1-f)^{\alpha_i}) \overset{(a)}{\leq} m \cdot \ln \frac{1}{1-f} \leq m \cdot \frac{f}{1-f} \leq 1-\xi_2\,,$$

where the inequality (a) follows from Jensen inequality and the observations that $g(n) \triangleq n\left(1-(1-f)^{1/n}\right)$ is increasing and converges to $-\ln(1-f)$ for $n \to \infty$.

Moreover, block-generation can be viewed as elementary independent experiments (one per slot, party, and chain) with respect to an indicator random variable $X_i \in \{0,1\}$ where $X_i = 1$ denotes the respective success. Thus, the probability P_{out} that, during any $\ell_{\mathsf{out}} = \xi_2\kappa/3 - 1$ rounds, at least ℓ_{out} blocks are generated, can be estimated by the Chernoff bound of Theorem 5 with $\mu = (1-\xi_2)\ell_{\mathsf{out}}$ and $\delta = \xi_2/(1-\xi_2)$. The lemma now follows by a union bound over the length of the execution and Lemma 1. □

Lemma 3. *Let $\kappa \in \mathbb{N}$ and $\xi_2 \in (0,1/2)$, let f denote the active slot coefficient in Π_{pc}^m and let m satisfy (3). Consider an L-round execution of Π_{pc}^m with $\mathcal{A}_\perp$. Given that there is no round where all* outbox *queues together contain at least $\xi_2\kappa/3$ blocks at the beginning of the round, the probability that there is a round where a party's* inbox *queue contains at least $(1-\xi_2/3)\kappa$ blocks at the beginning of the round is bounded by*

$$P_{\mathsf{in}} \leq L \cdot \exp\left(-\frac{\xi_2^2}{3}\left(\kappa - \frac{1}{1-\xi_2}\right)\right)\,.$$

Proof. Note that under $\mathcal{A}_\perp$, all parties' inboxes behave identically. Along the lines of Lemma 2, we observe that the probability P_{in} that, during any $\ell_{\mathsf{in}} = (1-2\xi_2/3)k - 1$ rounds, at least ℓ_{in} blocks are generated, is upper bounded as above by the Chernoff bound ($\mu = (1-\xi_2)\ell_{\mathsf{in}}$ and $\delta = \xi_2/(1-\xi_2)$) and the union bound. Additionally, at most $\xi_2\kappa/3$ blocks that were already present in the outbox queues might get added to each inbox during these ℓ_{in} rounds. We can hence conclude by Lemma 1 that the inbox never contains $(1-\xi_2/3)\kappa$ blocks at the beginning of a round — except for the above probability. □

The above lemmas allow us to establish the κ-boundedness of Π_{pc}^m through Theorem 3.

Proof of Theorem 3. Assume that some block has delay at least $\kappa = t_{\mathsf{out}} + t_{\mathsf{in}}$ spending t_{out} rounds in the outbox queue and t_{in} rounds in the inbox queue. Thus, $t_{\mathsf{out}} \geq \xi_2\kappa/3$ or $t_{\mathsf{in}} \geq (1-\xi_2/3)\kappa$. The respective probabilities are given by Lemmas 2 and 3, and

$$\varepsilon_\kappa \leq P_{\mathsf{out}} + (1-P_{\mathsf{out}})P_{\mathsf{in}} \leq 2L \cdot \exp\left(-\frac{\xi_2^2(\xi_2\kappa - 3)}{9}\right)\,,$$

as desired. □

C. Throughput

Now we focus on the throughput achieved by Π_{pc}^m, and express it as a parameter of the number of chains m.

Theorem 4 (Throughput)**.** *Fix parameters $k, \Delta \in \mathbb{N}$, $\xi_3 \in (0,1/2]$, let $f \in (0,1)$ denote the active slot coefficient in Π_{pc}^m, assume that $(1-f)^\Delta \geq 1/2$, and let $\widehat{f} \triangleq f(1-f)^\Delta$. Let $\mathcal{A} \in \mathbb{A}_{\mathsf{b}}$ be an active adversary, let $\alpha_{\mathcal{H}} \in [0,1]$ be a lower bound on the honest stake ratio throughout the execution, let ν be upper bound on the maximum stake shift over $2R$ slots, and assume $\widehat{\alpha_{\mathcal{H}}} \triangleq \alpha_{\mathcal{H}} - \nu > 0$. The protocol Π_{pc}^m achieves (θ, L_0)-throughput for any $L_0 \geq (4k+2\Delta)/((1-\xi_3)^2\widehat{\alpha_{\mathcal{H}}}\widehat{f})$ and for*

$$\theta = (1-\xi_3)^2 \cdot \left(1 - \frac{2k+3\Delta}{(1-\xi_3)^2\widehat{\alpha_{\mathcal{H}}}\widehat{f}L_0}\right) \cdot \widehat{\alpha_{\mathcal{H}}} \cdot \widehat{f} \cdot m \quad (4)$$

against $\mathcal{A}$ except with error probability

$$\varepsilon_{\mathsf{tp}} \triangleq 4 \cdot m \cdot \exp\left(-\xi_3^2\widehat{\alpha_{\mathcal{H}}}fk/2\right) + \varepsilon_{\mathsf{rob}}\,, \quad (5)$$

where $\varepsilon_{\mathsf{rob}}$ is the negligible robustness error-probability from Theorem 2.

Before we prove Theorem 4, we briefly discuss Bound (4). Intuitively, the optimal throughput of Π_{pc}^m is $\widehat{\alpha_{\mathcal{H}}} \cdot f \cdot m$, as each of the m chains contains a block once in f^{-1} slots on expectation, and with probability at least $\widehat{\alpha_{\mathcal{H}}}$ this leader is honest. Three factors separate Bound (4) from this optimum.

First, the term $1-\xi_3$ is used to apply a concentration bound (twice) and can be made arbitrarily small at the expense of increasing the minimum duration L_0 to achieve the stated throughput.

Second, for the term $1-(2k+3\Delta)/(cL_0)$, consider the case where the investigated interval is at the end of the protocol execution. We discount the $2k$ last blocks of this interval from each chain. By chain quality, the first k such blocks contain an honest main-chain block to reference all off-chain blocks published prior to the final sequence of $2k$ blocks. The final k blocks make this history stable. Note that this term also diminishes with increasing L_0.

Third, f is replaced by $\widehat{f}$ to account for possible occurrence of forks that are caused by the probabilistic leader selection in Π_{Praos} in combination with message delays. Note that the assumption $(1-f)^\Delta \geq 1/2$ is natural: we are not interested in a parameterization that does not satisfy it, as (4) would result in a throughput below $1/2$ anyway.

Proof of Theorem 4. Consider an environment $\mathcal{Z}$, let $T \xleftarrow{\$} \mathsf{Exec}(\Pi, \mathcal{A}, \mathcal{Z})$, and let I be a slot interval of length $|I| = L \geq L_0$. We will split the interval I into a *prefix interval* I_{pref} consisting of its first L_{pref} slots of I (for L_{pref} given below), and a *suffix interval* I_{suf} of $L_{\mathsf{suf}} = L - L_{\mathsf{pref}}$ slots. We then argue that with overwhelming probability:

(i) each chain contains at least $(1-\xi_3)^2 \cdot \widehat{\alpha_{\mathcal{H}}} \cdot \widehat{f} \cdot L_{\mathsf{pref}} - \Delta$ blocks in I_{pref};
(ii) all m chains produced by Π_{pc}^m contain at least $2k$ blocks in I_{suf} implying that all blocks contained in all m chains in I_{pref} are stable.

Both parts will rely on establishing a lower bound on the chain growth of a single Ouroboros Praos chain in the presence of an active adversary $\mathcal{A}$, assuming message delays are bounded by Δ.

We will consider Δ-*right isolated successful slots* along the lines of [19], [12]: these are slots with only honest slot leaders where the following Δ slots do not have any leaders at all. We will call these slots *good* for brevity and denote by $\mathsf{good}(t)$ a function that returns 1 if t is good and 0 otherwise. Moreover, let $I'_{\mathsf{pref}} \triangleq [t_1, t_2]$ denote the interval I_{pref} without its first Δ slots and let $L'_{\mathsf{pref}} \triangleq |I'_{\mathsf{pref}}| = |I_{\mathsf{pref}}| - \Delta = t_2 - t_1 + 1$.

We now argue that the growth by productive blocks corresponding to I_{pref} in a chain held by any party at the end of the execution is lower-bounded by the number of good slots appearing in I'_{pref}. This because

- an (honest) slot leader of the first such round will be aware of all blocks that were published during any slots prior to I_{pref} (as we ignore the first Δ slots of I_{pref}), and will thus not duplicate any content that originates from "outside" I_{pref}; and
- an (honest) slot leader of any following such round will have learned all blocks form its prior isolated rounds, and won't duplicate any content of any such prior block.

For a fixed chain $c \in [m]$, let $W_t^{(c)}$ be a random variable that takes the value 0 if the slot t has at least one honest slot leader, and $\perp$ otherwise (the symbols are chosen for consistency with [2]). Given the slot-leader selection rules of Π_{Praos} (and Π_{pc}) described in Section V-A, the probability that a particular slot has at least one slot leader for chain c is exactly f. Thanks to the local nature of the VRF-based leader-selection, $\mathcal{A}$ has no advantage in predicting a slot leader until he acts, and hence $\widehat{\alpha_{\mathcal{H}}} f \leq \Pr[W_t = 0] \leq f$.

To lower-bound the number of good slots in I'_{pref} that produce a block that is guaranteed to be seen by all honest party before the first slot in I_{suf}, we will rely on a concentration bound for this quantity derived in [2] and detailed in Appendix C. More concretely, observe that the sequence of random variables $W_{t_1}^{(c)}, \ldots, W_{t_2}^{(c)}$ corresponding to the interval I'_{pref} satisfies the $(f, 1)$-characteristic condition according to Definition 6 and we can invoke Lemma 4 with $\gamma = 1$ and $a = \widehat{\alpha_{\mathcal{H}}} f$ to bound the probability of having too few good slots by

$$p_{\mathsf{pref}} \triangleq \Pr\left[\sum_{t \in I'_{\mathsf{pref}}} \mathsf{good}(t) \leq (1-\xi_3)^2 \cdot \widehat{\alpha_{\mathcal{H}}} \cdot \widehat{f} \cdot L'_{\mathsf{pref}} - \Delta\right] \leq 2\exp\left(-\frac{\xi_3^2 \cdot \widehat{\alpha_{\mathcal{H}}}^2 \cdot f^2 \cdot L'_{\mathsf{pref}}}{4}\right) .$$

Since $L'_{\mathsf{pref}} = L_{\mathsf{pref}} - \Delta$ and $(1-\xi_3)^2 \cdot \widehat{\alpha_{\mathcal{H}}} \cdot \widehat{f} \leq 1$, this establishes (i).

We now consider the interval I_{suf} for $L_{\mathsf{suf}} \triangleq (2k + \Delta)/\left((1-\xi_3)^2 \widehat{\alpha_{\mathcal{H}}} \widehat{f}\right)$, and similarly as before, we define I'_{suf} to denote the interval of $L_{\mathsf{suf}} - \Delta$ slots obtained when removing the last Δ slots from I_{suf}. By the same argument as above, we observe that I'_{suf} will contain at least $2k$ good slots with an uncorrupted leader (and hence, in I_{suf}, at least $2k$ blocks will be appended to the longest chain that are seen by every honest party after the last slot of the L-interval) except with error $p_{\mathsf{suf}} \triangleq 2\exp\left(-\xi_3^2 \widehat{\alpha_{\mathcal{H}}}^2 f^2 k/2\right)$, establishing (ii).

Now, consider the blocks generated during the slot interval I'_{suf} of which there are at least $2k$ except for error probability p_{suf}. Due to the chain quality of Praos, the first k blocks contain at least one honest block — except for an error probability subsumed by $\varepsilon_{\mathsf{rob}}$ in (5). This block guarantees that all off-chain blocks from Interval I'_{pref} will be referenced by any longest chain extending these k blocks. Finally, by the common-prefix property of Praos, the last k blocks will stabilize this honest block — except for an error probability subsumed by $\varepsilon_{\mathsf{rob}}$ in (5).

Hence we can conclude that for L_{suf} given above and $L_{\mathsf{pref}} \triangleq L - L_{\mathsf{suf}}$, the interval I_{pref} contains at least $(1-\xi_3)^2 \cdot \widehat{\alpha_{\mathcal{H}}} \cdot \widehat{f} \cdot m \cdot L'_{\mathsf{pref}} - \Delta$ *stable* blocks in all m chains except with probability $m \cdot (p_{\mathsf{pref}} + p_{\mathsf{suf}}) + \varepsilon_{\mathsf{rob}}$, where $p_{\mathsf{suf}} = p_{\mathsf{suf}}^{(2k)} + p_{\mathsf{suf}}^{(\mathrm{cq})} + p_{\mathsf{suf}}^{(\mathrm{cp})}$, resulting in the throughput parameter

$$\theta \geq (1-\xi_3)^2 \cdot \left(1 - \frac{2k + 3\Delta}{(1-\xi_3)^2 \widehat{\alpha_{\mathcal{H}}} \widehat{f} L_0}\right) \cdot \widehat{\alpha_{\mathcal{H}}} \cdot \widehat{f} \cdot m \,.$$

Under the assumption $L_0 \geq (4k + 2\Delta)/((1-\xi_3)^2 \widehat{\alpha_{\mathcal{H}}} \widehat{f})$ we have $p_{\mathsf{pref}} \leq p_{\mathsf{suf}}$ and

$$\varepsilon_{\mathsf{tp}} \leq 4 \cdot m \cdot \exp\left(-\xi_3^2 \widehat{\alpha_{\mathcal{H}}}^2 f^2 k/2\right) + \Pr\left[\neg \mathsf{MaxDelay}_\Delta\right] .$$

□

To demonstrate throughput optimality, we instantiate the result of Theorem 4 with the maximum number of chains m that still guarantees κ-boundedness of the resulting protocol Π_{pc}^m according to Theorem 3, yielding the following corollary:

Corollary 1. *Under the notation and assumptions introduced in Theorems 3 and 4. the protocol Π_{pc}^m for*

$$m = (1-\xi_2)(1-f)/f$$

is $(\kappa, \varepsilon_\kappa)$-bounded for any κ and achieves (θ, L_0)-throughput against active adversaries for any $L_0 \geq (4k + 2\Delta)/((1-\xi_3)^2 \widehat{\alpha_{\mathcal{H}}} \widehat{f})$ and

$$\theta = (1-\xi_2) \cdot (1-\xi_3)^2 \cdot (1-f)^{\Delta+1} \cdot \left(1 - \frac{2k + 3\Delta}{(1-\xi_3)^2 \widehat{\alpha_{\mathcal{H}}} \widehat{f} L_0}\right) \cdot \widehat{\alpha_{\mathcal{H}}}$$

except with error probability $\varepsilon_{\mathsf{tp}}$.

Note that the throughput given by Corollary 1 matches the optimum $\theta = \widehat{\alpha_{\mathcal{H}}}$ up to a factor that is the product of values that can be brought arbitrarily close to 1 by tolerating a larger slot range L_0.

REFERENCES

[1] C. Badertscher, P. Ga zi, A. Kiayias, A. Russell, and V. Zikas. Ouroboros chronos: Permissionless clock synchronization via proof-of-stake. Cryptology ePrint Archive, Report 2019/838, 2019. https://eprint.iacr.org/2019/838.

[2] C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 913–930. ACM Press, Oct. 2018.

[3] V. Bagaria, S. Kannan, D. Tse, G. C. Fanti, and P. Viswanath. Deconstructing the blockchain to approach physical limits. *CoRR*, abs/1810.08092, 2018.

[4] L. Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. Swirlds Tech Report SWIRLDS-TR-2016-01, 2016. https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf.

[5] I. Bentov, P. Hubáček, T. Moran, and A. Nadler. Tortoise and hares consensus: the meshcash framework for incentive-compatible, scalable cryptocurrencies. Cryptology ePrint Archive, Report 2017/300, 2017. http://eprint.iacr.org/2017/300.

[6] I. Bentov, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. http://eprint.iacr.org/2016/919.

[7] X. Boyen, C. Carr, and T. Haines. Graphchain: a blockchain-free scalable decentralised ledger. In S. V. Lokam, S. Ruj, and K. Sakurai, editors, *Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts, BCC@AsiaCCS 2018, Incheon, Republic of Korea, June 4, 2018*, pages 21–33. ACM, 2018.

[8] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.

[9] R. Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop*, page 219, 2004.

[10] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. E. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer. On scaling decentralized blockchains - (A position paper). In J. Clark, S. Meiklejohn, P. Y. A. Ryan, D. S. Wallach, M. Brenner, and K. Rohloff, editors, *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, volume 9604 of *Lecture Notes in Computer Science*, pages 106–125. Springer, 2016.

[11] G. Danezis and S. Meiklejohn. Centrally banked cryptocurrencies. In *NDSS 2016*. The Internet Society, Feb. 2016.

[12] B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, Apr. / May 2018.

[13] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In S. Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, Jan. 2005.

[14] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, Santa Clara, CA, 2016. USENIX Association.

[15] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In N. Christin and R. Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454. Springer, Heidelberg, Mar. 2014.

[16] M. Fitzi, P. Gaži, A. Kiayias, and A. Russell. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. Cryptology ePrint Archive, Report 2018/1119, 2018. https://eprint.iacr.org/2018/1119.

[17] S. Forestier and D. Vodenicarevic. Blockclique: scaling blockchains through transaction sharding in a multithreaded block graph. *CoRR*, abs/1803.09029v4, 2018.

[18] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, Apr. 2015.

[19] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. Cryptology ePrint Archive, Report 2016/1048, 2016. http://eprint.iacr.org/2016/1048.

[20] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, Aug. 2017.

[21] L. Kiffer, R. Rajaraman, and a. shelat. A better method to analyze blockchain consistency. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 729–744. ACM Press, Oct. 2018.

[22] S. Lerner. Dagcoin draft. Manuscript, 2015. https://bitslog.files.wordpress.com/2015/09/dagcoin-v41.pdf.

[23] Y. Lewenberg, Y. Sompolinsky, and A. Zohar. Inclusive block chain protocols. In R. Böhme and T. Okamoto, editors, *FC 2015*, volume 8975 of *LNCS*, pages 528–547. Springer, Heidelberg, Jan. 2015.

[24] C. Li, P. Li, W. Xu, F. Long, and A. C. Yao. Scaling nakamoto consensus to thousands of transactions per second. *CoRR*, abs/1805.03870, 2018.

[25] W. Martino, M. Quaintance, and S. Popejoy. Chainweb: A proof-of-work parallel-chain architecture for massive throughput. Manuscript, 2018. http://kadena.io/docs/chainweb-v15.pdf.

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

[27] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.

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

[29] R. Pass, L. Seeman, and a. shelat. Analysis of the blockchain protocol in asynchronous networks. In J. Coron and J. B. Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Heidelberg, Apr. / May 2017.

[30] R. Pass and E. Shi. Hybrid consensus: Efficient consensus in the permissionless model. Cryptology ePrint Archive, Report 2016/917, 2016. http://eprint.iacr.org/2016/917.

[31] R. Pass and E. Shi. FruitChains: A fair blockchain. In E. M. Schiller and A. A. Schwarzmann, editors, *36th ACM PODC*, pages 315–324. ACM, July 2017.

[32] R. Pass and E. Shi. Thunderella: Blockchains with optimistic instant confirmation. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 3–33. Springer, Heidelberg, Apr. / May 2018.

[33] S. Popov. The tangle. Manuscript, 2017. https://iotatoken.com/IOTA_Whitepaper.pdf.

[34] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. SPECTRE: A fast and scalable cryptocurrency protocol. Cryptology ePrint Archive, Report 2016/1159, 2016. http://eprint.iacr.org/2016/1159.

[35] Y. Sompolinsky and A. Zohar. Accelerating Bitcoin's transaction processing. Fast money grows on trees, not chains. Cryptology ePrint Archive, Report 2013/881, 2013. http://eprint.iacr.org/2013/881.

[36] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in bitcoin. In R. Böhme and T. Okamoto, editors, *FC 2015*, volume 8975 of *LNCS*, pages 507–527. Springer, Heidelberg, Jan. 2015.

[37] Y. Sompolinsky and A. Zohar. Phantom: A scalable blockdag protocol. Cryptology ePrint Archive, Report 2018/104, 2018. https://eprint.iacr.org/2018/104.

[38] Y. Sompolinsky and A. Zohar. PHANTOM: A scalable BlockDAG protocol. Cryptology ePrint Archive, Report 2018/104, 2018. https://eprint.iacr.org/2018/104.

[39] C. Stathakopoulou, T. David, and M. Vukolić. Mir-bft: High-throughput bft for blockchains. *arXiv preprint arXiv:1906.05552*, 2019.

[40] R. Stütz, P. Gaži, B. Haslhofer, and J. Illium. Stake shift in major cryptocurrencies: An empirical study. *To appear at Financial Cryptography 2020*, 2020.

[41] H. Yu, I. Nikolic, R. Hou, and P. Saxena. OHIE: blockchain scaling made simple. *CoRR*, abs/1811.12628, 2018.

APPENDIX A
CHERNOFF INEQUALITY

We record here a form of the Chernoff inequality that we use in our arguments. See, e.g., [27] for a proof and further discussion.

Theorem 5 (Chernoff bound)**.** *Let $X_1, \ldots, X_T$ be independent random variables with $\mathsf{E}[X_i] = p_i$ and $X_i \in [0, 1]$. Let*

Fig. 4. **Functionality $\mathcal{F}_{\mathsf{sig}}$**

$\mathcal{F}_{\mathsf{sig}}$ interacts with a signer P_S and stakeholders $\mathsf{P}_1, \ldots, \mathsf{P}_n$ as follows:

- **Key Generation.** Upon receiving a message $(\mathsf{KeyGen}, sid, \mathsf{P}_S)$ from a stakeholder P_S, hand $(\mathsf{KeyGen}, sid, \mathsf{P}_S)$ to the adversary. Upon receiving $(\mathsf{VerificationKey}, sid, \mathsf{P}_S, v)$ from the adversary, output $(\mathsf{VerificationKey}, sid, v)$ to P_i, and record the triple (sid, P_S, v).
- **Signature Generation.** Upon receiving a message $(\mathsf{Sign}, sid, \mathsf{P}_S, M)$ from P_S, verify that (sid, P_S, v) is recorded for some sid. If not, then ignore the request. Else, send $(\mathsf{Sign}, sid, \mathsf{P}_S, M)$ to the adversary. Upon receiving $(\mathsf{Signature}, sid, \mathsf{P}_S, M, \sigma)$ from the adversary, verify that no entry $(M, \sigma, v, 0)$ is recorded. If it is, then output an error message to P_S and halt. Else, output $(\mathsf{Signature}, sid, M, \sigma)$ to P_S, and record the entry $(M, \sigma, v, 0)$.
- **Signature Verification.** Upon receiving a message $(\mathsf{Verify}, sid, M, \sigma, v')$ from some stakeholder P_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 P_i.

$X = \sum_{i=1}^{T} X_i$ *and* $\mu = \sum_{i=1}^{T} p_i = \mathsf{E}[X]$. *Then for all* $\delta \geq 0$ *we have*

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

and for all $0 \leq \delta \leq 1$ *we have*

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

APPENDIX B
HYBRID FUNCTIONALITIES

In Figure 4 we present the functionality $\mathcal{F}_{\mathsf{sig}}$ as defined in [9], where it is also shown that EUF-CMA signature schemes realize $\mathcal{F}_{\mathsf{sig}}$. This functionality is used in [12] to model signatures on transactions.

Figure 5 contains the functionality $\mathcal{F}_{\mathsf{init}}$ that determines the genesis stake distribution $\mathbb{S}_0$ and the nonce η (to be written in the genesis block). $\mathcal{F}_{\mathsf{init}}$ also takes stakeholders' public keys from them and packages them into the genesis block at the outset of the protocol. Note that $\mathcal{F}_{\mathsf{init}}$ halts if it is not possible to create a genesis block; all security guarantees we provide later in the paper are conditioned on a successful creation of the genesis block.

In Figures 6-7 we give a description of the hybrid functionalities $\mathcal{F}_{\mathsf{vrf}}$, $\mathcal{F}_{\mathsf{kes}}$, used by Π_{pc}, these are taken from [12].

Fig. 5. **Functionality $\mathcal{F}_{\mathsf{init}}$**

$\mathcal{F}_{\mathsf{init}}$ 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 $\mathsf{P}_1, \ldots, \mathsf{P}_n$ as follows:

- In the first round, upon a request from some stakeholder P_i of the form $(\mathsf{ver\_keys}, sid, \mathsf{P}_i, v_i^{\mathrm{vrf}}, v_i^{\mathrm{kes}}, v_i^{\mathrm{sig}})$, it stores the verification keys tuple $(\mathsf{P}_i, v_i^{\mathrm{vrf}}, v_i^{\mathrm{kes}}, v_i^{\mathrm{sig}})$ and acknowledges its receipt. If any of the n stakeholders does not send a request of this form to $\mathcal{F}_{\mathsf{init}}$, or if two different stakeholders provide two identical keys, 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 $\mathbb{S}_0 = \Big((\mathsf{P}_1, v_1^{\mathrm{vrf}}, v_1^{\mathrm{kes}}, v_1^{\mathrm{sig}}, s_1), \ldots, (\mathsf{P}_n, v_n^{\mathrm{vrf}}, v_n^{\mathrm{kes}}, v_n^{\mathrm{sig}}, s_n)\Big)$.
- In later rounds, upon a request of the form $(\mathsf{genblock\_req}, sid, \mathsf{P}_i)$ from some stakeholder P_i, $\mathcal{F}_{\mathsf{init}}$ sends $(\mathsf{genblock}, sid, \mathbb{S}_0, \eta)$ to P_i.

Finally, the functionality $\mathcal{F}_{\mathsf{ro}}$ represents a random oracle and its description is omitted.

APPENDIX C
SOME RESULTS FROM [2]

Here we present a technical result from [2] (Lemma 4 below) that will be useful for the throughput analysis of our PoS-based construction in Section V. First, we need to introduce some notation and terminology. We keep the definitions below in their full generality as presented in [2], [12], even though, looking ahead, we will only use them in a setting where the random variables W_i contain no 1-symbols (i.e., $\gamma = 1$).

Definition 6 (The characteristic conditions [2])**.** *Consider a family of random variables* $W_1, \ldots, W_n$ *taking values in* $\{0, 1, \perp\}$. *We say that they satisfy the* $(f;\gamma)$-characteristic conditions *if, for each* $k \geq 1$,

$$\Pr[W_k = \perp \,|\, W_1, \ldots, W_{k-1}] \geq (1-f),$$
$$\Pr[W_k = 0 \,|\, W_1, \ldots, W_{k-1}, W_k \neq \perp] \geq \gamma, \textit{and hence}$$
$$\Pr[W_k = 1 \,|\, W_1, \ldots, W_{k-1}, W_k \neq \perp] \leq 1-\gamma.$$

In the expressions above, conditioning on a collection of random variables indicates that the statement is true for any conditioning on the values taken by variables.

Definition 7 (Reduction mapping [12])**.** *For* $\Delta \in \mathbb{N}$, *we define the function* $\rho_\Delta \colon \{0,1,\perp\}^* \to \{0,1\}^*$ *inductively as follows:*

$$\rho_\Delta(\epsilon) = \epsilon,$$
$$\rho_\Delta(\perp \,\|\, w') = \rho_\Delta(w'),$$
$$\rho_\Delta(1 \,\|\, w') = 1 \,\|\, \rho_\Delta(w'),$$
$$\rho_\Delta(0 \,\|\, w') = \begin{cases} 0 \,\|\, \rho_\Delta(w') & \textit{if } w' \in \perp^\Delta \,\|\, \{0,1,\perp\}^*, \\ 1 \,\|\, \rho_\Delta(w') & \textit{otherwise}. \end{cases}$$

We call ρ_Δ *the* reduction mapping for delay Δ.

For our purposes, it is important that when the reduction mapping of Definition 7 is applied to a sequence W of symbols 0 and $\perp$ representing slots with and without an uncorrupted leader respectively, then every good slot results in a 0-symbol

Fig. 6. **Functionality $\mathcal{F}_{\mathsf{vrf}}$**

$\mathcal{F}_{\mathsf{vrf}}$ interacts with stakeholders $\mathsf{P}_1, \ldots, \mathsf{P}_n$ as follows:

- **Key Generation.** Upon receiving a message (KeyGen, sid) from a stakeholder P_i, hand $(\mathsf{KeyGen}, sid, \mathsf{P}_i)$ to the adversary. Upon receiving $(\mathsf{VerificationKey}, sid, \mathsf{P}_i, v)$ from the adversary, if P_i is honest, verify that v is unique, record the pair (P_i, v) and return $(\mathsf{VerificationKey}, sid, v)$ to P_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 P_i, verify that some pair (P_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_i, 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 P_i, verify that some pair (P_i, v) is recorded. If not, then ignore the request. Else, send $(\mathsf{EvalProve}, sid, \mathsf{P}_i, M)$ to the adversary. Upon receiving $(\mathsf{EvalProve}, sid, M, \pi)$ 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_i.
- **Malicious VRF Evaluation.** Upon receiving a message $(\mathsf{Eval}, sid, v, M, \pi)$ 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 $T(v, M) = (y, S)$ and output $(\mathsf{Evaluated}, sid, y)$ to $\mathcal{S}$. The same is performed in case (P_i, v) is recorded and P_i corrupted. Else, if $T(v, M) = (y, S')$ for some $S' \neq \emptyset$, union S to S' and 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 $(\cdot, v)$ and the entry $T(v, M)$ equals (y, S) with $\pi \in S$, then set $f = 1$.
  2) Else, if $v' = v$ for some recorded pair of the form $(\cdot, v)$, but no entry $T(v, M)$ 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. 7. **Functionality $\mathcal{F}_{\mathsf{kes}}$**

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

- **Key Generation.** Upon receiving a message $(\mathsf{KeyGen}, sid, \mathsf{P}_S)$ from a stakeholder P_S , send $(\mathsf{KeyGen}, sid, \mathsf{P}_S)$ to the adversary. Upon receiving $(\mathsf{VerificationKey}, sid, \mathsf{P}_S, v)$ from the adversary, send $(\mathsf{VerificationKey}, sid, v)$ to P_S, record the triple (sid, P_S, v) and set counter $\mathsf{k}_{\mathsf{ctr}} = 1$.
- **Sign and Update.** Upon receiving a message $(\mathsf{USign}, sid, \mathsf{P}_S, M, j)$ from P_S, verify that (sid, P_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 $(\mathsf{Sign}, sid, \mathsf{P}_S, M, j)$ to the adversary. Upon receiving $(\mathsf{Signature}, sid, \mathsf{P}_S, M, j, \sigma)$ from the adversary, verify that no entry $(M, j, \sigma, v, 0)$ is recorded. If it is, then output an error message to P_S and halt. Else, send $(\mathsf{Signature}, sid, M, j, \sigma)$ to P_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 P_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 P_i.

in $\rho(W)$. Note that the definition from [12] was slightly adjusted to have this property in our context where we need to isolate good slots by at least Δ empty slots, the statement below was adjusted accordingly.

Let $\#_s(X)$ denote the number of occurrences of the symbol s in the string X. The following lemma is a minor technical adaptation of Lemma 8(iv) from [2], where a constant $1/4$ is replaced by $1 - \xi_3$ in the application of concentration bounds and ε is set to 0.

Lemma 4 ([2], Lemma 8(iv))**.** *Let $W = W_1 \cdots W_n$ be a sequence of random variables, each taking values in $\{\perp, 0, 1\}$, which satisfy the $(f; \gamma)$-characteristic conditions and let*

$$X = X_1 \cdots X_\ell = \rho_\Delta(W_1 \cdots W_n)$$

be the random variables obtained by applying the reduction mapping (for delay Δ) to W. If

$$\Pr[W_i = \perp \mid W_1, \ldots, W_{i-1}] \leq (1 - a) \text{ and } \gamma(1 - f)^\Delta \geq 1/2$$

then for any $\xi_3 \leq 1/2$ we have

$$\Pr\left[\#_0(X) < (1 - \xi_3)^2 \gamma (1 - f)^\Delta a n - \Delta\right] \leq 2 \exp\left(-\frac{\xi_3^2 a^2 n}{4}\right).$$

APPENDIX D
GLOSSARY OF USED NOTATION

Protocol Parameters

R	epoch length in slots
L	total length of protocol execution in slots, $L = ER$
m	number of parallel chains
k	number of blocks that can be reverted in the selectchain procedure, see Figure 3
f	active block coefficient, see Equation (2)
$\widehat{f}$	notational shorthand for $f(1-f)^{\Delta}$
T_i^j	leadership threshold, see Equation (1)

Analysis Parameters

θ, L_0	throughput parameters, see Definition 5
μ	bandwidth-asymmetry parameter, see Section III
Δ	upper bound on network delay
ν	upper bound on stake shift
$\alpha_{\mathcal{H}}$	lower bound on honest stake ratio
$\widehat{\alpha_{\mathcal{H}}}$	notational shorthand for $\alpha_{\mathcal{H}} - \nu$
q	number of adversarial random-oracle queries
κ	parameter of the κ-boundedness property, see Definition 3
ξ_1	concentration-bound parameter from Theorem 1
ξ_2	concentration-bound parameter from Theorem 3
ξ_3	concentration-bound parameter from Theorem 4

Functionalities

$\mathcal{F}_{\mathsf{diff}}$	diffuse functionality modeling peer-to-peer network, see Figure 1
$\mathcal{F}_{\mathsf{sig}}$	digital signature, see Figure 4
$\mathcal{F}_{\mathsf{vrf}}$	verifiable random function, see Figure 6
$\mathcal{F}_{\mathsf{kes}}$	key-evolving digital signature, see Figure 7
$\mathcal{F}_{\mathsf{ro}}$	random oracle