Executing and Proving over Dirty Ledgers

Christos Stefo[1,2], Zhuolun Xiang[3], and Lefteris Kokoris-Kogias[1,4]

[1] IST Austria
[2] National Technical University of Athens
[3] Aptos Labs
[4] Mysten Labs

Abstract. Scaling blockchain protocols to perform on par with the expected needs of Web3.0 has been proven to be a challenging task with almost a decade of research. In the forefront of the current solution is the idea of separating the execution of the updates encoded in a block from the ordering of blocks. In order to achieve this, a new class of protocols called rollups has emerged. Rollups have as input a total ordering of valid and invalid transactions and as output a new valid state-transition.
If we study rollups from a distributed computing perspective, we uncover that rollups take as input the output of a Byzantine Atomic Broadcast (BAB) protocol and convert it to a State Machine Replication (SMR) protocol. BAB and SMR, however, are considered equivalent as far as distributed computing is concerned and a solution to one can easily be retrofitted to solve the other simply by adding/removing an execution step before the validation of the input.
This "easy" step of retrofitting an atomic broadcast solution to implement an SMR has, however, been overlooked in practice. In this paper, we formalize the problem and show that after BAB is solved, traditional impossibility results for consensus no longer apply towards an SMR. Leveraging this we propose a distributed execution protocol that allows reduced execution and storage cost per executor ($O(\frac{tog^2n}{n})$) without relaxing the network assumptions of the underlying BAB protocol and providing censorship-resistance. Finally, we propose efficient non-interactive light client constructions that leverage our efficient execution protocols and do not require any synchrony assumptions or expensive ZK-proofs.

1 Introduction

The rise of blockchain technology has lead to the rapid development of a variety of solutions for the State Machine Replication (SMR) problem. Nodes running an SMR algorithm need to both order a set of transactions as well as execute them to update their local state, two separate responsibilities that are usually conflated into a single consensus protocol. Recently, the idea of separating the total ordering of transactions from the execution has shown tremendous promise on increasing the scalability of blockchains [11,16,31] however all existing research focuses on the ordering layer assuming that after ordering every participant can locally execute the transactions and update the state.

In this work, we investigate the question of "how to scale execution after the ordering is done". In other words, given that transactions are ordered, how scalable can an execution protocol be. Currently there exist two proposed solutions. The first and most prevalent is that every consensus-node also executes and adds a commitment to the new-state on a succeeding block [5,11,13]. The second relies on a semi-trusted executor node that runs a "rollup" protocol [7]. The executor proposing a new state after locally executing the ordered transactions either provides a sufficiently large dispute window for some honest executor to challenge the proposal with a fraud proof [9,4,2], or a zk-proof [12,1] of correct execution. Neither of these solutions are built from first principles, the former is merely a synchrony assumption breaking the model of the underlying ordering-layer [20,25,14,13,24] whereas the later is proposed as a remedy to that assumption which forces mostly inefficient and non-general purpose zero-knowledge proof usage as well as allows for the executor to censor transactions.

In this paper, we take a step back and design from first principles. As a first contribution we merely point out that decoupling of ordering from execution is nothing more than taking a Byzantine Atomic Broadcast (BAB) [17], i.e. ensuring the total ordering of sent messages, and a deterministic execution engine [19,21,3] to solve the SMR problem. In blockchain systems, the BAB layer is called a *dirty* ledger because transactions are not checked for validity. The nodes taking part in the network, which we call *consensus nodes*, commit transactions without validation and only make sure the ledger is growing consistently.

Once we define our problem we propose a novel protocol for the execution layer of an underlying dirty ledger for both the permissioned and the Proof-of-Stake settings. Our protocol works in an asynchronous environment, making no extra assumptions and does not use zk-proving machinery. We merely assume the existence of a dirty ledger, that ensures both the total ordering and the availability of the transactions committed to it. Then, for the execution layer, we use a set of nodes that we call *executors*. They validate transactions and update the state of the system and can be a subset of the consensus nodes or external. Surprisingly an honest majority is sufficient for the executors even though we have no timing assumptions and only a poly-logarithmic number of them needs to execute every block. As a result our solution provides both better fault tolerance ($f \leq (1-\epsilon)\frac{n}{2}$ instead of $f < \frac{n}{3}$) and significantly better scalability in two dimensions, execution and storage, with expected $O(\frac{log^2 n}{n})$ (instead of $O(1)$) cost per executor per block meaning that the system can be truly scalable and decentralized.

Our Approach.

Our protocol can be roughly split into two steps. In the first step, we *elect* on expectation one executor per round by computing a VRF [29]. Then the executor *votes* by computing state commitments for the next $O(log^2 n)$ rounds. Hence every round will have an $O(log^2 n)$ number of executors. Their task is to construct verifiable certificates of the state such that a user (*executor* or light

client) can be convinced about the state without execution. At first glance, that problem seems related to the consensus problem [26] since executors need to agree on the state, but unlike the consensus problem, in each round all honest nodes have the same input, an ordered list of transactions. Therefore, as long as honest executors bootstrap in the correct state, a state commitment can be considered valid if and only if at least one honest executor has voted on it.

Since nodes update the state in a distributed fashion, we must guarantee its availability. More specifically, in each round, only the elected nodes obtain the state. However, to vote for the following $O(log^2 n)$ rounds, elected executors must acquire the state of the previous round. For that reason, every node stores the state of the rounds it has executed and provides it upon request.

A final general challenge for dirty ledgers is to define light client constructions. Straightforward solutions such as providing inclusion proofs for the transactions committed to the ledger are not sufficient since the transactions can be invalid. To solve this challenge, we present the first non-interactive light client construction for dirty ledgers in the asynchronous model. In a nutshell, light clients learn information about the state by verifying the certificates produced by the executors, i.e., the valid state commitments, along with inclusion proofs.

Our paper has the following contributions.

- We formalize the SMR problem by separating it into ordering and execution.
- We propose a solution for the SMR execution layer with $O(\frac{log^2(n)}{n})$ cost per executor and near-optimal fault tolerance $f < (1-\epsilon)n/2$, assuming the existence of the ordering layer.
- We extend our protocol for the proof-of-stake settings.
- We introduce the first non-interactive light client for dirty ledgers.

The structure of the paper is as follows. In Section 2 we present the related work, model and assumptions, as well as the problem definition of scaling execution. In Section 3, we overview our solutions for different settings, present the detailed solution of Horizontal Sampling in Section 4, and defer the solutions of deterministic and Proof-of-Stake in Appendix B. Section 5 discusses the light client protocol and Section 6 discusses the data availability problem. We provide a summary of terminologies in Appendix A, and all proofs of the protocols are deferred to Appendix C due to space constraints.

2 Preliminaries

In this section, we give an overview of the related work on two components, separating the ordering and execution of transactions and defining light client constructions for dirty ledgers. Then, we define the model and the assumptions that our protocols build upon. Last, we define formally an SMR architecture composed of an ordering and an execution layer and the light client constructions.

2.1 Related work

The natural way to separate the ordering from the execution is to let each node execute every round and add the state commitment to a subsequent block, leading to an average cost per block execution $O(1)$ [5,11,13]. The other promising approach to moving the computation of the state off-chain is employing a *rollup* protocol where a coordinator, updates the state of the system locally and only posts the state commitment on the main chain. There are two directions to verify the state commitments, *optimistic rollups* [6] and *ZK-rollups* [8].

In *optimistic rollups*, there is a dispute period during which executors can prove that a state commitment posted on the main chain is invalid. However, this technique requires synchrony assumptions and average cost per block execution $O(1)$ to guarantee that only valid state commitments are posted on the main chain. On the other hand, in *ZK-rollups*, the coordinator commits the state commitment along with a zero-knowledge proof (ZK-STARK) indicating that a specific set of transactions has been applied to the state. Nevertheless, *ZK-rollups* are not *censorship-resistant* since the coordinator can just not include some valid transactions in this set. Furthermore, computing ZK-STARKs is a computationally heavy for the users, and scaling general-purpose applications is challenging due to the difficulty to express general computation.

Finally, on the light-client for dirty ledgers domain, Tas et al. [32] proposed the first such solution in the synchronous model. In that work, a number of nodes, which are called $full\ nodes$, are in charge of updating the state of the system and providing state commitments to the light clients, proving their validity through an interactive game (bisection game). Unlike this solution we propose a light-client construction that is non-interactive, third-party verifiable (i.e., if a node is convinced it can convince other nodes as well) and works in the asynchronous model. This however, comes at the cost that we require an honest majority of executors that cannot be bribed or adaptively corrupted (for the probabilistic solution). We can also employ a fall-back mode in the protocol where any client not happy with the assumption above, but who assumes synchrony waits for any of the elected executors to provide a fraud-proof [10] or ask an honest full-node for the correctness of a state-commitment through bisection games. Both approaches have an honest minority assumption which will always be true with overwhelming probability. As a result, our proposal can easily be adapted to a flexible model [27] for heterogeneous clients.

2.2 Model and Assumptions

Communication model: We assume an asynchronous environment, where any message sent can be delayed for an unspecified, but finite, amount of time. The link between every two honest nodes is reliable, namely when an honest node sends a message to another honest node, the message will eventually arrive.
Cryptographic Primitives: We use κ to denote the security parameter. We assume a large number of participants and let the security parameter be a function of that number as we will discuss, while in Appendix D we propose a

fallback-mechanism for a small number of participants. We assume the adversary is computationally bounded, the communication channels are cryptographically secure, and the existence of hash functions, signatures, and encryption schemes. We use a computationally hiding and perfectly binding commitment scheme: $(Compute_{cmt}, Verify_{cmt})$. We require the commitment scheme to be deterministic and provide inclusion proofs, e.g., it can be a Merkle tree. Moreover, users employ a Verifiable Random function (VRF) [29].

Permissioned setting: We consider a fixed number of n nodes with their public keys known to every participant in the network. A genesis block G which describes the initial state of the system is provided both to the executors and to the clients. The adversary is static and can corrupt up to $f \leq (1-\epsilon)\frac{n}{2}$ nodes in a Byzantine fashion before the protocol starts. We discuss how to extend the protocols to deal with an adaptive adversary in Appendix F.

Proof of stake: With the term node we refer to each identity that has an account on the system. The point of reference in the proof-of-stake system is a unit coin, which is the smallest amount of money existing. Each coin is a unique string linked to its owner. We assume each node is equipped with a private-public key pair. A genesis block G which contains the initial stake distribution is accessible to all nodes. The stake distribution is dynamic, namely the coins might change hands over time. We assume that the total amount of stake is fixed and equal to W in every round. The adversary is static and can corrupt a portion of the stake holders holding at most f coins, such that $f \leq (1-\epsilon)\frac{W}{2}$ where $0 < \epsilon < \frac{1}{2}$, in a Byzantine fashion.

2.3 Problem Definition

Cohen et al. [15] introduced a modular SMR architecture, separating the data dissemination, ordering, and execution and they investigated solutions for the dissemination part. In this paper, we formulate the State Machine Replication (SMR) problem by diving it into an ordering and an execution layer. Solutions for the ordering layer include Blockchain protocols such as Byzantine Atomic Broadcast (BAB) [33,14,13,24], in which the nodes only agree on the order of the blocks without executing them. Our protocols are solutions for the execution layer.

State Machine Replication (SMR): A state machine consists of a set of state variables that encode its current state. External identities, users of the system, can issue commands to the state machine. The state machine executes the commands sequentially using a transition function to update the state of the system. Furthermore, the state machine might generate an output after executing each command. To provide fault-tolerant behavior, the state machine replicates in multiple copies. An SMR protocol aims to maintain synchronization between the replicas. In this paper, we illustrate that an SMR solution can be a composition of a protocol Π_1 for the *ordering layer* and a protocol Π_2 for the *execution layer*. Below, we define the *ordering* and the *execution* layers.

Ordering layer: Consider a number of nodes, some of which can be adversarial, receiving transactions from external identities. The nodes organize the

transactions in blocks. Furthermore, they employ a protocol Π_1 to agree on an order of the blocks. Each node i commits locally to a finalized ledger of blocks. We denote the ledger to which node i commits in the round r by T_r^i. The output of the ordering protocol, i.e. the order of the blocks in which the nodes reach, is a ledger $T = b_0 \leftarrow b_1 \leftarrow ... \leftarrow b_i$. We introduce the properties that an ordering protocol must satisfy:

- *O-Safety*: There is no round r for which exist two honest nodes i, j s.t. $T_r^i \neq T_r^j$.
- *O-Liveness*: If an honest node receives an input tx, then all honest nodes will eventually include tx in a block of their local ledger.

Execution Layer: Consider a number of nodes where some of them can be adversarial. Moreover, consider the ledger of blocks $T = b_0 \leftarrow b_1 \leftarrow ... \leftarrow b_i$ output by the ordering layer, accessible to everyone. Each block might contain invalid transactions. The validity of a transaction depends on the logic of the application. The nodes are responsible for applying only the valid transactions within the blocks committed to the ledger T. The invalid transactions within the blocks are disregarded. Each node updates the state of the system. We denote the state of the system in the round r according to node's i view by S_r^i.

State: As a blockchain state, we denote a structure keeping track of each user's possessions. The content of the state depends on the type of transactions committed to the ledger. For instance, in Ethereum, the state captures the balance accounts of the users, while in Bitcoin the UTXO model is adopted. Furthermore, the state can contain fragments of code, e.g., smart contracts.

Ideal functionality Π: We illustrate the correctness of the state of the system by introducing an ideal functionality Π. The functionality Π receives as input the ledger $T = b_0 \leftarrow b_1 \leftarrow ... \leftarrow b_i$ which is an output by the ordering layer. Π updates the state by applying all (and only) the valid transactions within the blocks committed to the ledger T. We denote the state of the system stored by Π for the round r by S_r^*. The initial state of the system S_0^* equal to the genesis block, $S_0^* = G$. To update the state in each round, Π uses the deterministic transition function *apply*. The inputs are the state of the previous round and the block to be executed in the current round. More specifically, in the round r, $S_r^* \leftarrow apply(b_r, S_{r-1}^*) = S_{r,len(b_r)}$ where

$$S_{r,j} = \begin{cases} S_{r-1}^* & \text{if } j = 0 \\ \text{apply\_tx}(S_{r,j-1}, tx_j) & \text{if } 1 \leq j \leq len(b_r) \end{cases} \text{ and } b_r = [tx_1, ..., tx_{len(b_r)}].$$

The state applied to the function *apply_ tx* remains unchanged when the input tx is an invalid transaction, namely *apply_ tx(S,tx)=S*. Therefore, for the ledger T, there exists a unique sequence of states $S_0^*, S_1^*, ..., S_i^*$ defined by the state transition function above.

In practice, nodes can employ any execution engine M that simulates the ideal functionality Π. When receiving as inputs the correct state of r and the block $r + 1$, the engine M outputs the correct state of $r + 1$.

An execution layer guarantees that the honest nodes simulate the ideal functionality Π. We proceed with defining the properties of an execution layer:

- *E-Safety*: There is no round r for which exists an honest node i that commits on a state S_r^i s.t. $S_r^i \neq S_r^*$.
- *E-Liveness*: For any round r where an honest node i commits a state S_r^i, there exists a round $r' > r$ where node i eventually commits a state $S_{r'}^i$ s.t. $S_r^i \neq S_{r'}^i$.

Since nodes keep updating their state without deviating from the ideal functionality Π, an execution protocol is *Censorship Resistant*, namely it satisfies the following property:

- *Censorship Resistance:* Every valid transaction tx committed to the ledger T will eventually be applied in the state.

Note that the liveness property ensures only that each honest node will eventually update its state. Since not all nodes execute for every round essentially, we do not require that the honest nodes update their states in the same rounds.

State Machine Replication: Finally, we formulate the SMR problem on top of the ordering and execution layers. More specifically, transactions issued by external identities constitute the input of the SMR. An SMR protocol consists of an ordering layer protocol Π_1 and an execution layer protocol Π_2 satisfying the properties *O-Safety*, *O-Liveness* and *E-Safety*, *E-Liveness* respectively. Nodes participating in those protocols may or may not be the same. The output of Π_1 which is a ledger of blocks T is the input of Π_2. The output of the machine is the output of Π_2, namely an ever-growing state sequence $S_0, S_1, S_2, ..., S_i$.

Light Client. Consider an execution layer Π_e with input a ledger T and the average size of the state that the ideal functionality Π outputs in any round $|S|$. The execution layer supports light client constructions. Light clients request succinct proofs from the participating nodes to learn desired information about the state of the system. We capture this idea by defining the *state proof* certificates.

- A *state proof* π_S for the round r is a succinct proof indicating that the state S is the correct state of the round r. Proof π_S is correct if and only if $S = S_r^*$, where S_r^* is the output of the functionality Π for the round r.
- A *state proof* π for the round r is succinct if it contains asymptotically less data than the history of states, namely if $\frac{len(\pi)}{r|S|} = o(1)$

Assume that the light client lc_i receives a *state proof* π_S for the round r without necessarily receiving the state S. Lc_i evaluates whether the proof is correct, in its perception, using a predicate $accept_{lc_i}(\pi_S, r)$ which yields either True or False. The light client lc_i accepts π_S if and only if $accept_{lc_i}(\pi_S, r) = True$. The properties which a light client execution layer protocol must satisfy are the following:

- *LC-Safety*: There is no round r for which exists a light client lc_i that receives a proof π_S for the round r s.t. $accept_{lc_i}(\pi_S, r) = True$ and $S \neq S_r^*$.
- *LC-Liveness*: A light client bootstrapping in the round r will eventually receive a proof $\pi_{S_{r'}}$ for a round $r', r' \geq r$ s.t. $accept_{lc_i}(\pi_{S'}, r') = True$.

Additional Assumptions: We assume the existence of an underlying ledger T as an output of an *ordering layer* Π that satisfies the properties of *O-Safety*, *O-Liveness*. The ledger T is accessible to every node. Furthermore, we assume that there are no duplicate transactions committed to T. Finally, we assume that for each round a random seed is provided by the dirty ledger, similarly to Algorand [22], or DAG-based BFT protocols [24,16,23].

3 Overview of the Protocols

In our proposed protocols, executors update the state of the system in a distributed fashion. We decompose the protocols in two phases, an *election phase* and a *voting phase*. The *election phase* will select a set of executors for every round. Then, in the *voting phase* the elected executors of that round compute and broadcast their signed state commitments. The *voting phase* outputs *valid* state commitments, as defined:

- A state commitment is considered to be valid if and only if either it is signed by at least one honest node or it is the genesis block.

The goal is to ensure that only *correct* state commitments, defined below, will become *valid*.

- A state commitment *cmt* is the *correct* state commitment of round r if and only if $cmt = compute_{cmt}(S_r^*)$, where S_r^* is state of round r defined by the ideal functionality of the execution layer as in Section 2.

There are two challenges when solving the problem. The first is to ensure that there is provably at least one honest node that has voted a state commitment to guarantee its validity. The second is to ensure that when elected nodes enter the *voting phase*, they have the state of the previous round available. Below we explain how we tackle these challenges for different settings.

Permissioned and Deterministic. First, we present a straightforward deterministic protocol for the permissioned settings to lay the foundation of our other solutions. We consider a total number of $n = 2f + 1$ executors (instead of $n > 3f$), with f executors corrupted by a static adversary. Every node executes for each round, i.e., each executor starts from the genesis block and updates the state by applying the valid transactions of the dirty ledger. For every round, executors compute, sign, and broadcast the corresponding state commitment. A state commitment is valid if it is signed by at least $f + 1$ nodes so that at least one honest node is included.

Probabilistic solutions. The straightforward deterministic solution requires every executor to run for every round, which is not scalable. For better scalability, we propose probabilistic protocols for the permissioned and the Proof-of-Stake settings. We assume up to $f \leq (1 - \epsilon)\frac{n}{2}$ executors can be corrupted by a static adversary, where ϵ is some constant. Our protocols guarantee the validity of a state commitment by requiring a threshold of executors to sign the state commitment. To ensure safety, the number of adversarial nodes executing in each

round must be less than this threshold. To ensure liveness, in each round, there must be enough honest nodes executing to form a valid state commitment. We set the threshold for the valid state commitment to be 1/2 of the number of elected executors, and demonstrate that the aforementioned property is satisfied with a overwhelming probability in the security parameter by electing only a poly-logarithmic number of nodes per round.

Vertical vs Horizontal Sampling. The straightforward probabilistic solution is to elect a *committee* of poly-logarithmic size per round who broadcasts signed state commitments. A state commitment is considered valid if it is signed by at least half of the committee members. We call this approach *Vertical Sampling*. Each node is elected on average once per $O(\frac{n}{polylogn})$ rounds and executes for only the respective rounds. Instead, we adopt an approach we call *Horizontal Sampling*, in which only expected constant number (e.g. one) of nodes are elected per round. In that solution, every node is elected on average every n rounds and executes for $O(polylogn)$ rounds. In both cases the cost per block execution is $O(\frac{polylogn}{n})$. However, since nodes update their execution states in a distributed fashion, elected nodes may need to retrieve the previous execution state from other nodes in order to execute the current round, which incurs high communication overhead. In *Horizontal Sampling*, in comparison to the *Vertical Sampling*, nodes request the state less frequently, resulting in a more scalable solution.

Permissioned and Randomized. First, we present the *Horizontal Sampling* protocol for the permissioned settings. During the *election phase*, each executor computes the VRF locally in each round. Only one node on average is elected per round. The elected node starts from the state of the previous round, computes and broadcasts state commitments for the following $O(polylogn)$ rounds. Hence, with only one executor elected per round, a poly-logarithmic number of nodes will vote for each round. State commitments signed by at least half of the elected nodes are considered valid.

Proof-of-stake (PoS). We then extend the *Horizontal Sampling* protocol for the Proof-of-Stake settings. In the permissioned settings, each node computes a VRF for the *election phase*. In PoS, the adversary can create numerous accounts to increase the probability of being elected. To make the protocol Sybil Resistant, each node's election probability is proportional to its stake. Concretely, nodes compute the VRF for all of their coins in the *election phase*. In the *voting phase*, elected nodes compute and broadcast their signed state commitments, as in the permissioned protocol.

An extra challenge in the PoS protocol is that the stake distribution changes over time. In every round, each node keeps track of its own stake and only the elected nodes execute the state. Therefore, elected nodes must prove the ownership of elected coins to the rest of the nodes. To this end, they construct and broadcast inclusion proofs along with their signed state commitments.

State availability. In the probabilistic protocols, not all nodes execute for every round to acquire the respective the state of every round. For liveness, our protocol must guarantee state availability, i.e.,. any node is able to acquire

the state of the previous round every time when it executes the current round. Since any valid state commitment is signed by at least one honest node, the corresponding state will eventually be available to any node requesting it.

Light clients. Lastly, we introduce a non-interactive light client construction for our protocols. We assume that at any given time, each light client is connected to at least one honest executor. Briefly, a non-interactive light client can learn information about the state of the system after receiving a valid state commitment from an executor, along with an inclusion proof (e.g. Merkle proof).

4 Protocols

In this section, we present our asynchronous execution layer protocols on top of an underlying dirty ledger. First, in the permissioned settings, we present the deterministic protocol demonstrating how to construct verifiable certificates that correspond to the correct state, i.e., the valid state commitments. The deterministic protocol suggests that a majority of honest nodes is a necessary and sufficient condition to construct valid state commitments. Due to its simplicity, we omit the details here and refer the reader to Appendix B.1. However, in the deterministic protocol, every node executes for every round resulting in cost per block execution O(1). Next, we define a probabilistic scalable protocol called Horizontal Sampling, where in every round we select only a poly-logarithmic number of nodes to execute so that the majority of them are honest with overwhelming probability. Due to space limitations, we only present the details of the horizontal sampling protocol and some intuitive descriptions for the Proof-of-Stake protocol in Section 4.2 in the main paper, and leave other protocol details in Appendix B.

4.1 Horizontal Sampling

In the deterministic protocol, all nodes execute in every round and broadcast their state commitments. Now, we proceed with building an efficient probabilistic protocol, called *Horizontal Sampling*, illustrated in Algorithm 1. We assume up to $f \leq (1-\epsilon)\frac{n}{2}$ executors can be corrupted by a static adversary where ϵ is some constant, and we choose the security parameter $\kappa = O(\log^2 n)$ for this section. Nodes first download the genesis block G which holds the initial state. In each round, every node checks whether it is elected (Algorithm 1, line 28). Elected nodes propose state commitments during the *voting phase*. The *voting phase* outputs valid state commitments, which are state commitments signed by enough executors.

Election phase: In each round, every node computes the VRF using its private key, the round number, and the corresponding random seed. This computation returns two values, a hash value of length $|h|$ and a proof of authenticity certifying this hash value (Algorithm 1, line 27). We refer to this proof as the *proof of election* of the leaders. All nodes with hash value in round r of less

than $X_r = \begin{cases} \kappa\frac{2^{|h|}}{n}, & \text{if } r = 1 \\ \frac{2^{|h|}}{n}, & \text{if } r > 1 \end{cases}$ are elected (Algorithm 1, line 28). In that way, in the first round there will be expected κ elected nodes constituting the *bootstrap committee*, while for $r > 1$ there will be only one node in expectation, which is called the leader.

Validity of a commitment: For the first κ rounds only the members of the *bootstrap committee* are voting. For any round $r \geq \kappa+1$ all the elected nodes in the interval $[r - \kappa + 1, r]$ compute the state commitments. In Lemma 5, we prove that the *bootstrap committee* consists of at least $\frac{\kappa}{2}$ honest nodes and at most $\frac{\kappa}{2} - 1$ adversarial nodes with overwhelming probability in n (we choose $\kappa = O(\log^2 n)$). The same property holds for the elected nodes in any interval of κ consecutive rounds according to Lemma 6. As a result, in each round, at least $\frac{\kappa}{2}$ honest and at most $\frac{\kappa}{2} - 1$ adversarial nodes will be responsible for voting. Therefore, a state commitment corresponding to a round r can be considered as valid if it is signed by at least $\frac{\kappa}{2}$ nodes among those that are elected to execute during the interval of rounds $[max(1, r - \kappa + 1), r]$ or if it is the genesis block. In figure 1, on the left side we present an example of the leaders' votes in the interval $[r, r + 3]$ where the malicious leader L_{r+2} votes for incorrect state commitment for rounds $r + 2, r + 3$; on the right side we present an example of the committee members voting for the execution state commitments of different rounds, and the malicious nodes try to create a fork on the execution state.

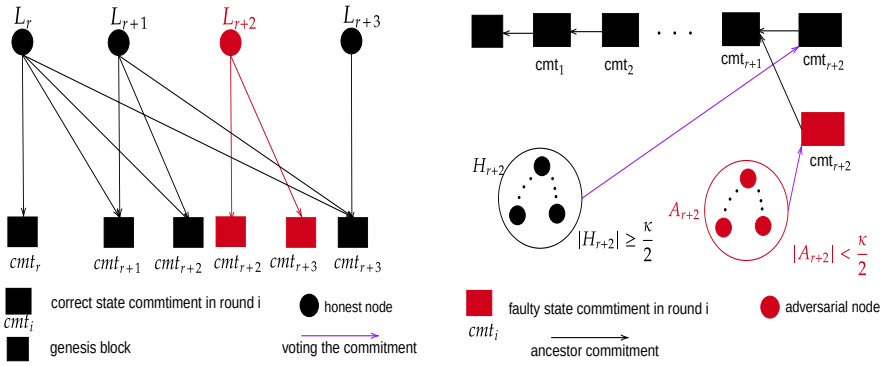

Fig. 1: Left figure illustrates the elected leaders' votes in the round interval $[r, r + 3]$, resulting in the fork in the chain of the proposed state commitments illustrated in on the right side. The set H_i (or A_i) consists of the votes of the honest (or adversarial) elected leaders in the interval $[r - \kappa + 3, r + 2]$.

Voting phase: For the first κ rounds, the *bootstrap committee* members form the respective valid state commitments. To update the state, they apply the transactions committed to the ledger for all these rounds starting from the

Algorithm 1: Horizontal Sampling: Node p_i with public key pk_i and secret key sk_i

$state(0) \leftarrow G$ // genesis block
$\texttt{threshold} \leftarrow \frac{\kappa}{2}, state\_com \leftarrow \{\}$
$r_{cur} \leftarrow 1$ // current round
/* threshold for the election process */
Procedure *Target*(r) **:**
  return $\frac{2^{|h|}}{n}\kappa$ **if** $r = 1$, **else return** $\frac{2^{|h|}}{n}$
/* verify the election proof with public key p_k in the round r */
Predicate *TimeToExecute*(p_k, r, u, π) **:**
  return $VerifyVRF_{p_k}(u, \pi, seed_r||r) \wedge u \leq Target(r)$
/* check whether *cmt* comes from a valid leader of round r_l that is responsible for executing in round r */
Predicate *AcceptCommitment*$(p_k, r_l, u, \pi, r, \sigma, cmt)$ **:**
  return $\neg(r_l > 1 \wedge r \leq \kappa) \wedge (r_l \leq r \leq r_l + \kappa - 1) \wedge Verify(\sigma, p_k, cmt||r) \wedge TimeToExecute(p_k, r_l, u, \pi)$
/* acquiring the state of round r */
Procedure *AcquireState*(r) **:**
  Wait until $\exists(cmt, r)$ $s.t. |state\_com[(cmt, r)]| \geq threshold$
  if $\texttt{state}(r) = null$ **then**
    request $\texttt{state}(r)$
    wait until receiving *state* s.t. $Compute_{cmt}(state) = cmt$
    $\texttt{state}(r) \leftarrow state$
/* compute and broadcast the signed state commitments for all the intermediate rounds within the interval $[r_l, r_l + \kappa - 1]$ */
Procedure *Execute*$(r_l, (u, \pi))$ **:**
  AcquireState$(r_l - 1)$ if $r_l > 1$
  for $r = r_l, ..., r_l + \kappa - 1$ **do**
    download $data(r)$ // data within the block with height r
    $\texttt{state}(r) \leftarrow apply(\texttt{state}(r-1), data(r))$
    continue if $r_l > 1 \wedge r \leq \kappa$ // only bootstrap committee votes
    $cmt \leftarrow Compute_{cmt}(\texttt{state}(r))$
    $\sigma \leftarrow Sign(cmt||r, pk_i, sk_i)$
    $state\_com[(r, cmt)].add((pk_i, r_l, u, \pi, \sigma))$
    Send $("state\ cmt", cmt, r_l, r, \sigma, u, \pi)$ to all nodes
/* Main loop, run leader election for each round */
while *True* **do**
  $(u, \pi) \leftarrow VRF_{sk}(seed_{r_{cur}}||r_{cur})$
  if $u \leq Target(r_{cur})$ **then**
    $Execute(r_{cur}, (u, \pi))$
  $r_{cur} \leftarrow r_{cur} + 1$
Upon *receiving*$("state\ cmt", cmt, r_l, r, \sigma, u, \pi)$ *from the node with public key* pk_j *for the first time for round* r_l **do :**
  if *AcceptCommitment*$(pk_j, r_l, u, \pi, r, \sigma, cmt)$ **then**
    $state\_com[(r, cmt)].add((pk_j, r_l, u, \pi, \sigma))$

genesis block. For every round, they compute and broadcast their signed state commitments along with their proof of election.

Now consider node p_i, an elected leader in some round $r \geq 2$ during the *voting phase* (Algorithm 1, Procedure Execute). First, p_i waits until witnessing a valid state commitment for the round $r-1$. After receiving the valid state, the leader acquires the corresponding state. If the state is not available from a previous execution, p_i requests it from all the nodes that have signed the commitment (Algorithm 1, lines 13-14) (more on data availability in section 6). Then, p_i downloads the data committed to the ledger for the intermediate rounds and applies it sequentially to obtain the state of the round $r+\kappa-1$. For each round, it constructs and signs the respective state commitment. Finally, p_i broadcasts the signed state commitments along with the proof of its election to the rest of the nodes. We note again that only *bootstrap committee* members vote for the first κ rounds (Algorithm 1 line 21). The rest of the nodes accept the received commitments only after confirming $p_i's$ signature and proof of election (Algorithm 1, lines 8-9).

Due to space limitation, we defer the correctness proof of the Horizontal Sampling algorithm to Appendix C.2.

4.2 Proof-of-stake settings

Now we extend the *Horizontal Sampling* protocol to the proof-of-stake setting. Participating nodes have accounts holding stake/coins, and we use W to denote the total amount of the stake in the system. New nodes can dynamically join the system, and we demonstrate bootstrapping in Algorithm 7. We assume up to $f \leq (1-\epsilon)\frac{W}{2}$ stake can be corrupted by a static adversary, where ϵ is some constant, and we choose the security parameter $\kappa = O(\log^2 W)$ for this section.

First, all nodes download the genesis block G which contains the initial stake distribution. The stake distribution can change over time. The full algorithms 5, 6 for the protocol are provided in the Appendix B.2. More specifically, we decompose the protocol into the following phases. In each round, every node participates in the *election phase* to check whether any of its coins is elected (Algorithm 5, Procedure Election). During the *voting phase*, nodes with at least one elected coin compute state commitments like in the permissioned protocol.

Tracking wealth: The stake distribution changes over time and the nodes do not necessarily acquire the execution state of each round. Hence the challenge for a node is to check whether the transactions it receives are successful or not. In our protocol, the node requests a *proof of payment* certificate from the payer, (see section 5), to verify that its state has changed as expected and therefore the transaction was successful.

Election phase: In each round, every node computes the VRF using its owned coins and the randomness seed coming from the dirty ledger to generate *proofs of election* for the elected coins. Similarly to the permissioned protocol, the PoS protocol elects a *bootstrap committee* for the first round, and elects on average one coin per round for every round $r > 1$. To keep the threshold of a valid state

commitment identical for every round, only the *bootstrap committee* members are voting for the first κ rounds, while for $r \geq 2$ the owner of an elected coin in round r can vote for every round in the interval $[max(r, \kappa + 1), r + \kappa - 1]$. A state commitment for the round r is valid if it is signed by the owners of at least $\frac{\kappa}{2}$ of the elected coins during the interval of rounds $[max(1, r - t + 1), r]$ or if it is the genesis block.

Proof of ownership: Since nodes track only their own stake, the elected nodes must prove that they own the elected coins. Hence, they provide inclusion proofs for their elected coins using the valid state commitment of the previous round, e.g., the commitment can be the Merkle root in a Merkle proof. We call these certificates *proofs of ownership* and the corresponding state commitment *parent commitment*. To be able to verify the *proofs of payment* in order to track its stake, and to compute the *proofs of ownership* in case of election, each node waits for the valid state commitment of the previous round before participating in the election phase.

Voting phase: The *voting phase* is similar to the permissioned protocol. First, the *bootstrap committee* members compute and broadcast their signed state commitments for every round $r \leq \kappa$ to form the respective valid state commitment. Then, every node with an elected coin in the round r, can start from the state corresponding to the valid state commitment of the round $r - 1$. Moreover, the node uses the valid state commitment to construct the *proof of ownership* for its elected coin. Finally, the elected node computes and broadcasts the signed state commitments for all the intermediate rounds along with the *proof of election* and the *proof of ownership* in the round r. To accept a signed state commitment, nodes first verify the related certificates. Especially for *proofs of ownership*, nodes wait until the *parent commitment* becomes valid.

Bootstrapping: Consider *Bob*, a node that wishes to join the network in the round r. We assume that *Bob* is connected to at least one honest executor. Bob has received from many nodes a data structure called *chain* that contains the state commitments signed by the elected nodes along with the respective certificates (signatures, proofs of election, and proofs of ownership) for each round.

Bob downloads the genesis block G first. For each *chain*, *Bob* applies Algorithm 7 to evaluate whether it is the correct one. For the first round, *Bob* verifies only the *proofs of election* of the *bootstrap committee* members since the initial stake distribution is contained in G. Then, for each vote up to round r, he verifies the signatures, the *proof of ownership*, and the *proof of election* of the elected nodes. When *Bob* receives the correct *chain*, it acquires the last valid state commitment in the *chain* and requests the corresponding state (Section 6).

5 Light clients protocol

Once we have a system where executors can verify that a payment has been made, it is simple to transform it to the first non-interactive, asynchronous light-client for dirty ledgers. In this section, we demonstrate how a light client can learn the

state of the system. First, we discuss how a light client can acquire and verify a *state proof*. Then, we use *state proofs* as a building block to prove that a change in the state occurred.

Assumptions: Each light client has access to the random seed for each round through the dirty ledger, in order to verify the leader election. In addition, each light client is connected to at least one honest executor. An executor uses a gossip protocol to obtain information necessary to react to a light client's requests, such as the state that corresponds to a valid state commitment.

Bootstrapping: Assume that the height of the dirty ledger equals h and a light client lc_i bootstraps in the round $r \leq h$. First, we illustrate how lc_i can verify a *state proof*. A validity proof of the state commitment corresponding to the state S constitutes the *state proof* π_S. The light client then chooses how to connect to the network. One option is to receive the corresponding state and derive the desired information after downloading and applying the data committed to the ledger on its own. Otherwise, lc_i can reconnect to the network whenever it needs a *proof of payment* certificate.

State Proof - Permissioned settings: To bootstrap in the round r, lc_i waits to receive a valid state commitment for some round greater than or equal to the round r. In the *deterministic protocol*, lc_i verifies that a state commitment is signed by at least $f + 1$ nodes using the Predicate *AcceptStateProof* in Algorithm 4. In the *Horizontal Sampling protocol*, a valid state commitment in a round r' is voted by at least $\frac{\kappa}{2}$ elected leaders in the interval $[max(1, r' - \kappa + 1), r']$. Each leader's vote includes their signature and proof of election. Lc_i verifies this using the Predicate *AcceptStateProof* in Algorithm 2.

Algorithm 2: Light Client protocol - Horizontal Sampling

```
1 threshold ← κ/2
  /* check whether there are at least κ/2 signatures for cmt by
     leaders of rounds [r − t + 1, r] in Σ                          */
2 Predicate AcceptStateProof(cmt, r, Σ)) :
3 |  Remove duplicates in Σ
4 |  return |AcceptCommitment(p_k, r_l, u, π, r, σ, cmt) : (p_k, r_l, u, π, σ) ∈ Σ| ≥
  |    threshold
5 Predicate PaymentProof(cmt, r, Σ, π_inclusion_proof) :
6 |  return AcceptStateProof(cmt, r, Σ)∧ state change occurred according to
  |    the π_inclusion_proof
```

State Proof - Proof-of-stake settings: The light client bootstraps using Algorithm 7. In a nutshell, for each round, lc_i requires and verifies the signatures, the *proof of ownership*, and the *proof of election* coming from the owners of the elected coins that have voted for the valid state commitments.

Proofs of payment: We now demonstrate how to provide certificates for successful transactions. Consider Alice and Bob, two light clients using our system.

Bob wishes to purchase a product from Alice, triggering a transaction that will be logged in the dirty ledger. Alice needs a proof that the payment is successful before providing the merchandise to Bob.

Assume that the transaction of Bob paying Alice is committed at round r. The certificate with which Bob proves that Alice's state is changed in round r is called *proof of payment*. More specifically, the certificate constitutes of a valid state commitment for any round greater than r and a short inclusion proof (e.g. a Merkle proof) indicating Alice's new state. Alice uses the Predicate Payment-Proof in Algorithm 2 to verify first the validity of the state commitment and then the inclusion proof, using the valid state commitment, to extract her new state. If the transaction is successful, Alice's state is changed during this interval.

6 Data availability

In this section, we discuss what data executors store locally to support the proposed protocols.

State availability: Nodes responsible for executing in a particular round need to acquire first the state of a previous round. It is also required by the *Proof of payment* and bootstrapping in the proof-of-stake settings (section 4.2).

We let the executors store every state they executed. In all of the proposed protocols, each valid state commitment is signed by at least one honest node which has stored the state with overwhelming probability. An executor requests the state that corresponds to a *valid* state commitment from all the nodes that have signed the respective state commitment. The honest node that has signed the state commitment will eventually provide it to the executor. The executor will verify that the state indeed corresponds to the valid state commitment.

Certificate availability: To support bootstrapping protocols, executors store the certificates related to the valid state commitments. In the deterministic protocol, they only store the signed state commitments (Algorithm 3 lines: 10, 17). In the Horizontal Sampling protocol, executors store the signed valid state commitments along with the leaders' proofs of election (Algorithm 1 lines: 24, 33), and in the proof-of-stake settings, they additionally keep the *proofs of ownership* of the elected coins (Algorithm 5 line 20, Algorithm 6 line 18).

7 Conclusion

In this paper, we demonstrated how *Horizontal Sampling* converts an atomic broadcast (BAB) solution to an SMR (or how to execute the state on top of a dirty ledger). To this end, *Horizontal Sampling* is an efficient distributed execution protocol that consists of two phases. First, there is a voting phase where a constant number of nodes are selected. Second, the selected nodes execute and propose state commitments for the following polylog rounds during the voting phase. *Horizontal Sampling* is a censorship-resistant solution that does not violate the network assumptions of the underlying ledger. Lastly, we illustrated

how to leverage *Horizontal Sampling* for defining non-interactive light clients that learn the state of the system.

Acknowledgements

Eleftherios Kokoris-Kogias is partially supported by Austrian Science Fund (FWF) grant No:F8512-N.

References

1. Bringing the World to Ethereum | Polygon. `https://polygon.technology`.
2. Fuel Network. `https://fuel.network`.
3. Neon Team. Neon evm. `https://neon-labs.org/Neon_EVM.pdf`. Accessed: 3-8-2022.
4. Optimism. `https://www.optimism.io`.
5. The DiemBFT Team. State machine replication in the diem blockchain, 2021. `https://developers.diem.com/docs/technical-papers/state-machine-replication-paper`.
6. Optimistic rollups: How they work and why they matter. `https://medium.com/stakefish/optimistic-rollups-how-they-work-and-why-they-matter-3f677a504fcf`, 2021.
7. Rollups, data availability layers & modular blockchains: introductory meta post. `https://polynya.medium.com/rollups-data-availability-layers-modular-blockchains-introductory-meta-post-5a1e7a60119d`, 2021.
8. What is a zero-knowledge (zk) rollup? `https://zebpay.com/blog/what-is-a-zero-knowledge-rollup-zk`, 2022.
9. Mustafa Al-Bassam. Lazyledger: A distributed data availability ledger with client-side smart contracts. *arXiv preprint arXiv:1905.09274*, 2019.
10. Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities. *arXiv preprint arXiv:1809.09044*, 160, 2018.
11. Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.
12. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 181–197. Springer, 2008.
13. Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.
14. Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.
15. Shir Cohen, Guy Goren, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Proof of availability & retrieval in a modular blockchain architecture. *Cryptology ePrint Archive*, 2022.

16. George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.
17. Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys (CSUR)*, 36(4):372–421, 2004.
18. David Eppstein, Michael T Goodrich, Frank Uyeda, and George Varghese. What's the difference? efficient set reconciliation without prior context. *ACM SIGCOMM Computer Communication Review*, 41(4):218–229, 2011.
19. Jose M Faleiro and Daniel J Abadi. Rethinking serializable multiversion concurrency control. *arXiv preprint arXiv:1412.2324*, 2014.
20. Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 296–315. Springer, 2022.
21. Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Yu Xia, Runtian Zhou, and Dahlia Malkhi. Block-stm: Scaling blockchain execution by turning ordering curse to a performance blessing. *arXiv preprint arXiv:2203.06871*, 2022.
22. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.
23. Neil Giridharan, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Bullshark: Dag bft protocols made practical. *arXiv preprint arXiv:2201.05677*, 2022.
24. Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 165–175, 2021.
25. Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. USENIX Association, 2016.
26. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pages 203–226. 2019.
27. Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1041–1053, 2019.
28. Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.
29. Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.
30. A Pinar Ozisik, Gavin Andresen, Brian N Levine, Darren Tapp, George Bissias, and Sunny Katkuri. Graphene: efficient interactive set reconciliation applied to blockchain propagation. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 303–317. 2019.
31. Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, and Marko Vukolić. Mir-bft: High-throughput robust bft for decentralized networks. *arXiv preprint arXiv:1906.05552*, 2019.

32. Ertem Nusret Tas, Dionysis Zindros, Lei Yang, and David Tse. Light clients for lazy blockchains. *arXiv preprint arXiv:2203.15968*, 2022.
33. Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

A Summary of Terminologies

We summarize the terminologies used in this paper in Table 1.

Notation	Description
n	total number of nodes in the permissioned settings
f	number of adversarial nodes (or coins held by adversarial nodes) in the permissioned settings (or in PoS)
W	total amount of stake in PoS
proof of election	proofs coming from the VRF computation of the elected nodes in the probabilistic protocols
proof of ownership with parent commitment *cmt*	inclusion proof with hash header *cmt* demonstrating that a node p_i owns a particular coin in PoS

Table 1: Terminologies

B Protocols

B.1 Deterministic Protocol

The deterministic protocol is described in Algorithm 3. Every node is responsible for executing in each round. More specifically, every node downloads the data committed to the ledger starting from the genesis block and applies it sequentially via the execution machine M, to acquire the execution state of each round. In every round, the nodes compute and sign the respective state commitment. Then, they broadcast the state commitment to the rest of the nodes. This process is illustrated in Algorithm 3 (Procedure Execute lines: 5-11). The remaining nodes accept signed state commitments once they have verified the sender's signature (Algorithm 3 Predicate AcceptCommitment, lines: 3-4). A state commitment is valid when it is signed by at least $f + 1$ nodes.

We present the light client protocol for the deterministic solution in Algorithm 4. A light client lc_i uses the Predicate *AcceptStateProof* to verify a *state proof* it has received for the round r, which is a proof of validity of the respective commitment *cmt*. More specifically, lc_i verifies that there are at least $f + 1$ signatures related to *cmt* coming from the executors. Furthermore, to verify a *proof*

Algorithm 3: Deterministic execution protocol: Node p_i with public key pk_i and secret key sk_i

$state(0) \leftarrow G, r_{cur} \leftarrow 1$
$threshold \leftarrow f + 1, state\_com \leftarrow \{\}$
`/* accept the state commitment` cmt `from the node with public key` pk_j `if the signature` σ `is valid */`
Predicate $AcceptCommitment(cmt, r, \sigma, pk_j)$ **:**
    return $Verify(\sigma, pk_j, cmt||r)$
`/* executing for round r: applying the data committed to the ledger to update the state, computing and broadcasting the state commitment to everyone */`
Procedure $Execute(r)$ **:**
    download $data(r)$
    $\texttt{state}(r) \leftarrow apply(\texttt{state}(r-1), data(r))$
    $cmt \leftarrow Compute_{cmt}(\texttt{state}(r))$
    $\sigma \leftarrow Sign(cmt||r, pk_i, sk_i)$
    $state\_com[(r, cmt)].add((\sigma, pk_i))$
    Send ("state commitment", cmt, r, σ) to all nodes
`/* Main loop. */`
while *True* **do**
    $Execute(r_{cur})$
    $r_{cur} \leftarrow r_{cur} + 1$
Upon *receiving("state commitment", cmt, r,* σ*) from the node with public key* pk_j *for the first time in the round* r **do :**
    if $AcceptCommitment(cmt, r, \sigma, pk_j)$ **then**
        $state\_com[(r, cmt)].add((\sigma, pk_j))$

of payment certificate, which constitutes of a *state proof* for the round r and an inclusion proof demonstrating that a specific change in the state has happened, lc_i employs the Predicate *PaymentProof*.

Algorithm 4: Light Client protocol - Deterministic Protocol

```
1 threshold ← f + 1
  /* check whether there are at least f + 1 signatures for cmt in Σ
     */
2 Predicate AcceptStateProof(cmt, r, Σ) :
3     Remove duplicates in Σ
4     return |AcceptCommitment(cmt, r, σ, p_j) : (σ, p_j) ∈ Σ| ≥ threshold
5 Predicate PaymentProof(cmt, r, Σ, π_inclusion_proof) :
6     return AcceptStateProof(cmt, r, Σ)∧ state change has occurred according
        to π_inclusion_proof
```

B.2 Proof of Stake Protocol

Algorithm 5 and 6 presents our Proof-of-Stake protocols. Algorithm 7 presents the bootstrapping protocol for a new node. The protocol consists of the *election phase* and the *voting phase*. The *election phase* outputs elected nodes per round. In the *voting phase*, the elected nodes propose state commitments in order to form the respective valid state commitments. We choose $\kappa = O(\log^2 W)$, where W is the total stake of the system.

Tracking wealth: Each node p_i tracks only its stake. The node p_i knows its stake when it bootstraps in the system. Furthermore, p_i knows whether a transaction it triggers is valid, to remove the corresponding coins from its stake. The challenge is determining whether a transaction it receives is successful, to include the respective coins in its stake. To this end, the node requests from the payer the *proof of payment* certificate we discuss in Section 5, which consists of a valid state commitment and an inclusion proof so that p_i can verify that its state has changed.

Election phase: Each node computes the VRF for all of its coins in every round (Algorithm 5, Procedure Election). When the occurring VRF value for a coin is less than $X_r = \begin{cases} \frac{2^{|h|}}{n}\kappa, \text{ if } r = 1 \\ \frac{2^{|h|}}{n}, \quad \text{if } r > 1 \end{cases}$ the coin is elected in the round r. That results in a *bootstrap committee* consisting of κ coins for the first round and electing on average one coin per round for every round $r > 1$.

To prove the election of its coin in the round r, a node p_i, must provide the *proof of election*, i.e., the proof coming from the VRF, and the *proof of ownership* which is an inclusion proof demonstrating that p_i owns the coin. The header of the inclusion proof is the valid state commitment of the previous round, which we call *parent commitment*.

To track its stake, by receiving *proofs of payment* when needed, and to provide *proofs of ownership* in case it is elected, each node waits until there is a valid state commitment for the previous round before entering the *election process* (Algorithm 6 line 21).

Valid state commitment: The *bootstrap committee* members vote for any round $\in [1, \kappa]$, while every node with an elected coin in a round $r \geq 2$ can vote for the rounds $[max(r, \kappa + 1), r + \kappa - 1]$. A state commitment in the round r is valid if it is signed by the owners of at least $\frac{\kappa}{2}$ coins elected in the interval $[max(1, r - \kappa + 1), r]$ or if it is the genesis block.

Voting phase: In the first k rounds the *bootstrap committee* members form the respective valid state commitments according to Lemma 12, by applying the data committed to the ledger starting from the genesis block and broadcasting the respective state commitments along with their *proof of election*.

An elected leader p_i in any round $r \geq 2$, enters the Procedure Execute in the Algorithm 6. The node p_i starts from the state of the previous round which it acquires by requesting it from the nodes that have signed the valid state commitment of the previous as we discuss in Section 6 (Algorithm 6 line 7). Furthermore, p_i uses the valid state commitment as *parent commitment* to construct the *proof of ownership* of its elected coins (Algorithm 6 line 22). Then, p_i computes and broadcasts to every node the signed state commitments up to the round $r + \kappa - 1$ along with the *proof of election* and the *proof of ownership*.

Nodes accept the state commitments proposed by the elected nodes only after verifying the *proof of election* and the *proof of ownership* certificates (Algorithm 6 lines 27-30). When receiving an inclusion proof with a *parent commitment* which is not valid yet, they store all the certificates in a local data structure (Algorithm 5 lines 13-15). When a state commitment becomes valid in the view of an honest node p_i, the node takes into account all the votes for which it is a *parent commitment* (Algorithm 5 lines: 21-24).

Bootstrapping: Now, consider the user u_i, which can be either a light client or an executor, bootstrapping in the system in a round $r_e \geq 1$. The user is connected to a network of executors. At least one of those executors is honest. The nodes to which u_i is connected send him a data structure called *chain* including the votes of the elected nodes for the valid state commitments along with the related certificates (signatures, *proof of election*, and *proof of ownership*) for every round.

For every received *chain*, u_i applies the Procedure Bootstrap described in Algorithm 7, until there is a correct *chain* reaching to a round $r \geq r_e$. For every round $r \geq 1$, u_i verifies the signature (Algorithm 7 line 5), and the *proofs of election and ownership* (Algorithm 7 line 6) of the elected leaders. We remind that in the first round, the elected nodes must provide only the *proofs of election* since the initial stake distribution is known to every node through the genesis block. When there are at least $\frac{\kappa}{2}$ valid votes for a state commitment *cmt*, u_i proceeds to the next round (Algorithm 7 lines: 9-11). If there is a round for which there are not enough valid votes to form the corresponding valid state commitment, the Procedure returns *null* (Algorithm 7 line 12). Otherwise, u_i

continues that process until the last valid state commitment in the chain. As an example, in figure 2 we present the correct chain up to the round h. Then, u_i can receive the corresponding state, requesting it from the nodes that have signed the state commitment as we discuss in Section 6.

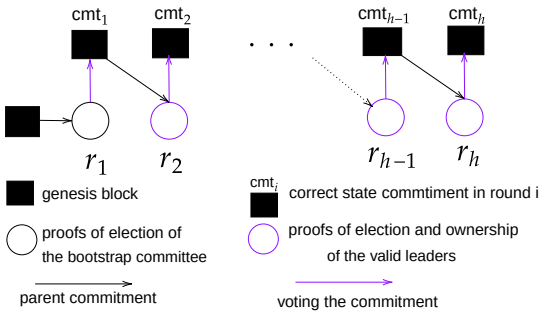

Fig. 2: A correct *chain* up to the round h. Only nodes elected during the interval of rounds $[i, i + \kappa - 1]$ are allowed to vote for a commitment in the round i.

.

Algorithm 5: Functions for the election phase: node p_i with pair of keys (pk_i, sk_i)

```
   /* threshold for the election process                                   */
 1 Procedure Target(r) :
 2     return (2^|h| / n) κ if r = 1, else return 2^|h| / n
   /* p_i evaluates whether a subset of its coins is elected in the
      round r, returning a list with the respective certificates          */
 3 Procedure Election(r, parent_cmt) :
 4     elected_coins ← {}
 5     for each coin that p_i owns in the round r do
 6         (v, π) ← VRF_{sk_i}(coin||seed_r)
 7         if v ≤ Target(r) then
 8             π_m ← null if r = 1, else π_m ← InclusionProof(parent_cmt, p_i, coin)
                elected_coins.add((pk_i, r, coin, v, π, π_m, parent_cmt))
 9     return elected_coins
   /* verify the proofs of election and ownership coming from a valid
      leader that can vote for the round r                                */
10 Procedure VerifyVotes(r, votes) :
11     valid_votes ← {}
12     for vote ← (pk_j, r_l, coin, u, π, π_m, parent_cmt) ∈ votes do
13         if |state_com[(r_l − 1, parent_cmt)]| < κ/2 then
14             tmp_certificates[(parent_cmt, r_l − 1)].add(cmt_r, r, votes)
15             continue // parent_cmt is not valid yet
16         if (r_l = 1 ∧ pk_j owns coin ∧ r ≤
             κ ∧ VerifyVRF_{pk_j}(u, π, coin||seed_1) ∧ u ≤ Target(1)) ∨ (r_l > 1 ∧ r >
             κ ∧ r_l ≤ r ≤ r_l + κ − 1 ∧ Verify_{InclusionProof}(π_m, parent_cmt)) ∧
             VerifyVRF_{pk_j}(u, π, coin||seed_{r_l}) ∧ u ≤ Target(r_l)) then
17             valid_votes.add(vote)
18     return valid_votes
   /* include the valid votes for cmt; if cmt becomes valid, update
      the votes for which cmt is the parent commitment                    */
19 Procedure UpdateVotes(votes, cmt, r) :
20     ∀vote ∈ votes : state_com[(cmt, r)].add(vote)
21     if |state_com[(cmt, r)]| ≥ threshold then
22         for (cmt_{r'}, r', certificates) ∈ tmp_certificates[(cmt, r)] do
23             induced_votes ← VerifyVotes(r', certificates)
24             UpdateVotes(induced_votes, cmt_{r'}, r') if induced_votes ≠ null
```

Algorithm 6: Functions for the voting phase: node p_i with pair of keys (pk_i, sk_i)

$state(0) \leftarrow G, state\_com \leftarrow \{\}, tmp\_certificates \leftarrow \{\}, elected\_coins \leftarrow \{\}, succesful\_rounds \leftarrow \{\}$
$r_{cur} \leftarrow 1$, `threshold` $\leftarrow \frac{\kappa}{2}$
$parent_{cmt} \leftarrow G$ `// valid state commitment of round` $r-1$
`/* acquire the state of the round` r `*/`
Procedure *AcquireState(r)* **:**
  if $state(r) = null$ **then**
    $cmt \leftarrow cmt'$ *s.t.* $state\_com[(cmt', r)] \geq threshold$
    request `state`(r)
    wait until receiving *state* s.t. $Compute_{cmt}(state) = cmt$
    `state`$(r) \leftarrow state$
`/* node` p_i `computes and broadcasts its state commitments for the rounds` $[r_l, r_l + \kappa - 1]$ `to everyone along with the proofs of election and ownership of its coins in the round` r_l `*/`
Procedure *Execute*$(r_l, elected\_coins)$ **:**
  AcquireState$(r_l - 1)$ if $r_l > 1$
  for $r = r_l, ..., r_l + \kappa - 1$ **do**
    download $data(r)$
    `state`$(r) \leftarrow apply($`state`$(r-1), data(r))$
    continue if $r_l > 1 \wedge r \leq \kappa$ `// only the bootstrap committee votes`
    $cmt \leftarrow Compute_{cmt}($`state`$(r))$
    $\sigma \leftarrow Sign(cmt||r, pk_i, sk_i)$
    $\forall vote \in elected\_coins : state\_com[(r, cmt)].add(vote)$
    $Send($`"state cmt"`$, r_l, r, cmt_r, \sigma, elected\_coins)$ to all nodes
`/* Main loop.` `*/`
while *True* **do**
  Wait until $\exists(cmt, r_{cur} - 1)$ *s.t.*$|state\_com[(cmt, r_{cur} - 1)]| \geq threshold$ if $r_{cur} > 1$
  $parent_{cmt} \leftarrow cmt$ if $r_{cur} > 1$ `// else` $parent_{cmt} \leftarrow G$
  $elected\_coins \leftarrow Election(r_{cur}, parent_{cmt})$
  if $elected\_coins \neq null$ **then**
    Execute$(r_{cur}, elected_coins)$
  $r_{cur} \leftarrow r_{cur} + 1$
`/* Receiving` cmt `for round` r`, the respective certificates (proofs of election and ownership), and the signature` σ`.` `*/`
Upon *receiving(*`"state cmt"`*,* $r_l, r, cmt_r, \sigma, certificates$*) from node with public key* pk_j *for the first time for the round* r_l **do :**
  return if $Verify(\sigma, pk_j, cmt_r||r) = false$
  $valid\_votes \leftarrow VerifyVotes(r, certificates)$
  $UpdateVotes(valid\_votes, cmt_r, r)$ **if** $valid\_votes \neq null$

Algorithm 7: Bootstrapping: user U_i

```
/* U_i aims to bootstrap in a round greater or equal than round r_e.
   */
Procedure Bootstrap(chain, r_e) :
    r ← 1, state_cmt ← {}, valid_state_cmt ← {}
    while chain[r] ≠ null do
        for (node, cmt, σ, votes) ∈ chain[r] do
            continue if Verify(σ, node, cmt||r) = false
            valid_votes ← VerifyVotes(r, votes)
            if valid_votes ≠ null then
                ∀vote ∈ valid_votes : state_cmt[(cmt, r)].add(vote)
                if |state_cmt[(cmt, r)]| ≥ κ/2 then
                    valid_state_cmt[r] ← cmt
                    break
        return null if ∄cmt s.t. |state_cmt[(cmt, r)]| ≥ κ/2
        r ← r + 1
    return valid_state_cmt if r ≥ r_e
    return null
```

C Proofs

Structure of the Proofs section: First, for each protocol, we will formulate helpful lemmas to prove safety and liveness of the execution protocols. For safety, we must argue first that there are some state commitments that the nodes can trust. The objective of our protocols is that only a correct state commitment can be valid. For liveness, we must ensure that as long as the ledger grows the nodes will be generating valid state commitments.

Then we will prove the properties *E-Safety*, *E-Liveness* for all the protocols together using the aforementioned Lemmas as a foundation. Additionally, we prove the properties *LC-Safety*, *LC-Liveness* for the light client execution layer protocol together using these Lemmas. In the probabilistic protocols, namely the Horizontal Sampling and the PoS protocol, the properties hold except with negligible probability in the security parameter.

C.1 Deterministic Protocol

Definition 1. *A state commitment is considered to be valid if and only if either it is signed by at least $f + 1$ nodes or it is the genesis block.*

Definition 2. Successful round*: A round r is* successful *if and only if a valid state commitment visible to every honest node exists for this round.*

Lemma 1. *Every valid state commitment is correct.*

Proof. We will prove the Lemma by reaching a contradiction. Assume that the state commitment cmt_r corresponding to the round $r \geq 1$ is valid but not correct. Since cmt_r is valid, it is signed by at least $f+1$ nodes and therefore from at least an honest node. We call the existing honest node p_i. Node p_i has started updating the system from the genesis block, applying the transactions committed to the ledger via the execution engine M. Thus, p_i must have obtained the correct state corresponding to the round r. So, we conclude that p_i must have misbehaved and thus we reach a contradiction. □

Lemma 2. *State availability: The state corresponding to a valid state commitment will be available to any user u_i.*

Proof: Consider the valid state commitment cmt_r for the round r. The commitment cmt_r is signed by at least $f+1$ nodes and therefore from at least an honest one, which we call p_i. Node p_i has stored the corresponding state. A user u_i requests the state from all the nodes that have signed the commitment cmt. Since p_j is honest, it will eventually send the corresponding state to u_i. □

Lemma 3. *Every round r corresponding to a block committed to the dirty ledger will eventually be* successful.

Proof: Consider the round $r \geq 1$. There are at least $f+1$ honest nodes executing in each round (Algorithm 3 Procedure Execute). All these honest nodes download the data committed to the dirty ledger up to round r and apply only the valid transactions to obtain the updated state. For each one of these rounds, the honest nodes compute the respective state commitment. Since the commitment scheme is deterministic the $f+1$ honest nodes output the same result forming a valid commitment for all the rounds up to the round r. □

C.2 Horizontal Sampling protocol

Permissioned Protocol In the following proofs: $\kappa = O(log^2 n)$.

Lemma 4. *Consider a number of n nodes with $f \leq (1-\epsilon)\frac{n}{2}$ Byzantine nodes and $\geq (1+\epsilon)\frac{n}{2}$ honest nodes, where $0 < \epsilon < \frac{1}{2}$. Let κ denote the security parameter. A sampling is taking place, where each node is chosen independently with a probability of $\frac{\kappa}{n}$. The following arguments hold except a negligible probability in κ:*
1. There will be at least $(1+\frac{\epsilon}{2})\frac{\kappa}{2}$ honest nodes chosen.
2. There will be at most $(1-\epsilon)\frac{\kappa}{2}$ Byzantine nodes chosen.

Proof. 1. Let H denote the number of selected honest nodes. Since each node is chosen with probability $\frac{\kappa}{n}$, we have $E[H] \geq (1+\epsilon)\frac{n}{2} \cdot \frac{\kappa}{n} = (1+\epsilon)\frac{\kappa}{2}$. From Chernoff bounds, we have $P[H \leq (1-\delta)E[H]] \leq e^{-\frac{\delta^2 E[H]}{2}}$ where $0 \leq \delta \leq 1$. Since $E[H] \geq (1+\epsilon)\frac{\kappa}{2}$, we have $P[H \leq (1-\delta)(1+\epsilon)\frac{\kappa}{2}] \leq P[H \leq (1-\delta)E[H]] \leq e^{-\frac{\delta^2 E[H]}{2}}$. Choosing $\delta = \frac{\epsilon}{2(1+\epsilon)}$, we have $P[H \leq (1+\frac{\epsilon}{2})\frac{\kappa}{2}] \leq e^{-\frac{\epsilon^2 \kappa}{16(1+\epsilon)}}$.

2. Let F denote the number of selected Byzantine nodes. Since each node is chosen with probability $\frac{\kappa}{n}$, we have $E[F] \leq (1-\epsilon)\frac{\kappa}{2}$. Again using Chernoff bounds, $P[F \geq (1+\delta)E[F]] \leq e^{-\frac{\delta^2 E[F]}{2+\delta}}, 0 < \delta < 1$, and the fact that $P[F \geq (1+\delta)(1-\epsilon)\frac{\kappa}{2}] \leq P[F \geq (1+\delta)E[F]]$, and choosing $\delta = \frac{\epsilon}{1-\epsilon}$ leads to $P[F \geq \frac{\kappa}{2}] \leq e^{-\frac{\epsilon^2\kappa}{4-2\epsilon}}$.

Definition 3. *A majority commitment corresponding to some round r is a state commitment voted by at least $\frac{\kappa}{2}$ nodes among the ones elected during the interval of rounds $[max(1, r-\kappa+1), r]$.*

Definition 4. *A state commitment is considered to be valid if and only if either it is a majority commitment or it is the genesis block.*

Lemma 5. *The* bootstrap committee *consists of at least $\frac{\kappa}{2}$ honest and at most $\frac{\kappa}{2}-1$ adversarial nodes except with negligible probability in n.*

Proof: In the first round each node is elected with a probability of $p = \frac{\kappa}{n}$ after computing the VRF. Since the majority of the nodes are honest and considering the VRF computations as independent Bernoulli trials, the argument holds by applying Lemma 4. Since $\kappa = O(log^2 n)$ the probabilities are overwhelming in n. □

Lemma 6. *In any interval of rounds $[r, r+\kappa-1], r \geq 1$, there will be at least $\frac{\kappa}{2}$ honest and at most $\frac{\kappa}{2}-1$ adversarial, not necessarily distinct, elected nodes with a high probability in n.*

Proof: Case 1, $r = 1$: Only votes coming from *bootstrap committee* members' are taken into account for this interval. Following from Lemma 5, there are at least $\frac{\kappa}{2}$ honest and at most $\frac{\kappa}{2}-1$ adversarial *bootstrap committee* members except with negligible probability in n.
Case 2, $r \geq 2$: Without loss of generality, consider the interval of rounds $[r, r+\kappa-1]$. In every round within that interval, each node computes the VRF for the election phase (Algorithm 1 line 27), which is a Bernoulli trial with a probability of success $p = \frac{1}{n}$. Let H (or F) denote the number of honest (or adversarial) elected nodes within that interval. Then $E[H] \geq \sum_{i=r}^{r+\kappa-1}(1+\epsilon)\frac{n}{2}\frac{1}{n} = (1+\epsilon)\frac{k}{2}$. Similarly, $E[F] \leq \sum_{i=r}^{r+\kappa-1}(1-\epsilon)\frac{n}{2}\frac{1}{n} = (1-\epsilon)\frac{k}{2}$. Since computing the VRF results in independent Bernoulli trials, we can apply Chernoff bounds for $E[H]$ and $E[F]$. With the same analysis done in Lemma 4: $P[H \leq (1-\delta)E[H]] \leq e^{-\frac{\epsilon^2\kappa}{16(1+\epsilon)}}$ and $P[F \geq \frac{\kappa}{2}] \leq e^{-\frac{\epsilon^2\kappa}{4-2\epsilon}}$. Since $\kappa = O(\log^2 n)$, we have these probabilities negligible in n. □

Lemma 7. *Every valid state commitment is correct except with negligible probability in n.*

Proof. We will prove this theorem by induction on the rounds for which a valid state commitment exists.

Base step: Genesis block is the valid and correct state commitment for round 0. Induction step: Consider the round $r \geq 1$ for which the valid state commitment cmt_r exists. Assume that the theorem holds for every round $r' < r$. First, if $1 \leq r \leq \kappa$, there are at most $\frac{\kappa}{2} - 1$ adversarial *bootstrap committee* members except with negligible probability in n according to Lemma 5. If $r \geq \kappa + 1$, the elected adversarial nodes in the interval $[r - \kappa + 1, r]$ are at most $\frac{\kappa}{2} - 1$ with overwhelming probability in n according to Lemma 6. So, for any $r \geq 1$, cmt_r must be signed by at least one honest node, which we call p_i. P_i was a leader in some round $r_l, max(1, r - \kappa + 1) \leq r_l \leq r$. Node p_i started the execution from the state corresponding to a valid state commitment $cmt_{r_{l-1}}$ of the round r_{l-1}. It verified that the state corresponds to $cmt_{r_{l-1}}$ by computing the respective commitment. According to the induction assumption, $cmt_{r_{l-1}}$ is correct. Finally, p_i applied the valid transactions committed to the ledger for all the intermediate rounds via the execution engine M. Therefore, since cmt_r is signed by p_i it must be correct. □

Lemma 8. *State availability: Consider a round for which a valid state commitment exists. Each node that requests the corresponding state will eventually receive it except with negligible probability in n.*

Proof: Consider node p_i that needs the state corresponding to a valid state commitment of a round r. P_i requests the state from all the nodes that have signed the state commitment. Each valid state commitment is signed by at least one honest node except with negligible probability in n. Otherwise, the adversarial nodes could form a valid state commitment on their own contradicting Lemma 7. Let us call p_j an existing honest node that have signed the valid state commitment. Node p_j has stored the corresponding state and will eventually deliver it to p_i. □

Definition 5. *We call the interval of rounds* $[r - \kappa + 1, r]$ well formed *if and only if every honest elected node for a round* $r_l \in [r - \kappa + 1, r]$ *has witnessed a valid state commitment for the round* $r_l - 1$.

Lemma 9. *Consider a* well formed *interval of rounds* $[r - \kappa + 1, r]$. *Round r will eventually become successful except with negligible probability in n.*

Proof: Consider a round $r \geq \kappa$ such that the interval $[r - \kappa + 1, r]$ is well formed. So, each honest node elected in a round $r_l \in [r - \kappa + 1, r]$ has witnessed a valid state commitment for the round $r_l - 1$. Following from Lemma 6 there are at least $\frac{\kappa}{2}$ honest nodes elected in that interval with overwhelming probability, constituting the subset H. Every node in H will acquire the state corresponding to the valid state commitment it has seen according to Lemma 8. The valid state commitment, and therefore the state, is correct according to Lemma 7. Nodes in H will next apply the valid transactions committed to the ledger and compute the state commitments up to the round r (Algorithm 1 lines: 18-22). The leaders will then broadcast the signed state commitment along with their proof of election to every node (Algorithm 1 line 25). Every node will verify

their signatures and their proofs of election (Algorithm 1 line 32). Since the commitment scheme is deterministic and the nodes in H start from a correct state, they will output the same result forming a valid state commitment for the round r. □

Lemma 10. *Every round r corresponding to a block committed to the dirty ledger will eventually be* successful *except with negligible probability in n.*

Proof: The interval of rounds $[2, \kappa+1]$ will eventually become well formed, since there are at least $\frac{\kappa}{2}$ honest *bootstrap committee* members, according to Lemma 5, that will compute and broadcast to every node the state commitments for every round $r \in [1, \kappa]$ along their *proofs of election*. Since the commitment scheme is deterministic their outputs will be identical, forming the respective valid state commitments for these rounds.

Applying Lemma 9 for the *well formed* interval $[r_c, r_c + \kappa - 1]$ results in a valid state commitment $cmt_{r_c+\kappa-1}$ visible to every honest node for the round $r_c + \kappa - 1$. Therefore, the interval $[r_c + 1, r_c + \kappa]$ becomes *well formed*. By applying this argument recursively for $r_c \geq 2$, every round r will eventually become *successful*. □

Lemma 11. *Efficiency guarantees: The expected number of rounds for which each node executes equals $\frac{h\kappa}{n}$, where h is the number of the blocks committed to the dirty ledger and $\kappa = O(log^2 n)$.*

Proof: In the first round, nodes are elected with a probability of $p = \frac{\kappa}{n}$ and execute for κ rounds. For the rounds $r \geq 2$, consider the random variable N depicting the number of times that the honest node p_i is elected in $h-1$ rounds. N follows the binomial distribution $N \sim B(h-1, \frac{1}{n})$. Each time that p_i is elected, it executes for κ rounds. Thus, the expected value of the random variable X capturing the number of rounds for which p_i executes in h rounds is equal to $E[X] = \frac{\kappa^2}{n} + \kappa E[N] = \frac{(h+\kappa-1)\kappa}{n}$. □

Proof-of-stake protocol In the following proofs: $\kappa = O(log^2 W)$.

Definition 6. *We say that an interval of rounds $[r, r+\kappa-1]$ is* honest prevalent *if at least $\frac{\kappa}{2}$ elected coins within that interval are held by honest nodes and at most $\frac{\kappa}{2} - 1$ elected coins by adversarial nodes.*

Definition 7. *Consider the node p_i. We call and denote by $C_{p_i,r}$ the stake of p_i in the round r according to its view as* relative stake*, and the stake of p_i according to the correct state S^*_r, denoted by $C^*_{p_i,r}$, as* actual stake*.*

Definition 8. *A* majority commitment *corresponding to some round r is a state commitment voted by the owners of at least $\frac{\kappa}{2}$ of the elected coins during the interval of rounds $[max(1, r-t+1), r]$.*

Definition 9. *A state commitment is considered to be valid if and only if either it is a majority commitment or it is the genesis block.*

Lemma 12. *Bootstrap committee: Consider the interval of rounds* $[1, \kappa]$*. The following arguments hold except with negligible probability in* W*: i)* $[1, \kappa]$ *is* honest prevalent*, ii) for every round* $r \in [1, \kappa]$ *there will eventually exist a valid state commitment which is correct and visible to every honest node, iii) every node requesting a state corresponding to any of the valid state commitments will eventually receive it.*

Proof: i) Only votes coming from nodes elected in the first round are valid for this interval. In the first round, every node computes the VRF for each coin it owns, events that are independent Bernoulli trials with a probability of success $p = \frac{X_1}{2^{|h|}} = \frac{\kappa}{W}$. Since the majority of the coins in the initial stake distribution are held by honest nodes, we can apply Lemma 4 where instead of nodes we sample their coins, to conclude that there will be at least $\frac{\kappa}{2}$ elected coins owned by honest nodes and at most $\frac{\kappa}{2} - 1$ elected coins held by adversarial nodes.
ii) Since the interval is *honest prevalent*, there are at least $\frac{\kappa}{2}$ elected coins held by honest nodes. These honest nodes will apply the valid transactions committed to the ledger to update the state for every round $r \in [1, \kappa]$ and broadcast the corresponding state commitments to every node (Algorithm 6 Procedure Execute). Since the commitment scheme is deterministic and they all start from the genesis block their output will be identical, forming the corresponding valid state commitments which must be correct.
iii) There are at most $\frac{\kappa}{2} - 1$ coins held by adversarial nodes within that interval. Therefore, each valid state commitment is signed by at least one honest node which has stored the state and will eventually provide it to every node requesting it. □

Lemma 13. *Bootstrapping: Consider two nodes* p_i, p_j *that have bootstrapped in the system in the rounds* $r_i, r_j \geq 1$ *respectively. If* $r_i \leq r_j$ *and* p_j *have accepted the chain* $chain_j$ *to bootstrap,* p_i *would consider as valid every state commitment included in* $chain_j$*.*

Proof: For every round $r \geq 1$, p_j has access to the random seed via the dirty ledger for the proofs of election of the elected nodes. $Chain_j$ starts from the genesis block and continues with the valid state commitments that the *bootstrap committee* forms according to Lemma 12. In any round $r > 1$, p_j verifies the respective proofs of ownership coming from nodes with elected coins only when the parent commitment is the valid state commitment of the previous round. For the *voting phase*, p_j takes into account only votes coming from elected nodes. Since p_i and p_j perform the same verifications for the *election* and the *voting* phase, p_i would accept as valid every state commitment included in $chain_2$. □

According to Lemma 13 we cannot distinguish bootstrapping nodes from nodes that have been participating from the first round. From now on, we refer to both as nodes.

Lemma 14. *Consider the interval of rounds* $[r, r + \kappa - 1], r \geq 2$*. If for every round* $r' \in [r - 1, r + \kappa - 2]$ *and for each honest node* p_i*: i)* p_i *eventually receives a valid state commitment which is correct, ii)* $C_{p_i,r'} = C^*_{p_i,r'}$*, then the interval*

$[r, r+\kappa-1]$ *will eventually become* honest prevalent *in every honest node's view except with negligible probability in* W.

Proof: Each coin is elected with a probability of $p = \frac{1}{W}$ after its owner computes the VRF for it. Adversarial nodes cannot construct a proof of ownership for coins they do not own and every honest node with elected coins in any round $r' \in [r, r+\kappa-1]$ will provide a proof of inclusion with parent commitment the valid state commitment of round $r'-1$ that will be accepted by everyone. Since the valid state commitments are correct and in each round the majority of the coins are held by honest nodes by assumption, the argument holds with the analysis done in Lemma 6 (case 2) where the independent Bernoulli trials are the coins' election. □

Lemma 15. *Consider the* honest prevalent *interval of rounds* $[r, r+\kappa-1]$ *and that i) every honest node eventually receives a valid state commitment which is correct for every round in the interval* $[r-1, r+\kappa-2]$*, ii) every honest node with an elected coin in a round* $r' \in [r, r+\kappa-1]$ *will eventually receive the corresponding state of the round* $r'-1$*. There will eventually exist a valid state commitment for the round* $r+\kappa-1$ *which will be correct and visible to every honest node except with negligible probability in* W.

Proof: i) Consider the set H including the honest nodes that own elected coins with that interval. Now, consider the node $p_i \in H$ with elected coins in a round $r' \in [r, r+\kappa-1]$. P_i starts executing from the correct state of the round $r'-1$. Then, it applies the data committed to the ledger up to the round $r'+\kappa-1 \geq r$ via the execution engine M, computes and broadcast to every node the state commitments for all the intermediate rounds, which must be correct since p_i started from a correct state. Node p_i provides the *proof of election* and the *proof of ownership* with the parent commitment $cmt_{r'-1}$, which is the valid and correct state commitment of the round $r'-1$, for all of its elected coins along with the signed state commitment. All honest nodes will verify the *proof of election*, the signature, and the *proof of ownership* since $cmt_{r'-1}$ will eventually be visible to everyone by assumption. Since the nodes in H own at least $\frac{\kappa}{2}$ elected coins and the commitment scheme is deterministic, they will form a valid state commitment for the round r which will be visible to every node. □

Lemma 16. *State availability: Consider the* honest prevalent *interval of rounds* $[r, r+\kappa-1]$*. If the valid state commitment cmt for the round* $r+\kappa-1$ *exists, it will be available to all the nodes requesting it except with negligible probability in* W.

Proof: Consider node p_i that has seen a valid state commitment for the round r and requests the corresponding state. P_i requests the state from all the nodes with elected coins in that interval that have signed for the valid state commitment. Since $[r, r+\kappa-1]$ is *honest prevalent*, there are not enough adversarial members with elected coins to form a *valid* state commitment on their own. Thus, *cmt* must be signed by at least one honest node. This honest node has stored the corresponding state and will eventually send it to p_i. □

Lemma 17. *For every $r \geq 1$, the interval of rounds $[r, r+\kappa-1]$ will eventually become* honest prevalent *in every honest node's view except with negligible probability in W.*

Proof: The interval $[1, \kappa]$ is *honest prevalent* and for every round within that interval there is a corresponding valid state commitment which is correct according to Lemma 12. Since there is a valid state commitment visible to every node for each round, every honest node that was paid within that interval requesting a *proof of payment* will eventually receive it by the payer. Therefore, each honest node p_i can determine whether the transactions it receives are successful and it knows which coins it has spent, so for every round $r \in [1, \kappa]$ $C_{p_i,r} = C^*_{p_i,r}$.

Since there is a valid and correct state commitment for every round in $[1, \kappa]$ and each node tracks its *actual stake*, the interval $[2, \kappa+1]$ will become *honest prevalent* according to Lemma 14. For each round in $[1, \kappa]$, the state corresponding to the valid commitment will be available according to Lemma 12. Finally, we can apply Lemma 15 to conclude that a valid state commitment which is correct will eventually exist for the round $\kappa+1$. For the intervals $[r, r+\kappa-1], r \geq 3$, we apply the same argument recursively where for the state availability of the round $r+\kappa-2$ we use Lemma 16. □

Lemma 18. *Every valid state commitment is correct except with negligible probability in W.*

Proof: We will prove the Lemma by reaching a contradiction. Assume that there is a round r to which the *valid* state commitment cmt_r that is not *correct* corresponds. Then, there must be a round $r^*, 1 \leq r^* \leq r-\kappa+1$ which is the first round s.t. there are at least $\frac{\kappa}{2}$ elected coins within the interval $[r^*, r^*+\kappa-1]$ held by adversarial nodes. By Lemma 17 we conclude that the interval $[r^*, r^*+\kappa-1]$ is *honest prevalent* and thus we reach a contradiction. □

We say that an interval of rounds $[r, r+\kappa-1]$ is *adversarial prevalent* if at least $\frac{\kappa}{2}$ elected coins within that interval are held by adversarial nodes.

Lemma 19. *Every round r corresponding to a block committed to the dirty ledger will eventually be* successful *except with negligible probability in W.*

Proof: For every round $r \in [1, \kappa]$ the *bootstrap committee* forms the respective valid state commitments following from Lemma 12. For every round $r \geq \kappa+1$ the interval $[r-\kappa+1, r]$ is *honest prevalent* and the honest nodes with elected coins within that interval form the valid state commitment for round r according to Lemma 17. □

Lemma 20. *Efficiency guarantees: The expected number of rounds for which an honest node p_i executes equals $\kappa(\kappa W_{p_i,1} + \sum_{r=2}^{h} W_{p_i,r})/W$, where $W_{p_i,r}$ is $p_i's$ stake in the round r, W is the total stake distribution, h is the number of the blocks committed to the dirty ledger, and $\kappa = O(log^2 W)$.*

Proof: The probability with which at least one of $p_i's$ coin is elected in the round r equals $p_{i,r} = \begin{cases} \frac{\kappa W_{p_i,r}}{W}, \text{ if } r = 1 \\ \frac{W_{p_i,r}}{W} \quad \text{if } r > 1 \end{cases}$. When p_i is elected, it executes for κ rounds. We

denote the number of the rounds for which node p_i will execute by the random variable X, expressed as the sum of independent Poisson trials, with excepted value: $E[X] = \sum_{r=1}^{h} \kappa p_{i,r} = \kappa \frac{\kappa W_{p_i,1}}{W} + \sum_{r=2}^{h} \kappa \frac{W_{p_i,r}}{W}$. □

C.3 Execution protocol

We refer to the Lemmas 1, 7, 18 and 3, 10, 19 that guarantee, respectively, that valid state commitments are correct and that executors keep producing them as long as the ledger grows. We prove the properties $E - Safety, E - Liveness$ on top of these Lemmas.

The rationale behind the following definition, which indicates when nodes *commit* to a state, is that we only need to guarantee the generation of valid state commitments and the corresponding states' availability. When those conditions are satisfied, nodes can acquire the state whenever they wish.

Definition 10. *We say that node p_i commits to the state S in the round r, if and only if p_i has received a valid state commitment cmt for that round such that $cmt = compute_{cmt}(S)$.*

Theorem 1. *All of our protocols (Deterministic B.1, Horizontal Sampling 4.1, PoS B.2) satisfy the E-Safety property.*

Proof: Nodes commit only to valid state commitments. Valid state commitments are correct according to Lemmas 1, 7, 18 for each respective protocol (Deterministic, Horizontal Sampling, PoS). Therefore, if node p_i commits to the state S in the round r it must hold that $S = S_r^*$. □

Theorem 2. *All of our protocols (Deterministic B.1, Horizontal Sampling 4.1, PoS B.2) satisfy the E-Liveness property.*

Proof: In all the protocols (Deterministic, Horizontal Sampling, PoS), nodes form a valid state commitment visible to every node for every round according to Lemmas 3,10,19 respectively. Therefore, in all protocols, for every round r and every node p_i there will a round $r' > r$ for which p_i commits to a different state. □

Theorem 3. *All of our protocols (Deterministic B.1, Horizontal Sampling 4.1, PoS B.2) satisfy the* Censorship resistance *property.*

Consider a valid transaction tx committed to the ledger in the block with height r. Enough honest nodes will eventually execute the block in the round r, apply all the valid transactions, and therefore tx, to the state, and form the corresponding valid state commitment according to Lemmas 3,10,19 for each protocol (Deterministic, Horizontal Sampling, PoS) respectively. Following from Lemmas 1, 7, 18, the valid state commitment will be correct, namely every valid transaction, including tx, must have been applied to the state.□

C.4 Light clients

Again we use Theorems 1, 7, 18 and 3, 10, 19 concerning the validity and availability of state commitments.

Lemma 21. *Light nodes perform the same verifications with the executors for valid state commitments in all of our proposed protocols: i) Deterministic B.1, ii) Horizontal Sampling 4.1, iii) PoS B.2.*

Proof: i) In the Deterministic protocol, the light clients know in advance the public keys of the executors.
ii) In the Horizontal Sampling, light clients know the public keys of the executors before the protocol starts as well as they have access to the random seed for each round through the dirty ledger, to verify the proofs of the election of the nodes.
iii) Following from Lemma 13. □

Theorem 4. *All of the proposed protocols (Deterministic B.1, Horizontal Sampling 4.1, PoS B.2) satisfy the LC-Safety property.*

Proof: A light node joins the network after having received a *state proof* consisting of a valid state commitment and the respective certificates. Assume that there exists a round r in which the adversary provides a state proof $\pi_{\tilde{S}}$ corresponding to a faulty state $\tilde{S}$ to a light client lc_i s.t. lc_i outputs $accept_{lc_i}(\pi_{\tilde{S}}, r) = True$. Since light clients perform the same verifications as honest nodes, according to Lemma 21, the adversary would convince the executors about the validity of the faulty commitment $cmt = Compute_{cmt}(\tilde{S})$ as well. That is contradicting Lemmas 1, 7, 18 for each respective protocol (Deterministic, Horizontal Sampling, PoS). □

Theorem 5. *All of the proposed protocols (Deterministic B.1, Horizontal Sampling 4.1, PoS B.2) satisfy the LC-Liveness property.*

Proof: Consider the light client lc_i that requests a *state proof* for a round greater or equal than r. Each round will eventually become *successful* in all protocols (Deterministic, Horizontal Sampling, PoS) as follows from the Lemmas 3, 10, 19. The honest executors store all the certificates related to the valid state commitments. Lc_i is connected to at least one honest node, let us call it p_i. Since round r will become successful, p_i will eventually see a valid state commitment corresponding to a round $r^* \geq r$. The *state proof* π_S which p_i will construct and send to lc_i, contains the state commitment and the respective certificates indicating its validity. The light client will verify π_S, since the honest executor had already verified the certificates, and output $accept_{lc_i}(\pi_S, r^*) = True$. □

For the following Lemma, we denote the size of the nodes' signatures by c_1, the size of the *proof of election* (proof coming from the VRF) by c_2, and the average size of the *proof of ownership* (inclusion proof) by $I(|S|)$. We assume that $\frac{I(|S|)}{|S|} = o(1)$, where $|S|$ is the average size of the state. For example, in Merkle proofs $I(|S|) = O(log(|S|))$.

Lemma 22. *A* state proof π *for a round* $r \geq 1$*, is succinct in all of our proposed protocols: i) Deterministic B.1, ii) Horizontal Sampling 4.1, iii) PoS B.2.*

Proof: i) In the Deterministic protocol, π constitutes of $f + 1$ signatures and therefore $len(\pi) = (f + 1)c_1$.
ii) In the Horizontal Sampling, π consists of $O(log^2 n)$ signatures and the respective *proofs of election* and thus $len(\pi) = (c_1 + c_2)O(log^2 n)$.
iii) In the Proof-of-Stake protocol, for every round up to the round r there are $O(log^2 W)$ votes along with the signatures and the proofs of election and ownership. Consequently, $len(\pi) = r(c_1 + c_2 + I(S))O(log^2 W)$. □

D Discussion and Extensions

In this section, we discuss technical details and extensions related to the implementation of the proposed protocols.

- For simplicity, in the proposed protocols, we presented straightforwards solutions for the leader election phase in the proof-of-stake protocol and the state transition in all the protocols. First, for the leader election, instead of assuming that nodes compute the VRF for all of their coins, we can deploy the sortition algorithm presented in Algorand [22]. In that case, nodes compute the VRF only once and their voting power is determined by the sortition algorithm as a function of the occurring hash value and their stake. Furthermore, for the state transition protocol there are efficient protocols to compute the difference between the state of different rounds using Bloom Filters and Invertible Bloom Filters [30,18].
- Security parameter κ: In the main paper we assumed a large number of participants, and therefore the analysis is asymptotic choosing the security parameter as a function of the number of participants. Now, we cover the case where the number of participants is small. In a nutshell, we will allow the probability of failure to be low, but not necessarily negligible, and define a fallback mechanism as follows.
  The first challenge we must address is the case of a safety violation, i.e., enough adversarial nodes are elected and vote for an invalid commitment. In the asynchronous model consensus is solvable only for $f < \frac{n}{3}$, and in figure 3 we demonstrate that in that case the probability of a safety violation is sufficiently low even for a small κ. Blockchain systems with $f > \frac{n}{3}$ work in the synchronous model, in which case we adopt the following approach. When a safety violation happens, the honest elected leaders construct a fraud proof proving that the transition from the latest valid state commitment to the faulty one is not correct. To avoid spamming, i.e., adversarial nodes broadcast faulty fraud proofs in every round, we slash the nodes that provide faulty fraud proofs. An occurring challenge is that if a client had to check every fraud proof, in the worst case it would execute the whole ledger. To support light client constructions, all executors sign the correct fraud proofs and the light clients take into account only the fraud proofs voted from the majority of the nodes.

On the other hand, for the rounds that there are not elected enough honest nodes to form a valid state commitment, the protocol proceeds in epochs constituting of rounds where the election process restarts until electing enough honest nodes.
Moreover, in Appendix E we give an upper bound of the failure probability as a function of the number of nodes that execute per round (κ) for different of values of the constant ϵ.

– Adaptivity of the adversary: All of the proposed protocols refer to a static adversary. In Appendix F, we discuss how to extend those protocols to withstand an adaptive adversary.

E Elected Nodes per round

In this section, we bound the failure probabilities for different values of the constant ϵ as a function of the number of elected nodes to execute per round (κ). We remind that for a small number of participants, we can either use the Deterministic protocol B.1 or employ a fallback mechanism.

By applying $\epsilon = \frac{1}{10}$ ($f = \frac{9}{20}n$) to Lemma 4 a safety violation, i.e. more than κ adversarial nodes are elected to execute for a round, occurs with a probability of at most: $P[F \geq \frac{\kappa}{2}] \leq e^{-\frac{\kappa}{342}}$. Moreover, the probability that there will not be enough honest nodes to execute for a round is at most $P[H \leq \frac{21}{20}\frac{\kappa}{2}] \leq e^{-\frac{\kappa}{1760}}$.

For $\epsilon = \frac{1}{3}$ ($f = \frac{n}{3}$), we use Chernoff bounds again to calculate the respective probabilities. A safety violation happens with a probability of at most $P[F \geq \frac{\kappa}{2}] = P[F \geq (1+\frac{1}{2})\frac{\kappa}{3}] \leq e^{-\frac{\kappa}{30}}$ and a liveness failure happens with a probability of at most $P[H \leq \frac{\kappa}{2}] = P[H \leq (1-\frac{1}{4})\frac{2\kappa}{3}] \leq e^{-\frac{\kappa}{48}}$.

For $\epsilon = \frac{1}{2}$ ($f = \frac{n}{4}$), we use the Chernoff bounds to compute the same probabilities:
$P[F \geq \frac{\kappa}{2}] \leq e^{-\frac{\kappa}{12}}$ and $P[H \leq \frac{\kappa}{2}] = P[H \leq (1-\frac{1}{4})\frac{2\kappa}{3}] \leq e^{-\frac{\kappa}{18}}$.

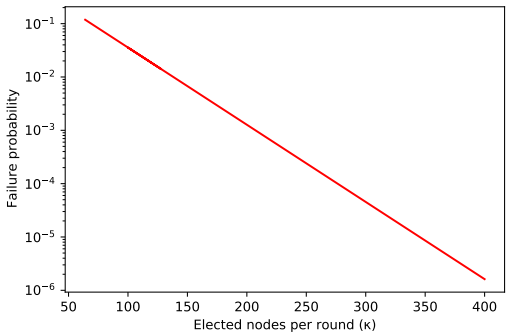

Fig. 3: An upper bound on the probability of a safety violation for $f = \frac{n}{3}$ as a function of the number of nodes that execute per round.

.

F Towards an adaptive adversary

Challenges for dealing with an adaptive adversary. Corrupting nodes during the run is not a problem for the deterministic protocol. The adversary can corrupt at most f executors, hence each valid state commitment is signed by at least one node that will always act truthfully. Additionally, in each round, at least $f + 1$ honest executors are responsible for executing and storing the respective state. Consequently, even in the presence of an adaptive adversary, the deterministic protocol ensures safety and liveness.

Nevertheless, the probabilistic protocols face the following challenges. First, elected nodes request the state of a previous round before executing. As a result, the adversary can identify which nodes were elected in any round after receiving the state request and corrupt them to form a faulty valid state commitment. Now consider that we have solved the aforementioned challenge, namely the adversary cannot distinguish the elected leaders before they propose a state commitment. Even in that case, the adversary can corrupt the leaders of a round after witnessing their votes and make them vote again. In that case, a receiver (an executor or a light client) cannot distinguish which votes are the valid ones.

Withstanding the adaptive adversary To deal with the adaptive adversary, we must address two challenges. First, the elected nodes should not announce their election before completing the execution. Second, the adversary must be unable to re-execute, i.e. sign another state commitment, after corrupting an elected node that has previously executed.

To resolve the first challenge, once the elected leaders finish the execution, they broadcast the signed state commitment along with the updated state to all nodes. In that way, the state will eventually be available to all executors. Therefore, the elected nodes will have acquired the state needed before executing.

For the second challenge, we use ephemeral keys that can be utilized by Merkle Trees [28] or as was proposed by Micali et al. [22]. In a nutshell, each node has a pair of masterkeys which it uses to generate a pair of keys for each round. After executing for a round, honest nodes delete the corresponding private key. Thus, even if the adversary corrupts an elected node, signing another state commitment for that round is impossible.

To summarize, an elected leader computes and signs the state commitment of that round, deletes its private key, and then broadcasts the signed state commitment along with the updated state and the election certificates (*proof of election* and the *proofs of ownership*).

Note that, in the asynchronous model, sending the state commitment along with the state in batches is essential. If elected leaders only broadcast the state commitment first, the adversary would manage to attack the system with respect to liveness. More specifically, the adversary would corrupt the executors that have signed the valid state commitment and prevent them from sending the state to the nodes requesting it. We call this scenario *corrupted round* attack.

Relaxing the network settings. Now, we explain how we can decouple the transition of the state commitment from the state in the synchronous model. Namely, elected nodes compute the state commitment, delete their private key,

and then broadcast the signed state commitment and the updated state sequentially. This is done to avoid delays occurring in the network when batching the state commitments with the respective state.

Due to the synchrony assumptions, the executors know whether a round is corrupted or not. They will therefore just execute for the rounds in which the system is under the *corrupted round* attack. Fortunately, the adversary can perform this attack for at most f rounds. In particular, each valid state commitment is signed by at least one honest leader, that has stored the corresponding state. To "corrupt" a round, the adversary must corrupt all the honest leaders that have signed the valid state commitment.

Comments on the efficiency. An elected leader would have to wait until receiving the state of the previous round even with a static adversary. However, the waiting time, i.e. the time delay between the election of a node until it receives the state, might get increased since the state is being broadcast to every node. In practice, we can employ an efficient state transition protocol where the elected nodes would only broadcast the difference of the state in consecutive rounds [30,18]. We should note that again Horizontal Sampling is more efficient than Vertical Sampling since the nodes require the state less frequently, which means that the impact of the waiting time increase on them is less.