Proofs of Proof-of-Stake with Sublinear Complexity

Shresth Agrawal
Technische Universität München, Germany

Joachim Neu
Stanford University, USA

Ertem Nusret Tas
Stanford University, USA

Dionysis Zindros
Stanford University, USA

Abstract

Popular Ethereum wallets (like MetaMask) entrust centralized infrastructure providers (*e.g.*, Infura) to run the consensus client logic on their behalf. As a result, these wallets are light-weight and high-performant, but come with security risks. A malicious provider can mislead the wallet by faking payments and balances, or censoring transactions. On the other hand, light clients, which are not in popular use today, allow decentralization, but are concretely inefficient, often with asymptotically *linear* bootstrapping complexity. This poses a dilemma between decentralization and performance.

We design, implement, and evaluate a new proof-of-stake (PoS) *superlight* client with concretely efficient and asymptotically *logarithmic* bootstrapping complexity. Our proofs of proof-of-stake (PoPoS) take the form of a Merkle tree of PoS epochs. The verifier enrolls the provers in a bisection game, in which honest provers are destined to win once an adversarial Merkle tree is challenged at sufficient depth. We provide an implementation for mainnet Ethereum: compared to the state-of-the-art light client construction of Ethereum, our client improves time-to-completion by 9×, communication by 180×, and energy usage by 30× (when bootstrapping after 10 years of consensus execution). As an important additional application, our construction can be used to realize trustless cross-chain bridges, in which the superlight client runs within a smart contract and takes the role of an on-chain verifier. We prove our construction is secure and show how to employ it for other PoS systems such as Cardano (with fully adaptive adversary), Algorand, and Snow White.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Proof-of-stake, blockchain, light client, superlight, bridge, Ethereum

1 Introduction

If I want to check how much money I have on Ethereum, the most secure way is to run my own full node. Sadly, this requires downloading more than 500 GB of data and can take up to 5 days to sync. Because of this, most Ethereum users today outsource the task of maintaining the latest state to a third-party provider such as Infura or Alchemy. This allows the user to run a lightweight wallet, such as MetaMask[1], on their browser or smartphone.

What can go wrong if Infura is compromised or turns malicious? As per the current wallet design, the wallet will blindly trust the provider and therefore show incorrect data. This is enough to perform a double spend. For example, imagine Bob wants to sell his car to Eve. Regrettably, Eve has compromised Infura. Eve claims that she has paid Bob. Bob checks his MetaMask wallet and sees an incoming and confirmed transaction from Eve. Unfortunately, even though the wallet is non-custodial, this transaction never happened, but is fraudulently reported by Infura to Bob. Since Bob trusts his wallet, and his wallet trusts Infura, he hands

[1] MetaMask has 21,000,000 monthly active users as of July 2022 [51] and is the most popular non-custodial wallet [17].

over the car keys to Eve. By the time Bob realizes he cannot use this money, Eve has long disappeared with his car in Venice. From the point of view of Infura, this is a huge liability. If Infura is compromised, then all of its users are immediately compromised.

This creates a dilemma between good performance and security for the users. This problem is not unique to Ethereum and appears in all proof-of-stake (PoS) systems. In this paper, we resolve this dilemma by constructing a PoS blockchain client which is both efficient and secure. We solve this problem by constructing a protocol that allows efficiently verifying the proof-of-stake blockchain without downloading the whole PoS. We call such constructions succinct *proofs of proof-of-stake*. These allow us to build *superlight clients*, clients whose communication complexity grows sublinearly with the lifetime of the system.

Contributions.

1. We give the first formal definition for succinct proof of proof-of-stake (PoPoS) protocols.
2. We put forth a solution to the long-standing problem of efficient PoS bootstrapping. Our solution is exponentially better than previous work.
3. We report on our implementation of a fully functional and highly performant superlight client for mainnet Ethereum. It is the first such construction for Ethereum. We measure and contrast the performance of our client against the currently proposed design for Ethereum.
4. We theoretically show our construction is secure for Ethereum and other PoS blockchains.

1.1 Construction Overview

Let us discuss the intuition about how our construction works. We start with the existing full node design and iteratively make it more lightweight.

Full nodes. A *full node* client boots holding only the genesis block and connects to other full nodes (known as *provers*) in order to synchronize to the latest tip. The full node downloads the entire chain block-by-block verifying each transaction in the process. This incurs high communication and computational complexity. Once the client verifies the latest tip, it has calculated the latest state and can answer user queries such as *"what is my balance?"*. In order for the full node to get to the correct tip, at least one connection to an honest peer is required (this is known as the *existential honesty* assumption [29, 28, 30, 47]).

Sync committees. Let's try to improve on the efficiency of the full node to make it a *light client*. PoS protocols typically proceed in *epochs* during which the validator set is fixed. In each epoch, a subset of validators is elected by the protocol as the epoch *committee*. This is a set of public keys. The security of the protocol assumes that the majority [19, 42, 21, 2] or super-majority [9, 13, 5, 54] of the committee members are honest during the epoch. The current committee signs the latest state. Therefore, the client does not need to download all the blocks, but instead only needs to download the latest state and verify the committee signatures on it. However, the stake changes hands in every epoch. Hence, to perform the verification at some later epoch, the client needs to keep track of the current committee. To help the client in this endeavor, the committee members of each epoch, while active, sign a *handover* message inaugurating the members of the new committee [31]. This enables the light client to discover the latest committee by processing a sequence of such handovers.

Optimistic light client construction. Light clients like these already exist. Regrettably, they still need to download all committees of the past to verify the current state. In this paper we propose better solutions. The first idea, which we call the *optimistic light client construction*, is for the prover to work as follows: For each committee, take its members, concatenate them all together, and hash them into one hash. The prover then sends this sequence of hashes (one for each committee) to the client. Since the client is connected to at least one honest prover, at least one of these provers will answer truthfully. If multiple provers give conflicting claims to the client, all it needs to do is to find which one is truthful. To do this, it compares the claims of all provers pairwise. If two provers disagree, the client focuses on the first point of disagreement in their hash sequences, and asks each prover to provide the respective handover signatures to substantiate their claim. Each prover reveals the committee attested by the hash at that point, the previous committee and the associated handover messages. Upon validating these messages, which can be done locally and efficiently, the client identifies the truthful party, and accepts its state. A lying prover will not be able to provide such a handover for an invalid committee. Once the client rejects the invalid committee claims, it will have calculated the latest committee, and can proceed from there as usual.

Superlight clients. Even though the complexity is concretely improved, the sequence of committee hashes still grows linearly with the lifetime of the protocol. To achieve sublinear complexity, we improve the procedure to find the first point of disagreement. To this end, our final PoPoS protocol requires each prover to organize its claimed sequence of committees—one per epoch—into a Merkle tree [46]. The roots of those trees are then sent over to the client, who compares them. Upon detecting disagreement at the roots, the client asks the provers to reveal the children of their respective roots. By repeating this process recursively on the mismatching children, it arrives at the *first point of disagreement* between the claimed committee sequences, in logarithmic number of steps. After the first point of disagreement is found, the client works similarly to the optimistic light client construction. This process achieves logarithmic communication.

Bridges. Our PoPoS construction has two main applications: *Superlight* proof-of-stake clients that can bootstrap very efficiently, and *trustless bridges* that allow the passing of information from one proof-of-stake chain to another. For bridging, we note that a trustless bridge is nothing more than an on-chain superlight client. It connects a source PoS blockchain to any other destination blockchain. To do this, a smart contract on the destination chain, which runs an implementation of our superlight client code (*e.g.*, in Solidity), is deployed. Whenever some information of interest appears on the source chain, any helpful but untrusted relayer can submit this information, together with the PoPoS proof to the smart contract. If the information is inaccurate, another relayer challenges the first one by participating in our interactive bisection game within a dispute period [43], submitting one transaction for every round of interaction of the PoPoS protocol. If both chains are PoS, the bridge can be made bidirectional by running PoPoS superlight clients on both chains. To incentivize on-chain participation, relayers who submit accurate information are rewarded, whereas relayers who submit inaccurate information need to put up a collateral which is slashed and is used to reward the challenger.

Table 1 Comparison with previous works in terms of *asymptotic* $\widetilde{\Theta}$ communication complexity, interactivity, and cryptographic model. Interactivity is the number of rounds, ignoring constants. Low communication and rounds of interactivity are preferable. N: number of epochs. L: number of blocks per epoch. Common prefix parameter k is constant.

	SPV PoW [29]	**KLS** [38]	**FlyClient** [6]	**Superblocks** [40, 39]	**Full PoS** [42]	**Mithril** [14]	**Coda** [4]	**PoPoS** (this work)
Communication	NL	$\log(NL)$	$\text{poly}\log(NL)$	$\log(NL)$	NL	$N+L$	1	$\log(N)$
Interactivity	1	$\log(NL)$	1	1	1	1	1	$\log(N)$
Work/stake	work	work	work	work	stake	stake	both	stake
Model	RO	RO	RO	RO	standard	RO	CRS	standard
Primitives	hash	hash	hash	hash	hash, sig	hash, sig, ZK	hash, sig, ZK	hash, sig

1.2 Implementation Overview

In addition to our theoretical contributions, we report on our open source implementation (spanning about 8,200 lines of TypeScript code and 2,300 downloads from the community) of our protocol for the Ethereum mainnet. Our implementation is fully functional and supports all RPC queries used by typical wallets, from simple payments to complex smart contract calls. It takes the form of a modular daemon that augments any existing Ethereum wallet's functionality (such as MetaMask's) to be trustless, without changing any user experience. We perform measurements using the light client protocol currently proposed for Ethereum. We find that this protocol, while much more efficient than a full node, is likely insufficient to support communication-, computation-, and battery-constrained devices such as browsers and mobile phones. Next, we measure the performance of our implementation of an *optimistic light client* for Ethereum that achieves significant gains over the traditional light client. We demonstrate this implementation is already feasible for resource-constrained devices. Lastly, our implementation includes a series of experiments introducing a *superlight client* for Ethereum that achieves exponential asymptotic gains over the optimistic light client. These gains constitute concrete improvements over the optimistic light client when the blockchain system is long-lived and has an execution history of a few years. We compare all three clients in terms of communication (bandwidth and latency), computation, and energy consumption.

1.3 Related Work

Table 1 compares the current paper with related work. Proof-of-work superlight clients have been explored in the interactive [38] and non-interactive [40] setting using various constructions [39, 35, 6]. Such constructions are backwards compatible [56, 41] and have been deployed in practice [20]. They have also been used in the context of bridges [36, 1, 43, 55]. Several proof-of-stake-specific clients [31, 14, 24] improve the efficiency of full nodes, but require linear communication. Chatzigiannis et al. [15] provide a survey of light clients.

Our construction is based on refereed proofs [12, 33, 34, 49]. PoPoS constructions can also be built via (recursive) SNARKs/STARKs [4, 27], some achieving constant communication. However, these are clients for chains that were designed from the start to be proof-friendly. Current attempts [53, 48] to retrofit popular PoS protocols (*e.g.*, Ethereum and Cosmos) with SNARK-based proofs are expensive (annual prover operating-cost of six to seven figures

USD).[2] Additionally, since [53, 48] do not include recursive proving/verification, they require the validator to remain online (to avoid linear overhead), which may be a suitable assumption for bridges, but not for bootstrapping light clients (*e.g.*, intermittently online mobile wallets). Unlike proof-based approaches, our protocol does not require changes in the PoS protocol, and relies on simple primitives such as hashes and signatures. Our prover is stateless and adds only a few milliseconds of computation time to an existing full node. Our light client allows for bootstrapping from genesis with sublinear overhead.

Some clients obtain a *checkpoint* [44, 8] on a recent block from a trusted source, after which they download only a constant number of block headers to identify the tip. Our construction allows augmenting these clients so that they can *succinctly verify* the veracity of a checkpoint without relying on a trusted third party.

1.4 Outline

We present our theoretical protocol in a *generic* PoS framework, which typical proof-of-stake systems fit into. We prove our protocol is secure if the underlying blockchain protocol satisfies certain simple and straightforward *axioms*. Many popular PoS blockchains can be made to fit within our axiomatic framework. We define our desired primitive, the proof of proof-of-stake (PoPoS), together with the axioms required from the underlying PoS protocol in Sec. 3. We iteratively build and present our construction in Secs. 4 and 5. We present the security claims in Sec. 8. For concreteness, and because it is the most prominent PoS protocol, we give a concrete construction of our protocol for Ethereum in Sec. 6. Ethereum is the most widely adopted PoS protocol. Interestingly, Ethereum directly satisfies our axiomatic framework and does not require any changes on the consensus layer at all. The applicability of our framework to other PoS chains such as Ouroboros (Cardano), Algorand, and Snow White are discussed in App. D.

The description of our implementation and the relevant experimental measurements showcasing the advantages of our implementation are presented in Sec. 7.

2 Preliminaries

Proof-of-stake. Our protocols work in the proof-of-stake (PoS) setting. In a PoS protocol, participants transfer value and maintain a balance sheet of *stake*, or *who owns what*, among each other. It is assumed that the *majority of stake* is honestly controlled at every point in time. The PoS protocol uses the current stake *distribution* to establish consensus. The exact mechanism by which consensus is reached varies by PoS protocol. Our PoPoS protocol works for popular PoS flavours.

Primitives. Participants in our PoS protocol transfer stake by *signing* transactions using a signature scheme [37]. The public key associated with each validator is known by everyone. The signatures are key-evolving, and honest validators delete their old keys after signing [32,

[2] For example, according to [53, Section 6], proving consensus (of 128 validators) on *one* Cosmos block takes 18 seconds on 32 instances of Amazon AWS `c5.24xlarge`. (We are unable to independently reproduce this finding because the authors of zkBridge have not disclosed their code.) At Cosmos' block rate of 1 block per second, that would require 18×32 continuously operating `c5.24xlarge` instances, costing annually \$12,967,488 on Amazon AWS (annual pricing, `us-east-1` region, June 2023), or \$1,749,600 on Hetzner's equivalents. Scaling this proportionally for Ethereum ($\times 4$ for sync committee size 512, /12 for block rate 1 block per 12 seconds) yields an estimate of \$583,333 annually.

21].[3] We also use a collision resistant hash function. We highlight that it does not need to be treated in the Random Oracle model, and no trusted setup is required for our protocol (beyond what the underlying PoS protocol may need).

Types of nodes. The stakeholders who participate in maintaining the system's consensus are known as *validators.* In addition to those, other parties, who do not participate in maintaining consensus, can join the system, download its full history, and discover its current state. These are known as *full nodes.* Clients that are interested in joining the system and learning a small part of the system state (such as their user's balance) without downloading everything are known as *light clients.* Both full nodes and light clients can join the system at a later time, after it has already been executing for some duration $|\mathcal{C}|$. A late-joining light client or full node must *bootstrap* by downloading some data from its peers. The amount of data the light client downloads to complete the bootstrapping process is known as its *communication complexity.* A light client is *succinct* if its communication complexity is $\mathcal{O}(\text{poly}\log(|\mathcal{C}|))$ in the lifetime $|\mathcal{C}|$ of the system. Succinct light clients are also called *superlight clients.* The goal of this paper is to develop a PoS superlight client.

Time. The protocol execution proceeds in discrete *epochs*, roughly corresponding to moderate time intervals such as one day. Epochs are further subdivided into *rounds*, which correspond to shorter time durations during which a message sent by one honest party is received by all others. In our analysis, we assume synchronous communication. The validator set stays fixed during an epoch, and it is known one epoch in advance. The validator set of an epoch is determined by the snapshot of stake distribution at the beginning of the previous epoch. To guarantee an honest majority of validators at any epoch, we assume a *delayed honest majority* for a duration of *two epochs*: Specifically, if a snapshot of the current stake distribution is taken at the beginning of an epoch, this snapshot satisfies the honest majority assumption for a duration of two full epochs. Additionally, we assume that the adversary is *slowly adaptive*: She can corrupt any honest party, while respecting the honest majority assumption, but that corruption only takes place two epochs later. This assumption will be critical in our construction of *handover* messages that allow members of one epoch to inaugurate a committee representing the next epoch (*cf.* Sec. 4).

The prover/verifier model. The bootstrapping process begins with a light client connecting to its full node peers to begin synchronizing. During the synchronization process, the full nodes are trying to convince the light client of the system's state. In this context, the light client is known as the *verifier* and the full nodes are known as the *provers.* As usual, we assume the verifier is connected to at least one honest prover. The verifier queries the provers about the state of the system, and can exchange multiple messages to interrogate them about the truth of their claims during an *interactive protocol.*

Assumptions. We make two central assumptions: Firstly, that the light client can communicate *interactively* with full nodes. This is contrary to, for example, proof-based clients. Interactivity incurs a penalty when our light client runs on-chain, because it requires receiving data over the course of multiple transactions. This is expensive in gas and time consuming

[3] Instead of key-evolving signatures, Ethereum relies on a concept called *weak subjectivity* [8]. This alternative assumption can also be used in the place of key-evolving signatures to prevent posterior corruption attacks [22].

in delays. Secondly, that the light client has *at least one honest* connection. Many protocols assume this. For example, a Bitcoin full node is not secure if all connections are dishonest. In current systems, such as Ethereum, light clients typically connect to RPC nodes instead of full nodes. It is better to trust at least *one among many* RPC connections is honest (as opposed to having a single RPC connection), but one may still become eclipsed. To resolve this concern, we advocate for light clients to connect to full nodes directly, which would make this assumption more reasonable. Due to these two assumptions, we caution the reader to be aware of the current limitations of our work, understanding it is not always applicable.

Notation. We use ϵ and $[\,]$ to mean the empty string and empty sequence. By $x \parallel y$, we mean the string concatenation of x and y encoded in a way that x and y can be unambiguously retrieved. We denote by $|\mathcal{C}|$ the length of the sequence $\mathcal{C}$; by $\mathcal{C}[i]$ the i^{th} (zero-based) element of the sequence, and by $\mathcal{C}[-i]$ the i^{th} element from the end. We use $\mathcal{C}[i{:}j]$ to mean the subarray of $\mathcal{C}$ from the i^{th} element (inclusive) to the j^{th} element (exclusive). Omitting i takes the sequence to the beginning, and omitting j takes the sequence to the end. We write $A \preccurlyeq B$ to mean that the sequence A is a prefix of B. We use λ to denote the security parameter. Following Go notation, in our multi-party algorithms, we use $m \dashrightarrow A$ to indicate that message m is sent to party A and $m \dashleftarrow A$ to indicate that message m is received from party A.

Ledgers. The consensus protocol attempts to maintain a unified view of a *ledger* $\mathbb{L}$. The ledger is a sequence of *transactions* $\mathbb{L} = (\mathsf{tx}_1, \mathsf{tx}_2, \ldots)$. Each validator and full node has a different view of the ledger. We denote the ledger of party P at round r as $\mathbb{L}^P_r$. Nodes joining the protocol, whether they are validators, full nodes, or (super)light clients, can also *write* to the ledger by asking for a transaction to be included. In a secure consensus protocol, all honestly adopted ledgers are prefixes of one another. We denote the longest among these ledgers as $\mathbb{L}^{\cup}_r$, and the shortest among them as $\mathbb{L}^{\cap}_r$. We will build our protocol on top of PoS protocols that are secure. A *secure* consensus protocol enjoys the following two virtues:

▶ **Definition 1** (Consensus Security). *A consensus protocol is* secure *if it is:*

1. ***Safe:*** *For any honest parties P_1, P_2 and rounds $r_1 \leq r_2$: $\mathbb{L}^{P_1}_{r_1} \preccurlyeq \mathbb{L}^{P_2}_{r_2}$.*
2. ***Live:*** *If all honest validators attempt to* write *a transaction during u consecutive rounds $r_1, \ldots, r_u$, it is included in $\mathbb{L}^P_{r_u}$ of any honest party P.*

Transactions. A transaction encodes an update to the system's state. For example, a transaction could indicate a value transfer of 5 units from Alice to Bob. Different systems use different transaction formats, but the particular format is unimportant for our purposes. A transaction can be applied on the current *state* of the system to reach a new state. Given a state st and a transaction tx, the new state is computed by applying the state transition function δ to the state and transaction. The new state is then $\mathsf{st}' = \delta(\mathsf{st}, \mathsf{tx})$. For example, in Ethereum, the state of the system encodes a list of balances of all participants [7, 52]. The system begins its lifetime by starting at a genesis state st_0. A ledger also corresponds to a particular system state, the state obtained by applying its transactions iteratively to the genesis state. Consider a ledger $\mathbb{L} = (\mathsf{tx}_1 \cdots \mathsf{tx}_n)$. Then the state of the system is $\delta(\cdots\delta(\mathsf{st}_0, \mathsf{tx}_1), \cdots, \mathsf{tx}_n)$. We use the shorthand notation δ^* to apply a sequence of transactions $\overline{\mathsf{tx}} = \mathsf{tx}_1 \cdots \mathsf{tx}_n$ to a state. Namely, $\delta^*(\mathsf{st}_0, \overline{\mathsf{tx}}) = \delta(\cdots\delta(\mathsf{st}_0, \mathsf{tx}_1), \cdots, \mathsf{tx}_n)$.

Because the state of the system is large, the state is compressed using an authenticated data structure (*e.g.*, Merkle Tree [46]). We denote by $\langle \mathsf{st} \rangle$ the state *commitment*, which is this short representation of the state st (*e.g.*, Merkle Tree root). Given a state commitment $\langle \mathsf{st} \rangle$ and a transaction tx, it is possible to calculate the state commitment $\langle \mathsf{st}' \rangle$ to the new state $\mathsf{st}' = \delta(\mathsf{st}, \mathsf{tx})$. However, this calculation may require a small amount of auxiliary data π such as a Merkle tree proof of inclusion of certain elements in the state commitment $\langle \mathsf{st} \rangle$. We denote the transition that is performed at the state commitment level by the *succinct transition function* $\langle \delta \rangle$. Concretely, we will write that $\langle \delta(\mathsf{st}, \mathsf{tx}) \rangle = \langle \delta \rangle (\langle \mathsf{st} \rangle, \mathsf{tx}, \pi)$. This means that, if we take state st and apply transaction tx to it using the transition function δ, and subsequently calculate its commitment using the $\langle \cdot \rangle$ operator, the resulting state commitment is the same as the one obtained by applying the succinct transition function $\langle \delta \rangle$ to the state commitment $\langle \mathsf{st} \rangle$ and transaction tx using the auxiliary data π. If the auxiliary data is incorrect, the function $\langle \delta \rangle$ returns $\perp$ to indicate failure. If the state commitment uses a secure authenticated data structure such as a Merkle tree, we can only find a unique π that makes the $\langle \delta \rangle$ function run successfully.

3 The PoPoS Primitive

The PoPoS abstraction. Every verifier $\mathcal{V}$ online at some round r holds a state commitment $\langle \mathsf{st} \rangle_r^{\mathcal{V}}$. To learn about this recent state, the verifier connects to provers $\mathcal{P} = \{P_1, P_2, \cdots, P_q\}$. All provers except one honest party can be controlled by the adversary, and the verifier does not know which party among the provers is honest (the verifier is assumed to be honest). The honest provers are always online. Each of them maintains a ledger $\mathbb{L}_i$. They are consistent by the safety of the underlying PoS protocol. Upon receiving a query from the verifier, each honest prover sends back a state commitment corresponding to its current ledger. However, the adversarial provers might provide incorrect or outdated commitments that are different from those served by their honest peers. To identify the correct commitment, the light client mediates an *interactive* protocol among the provers:

▶ **Definition 2** (Proof of Proof-of-Stake). *A* Proof of Proof-of-Stake protocol *(PoPoS) for a PoS consensus protocol is a pair of interactive probabilistic polynomial-time algorithms (P, V). The algorithm P is the* honest prover *and the algorithm V is the* honest verifier. *The algorithm P is ran by an online full node, while V is a light client booting up for the first time holding only the genesis state commitment $\langle \mathsf{st}_0 \rangle$. The protocol is executed between V and a set of provers $\mathcal{P}$. After completing the interaction, V returns a state commitment $\langle \mathsf{st} \rangle$.*

Security of the PoPoS protocol. The goal of the verifier is to output a state commitment consistent with the view of the honest provers. This is reflected by the following security definition of the PoPoS protocol.

▶ **Definition 3** (State Security). *Consider a PoPoS protocol (P, V) executed at round r, where V returns $\langle \mathsf{st} \rangle$. It is* secure *with* parameter *ν if there exists a ledger $\mathbb{L}$ such that $\langle \mathsf{st} \rangle = \delta^*(\mathsf{st}_0, \mathbb{L})$, and $\mathbb{L}$ satisfies:*

- ***Safety:*** *For all rounds $r' \geq r + \nu$: $\mathbb{L} \preccurlyeq \mathbb{L}_{r'}^{\cup}$.*
- ***Liveness:*** *For all rounds $r' \leq r - \nu$: $\mathbb{L}_{r'}^{\cap} \preccurlyeq \mathbb{L}$.*

State security implies that the commitment returned by a verifier corresponds to a state recently obtained by the honest provers.

4 The Optimistic Light Client

Before we present our succinct PoPoS protocol, we introduce sync committees and handover messages, two necessary components we use in our construction. We also propose a highly performant optimistic light client as a building block for the superlight clients.

Sync committees. To allow the verifier to achieve state security, we introduce a *sync committee* (first proposed in the context of PoS sidechains [31]). Each committee is elected for the duration of an epoch, and contains a subset, of fixed size m, of the public keys of the validators associated with that epoch. The committee of the next epoch is determined in advance at the beginning of the previous epoch. All honest validators agree on this committee. The validators in the sync committee are sampled from the validator set of the corresponding epoch in such a manner that the committee retains honest majority during the epoch. The exact means of sampling are dependent on the PoS implementation. One way to construct the sync committee is to sample uniformly at random from the underlying stake distribution using the epoch randomness of the PoS protocol [42, 25]. The first committee S^0 is recorded by the genesis state st_0. We denote the set of public keys of the sync committee assigned to epoch $j \in \mathbb{N}$ by S^j, and each committee member public key within S^j by S^j_i, $i \in \mathbb{N}$.

Handover signatures. During each epoch j, each honest committee member S^j_i of epoch j signs the tuple $(j+1, S^{j+1})$, where $j+1$ is the next epoch index and S^{j+1} is the set of all committee member public keys of epoch $j+1$. We let σ^j_i denote the signature of S^j_i on the tuple $(j+1, S^{j+1})$. This signature means that member S^j_i approves the inauguration of the next epoch committee. We call those *handover signatures*[4], as they signify that the previous epoch committee *hands over* control to the next committee. When epoch $j+1$ starts, the members of the committee S^j assigned to epoch j can no longer use their keys to create handover signatures.[5] As soon as more than $\frac{m}{2}$ members of S^j have approved the inauguration of the next epoch committee, the inauguration is ratified. This collection of signatures for the handover between epoch j and $j+1$ is denoted by Σ^{j+1}, and is called the *handover proof.* A *succession* $\mathbb{S} = (\Sigma^1, \Sigma^2, \dots, \Sigma^j)$ at an epoch j is the sequence of all handover proofs across an execution until the beginning of the epoch.

In addition to the handover signature, at the beginning of each epoch, honest committee members sign the *state commitment* corresponding to their ledger. When the verifier learns the latest committee, these signatures enable him to find the current state commitment.

A naive linear client. Consider a PoPoS protocol, where each honest prover gives the verifier a state commitment and signatures on the commitment from the latest sync committee S^{N-1}, where N is the number of epochs (and $N-1$ is the last epoch). To convince the verifier that S^{N-1} is the correct latest committee, each prover also shares the sync committees $S^0 \dots S^{N-2}$ and the associated handover proofs in its view. The verifier knows S_0 from the genesis state st_0, and can verify the committee members of the future epochs iteratively through the handover proofs. Namely, upon obtaining the sync committee S^j, the verifier accepts a committee S^{j+1} as the correct committee assigned to epoch $j+1$, if there are signatures on

[4] Handover signatures between PoS epochs were introduced in the context of PoS sidechains [31]. Some practical blockchain systems already implement similar handover signatures [57, 45].

[5] This assumption can be satisfied using key-evolving signatures [32, 21], social consensus [8], or a static honest majority assumption.

the tuple $(j+1, S^{j+1})$ from over half of the committee members in S^j. Repeating the process above, the verifier can identify the correct committee for the last epoch. After identifying the latest sync committee, the verifier checks if the state commitment provided by a prover is signed by over half of the committee members. If so, he accepts the commitment.

It is straightforward to show that this strawman PoPoS protocol (which we abbreviate as TLC) is secure (Def. 3) under the following assumptions:

1. The underlying PoS protocol satisfies safety and liveness.
2. The majority of the sync committee members are honest.

When all provers are adversarial, the verifier might not receive any state commitment from them. Even though generally at least one prover is assumed to be honest, the strawman protocol does not need this for the *correctness* of the commitment accepted by the verifier, since the verifier validates each sync committee assigned to consecutive epochs, and does not accept commitments not signed by over $\frac{m}{2}$ members of the latest committee.

Regrettably, the strawman protocol is $\mathcal{O}(|\mathcal{C}|)$ and not succinct: To identify the lastest sync committee, the verifier has to download each sync committee since the genesis block. In the rest of this paper, we will improve this protocol to make it succinct.

The optimistic light client (OLC). We now reduce the communication complexity of the verifier. In this version of the protocol, instead of sharing the sync committees $S^0 \dots S^{N-2}$ and the associated handover proofs, each honest prover P sends a sequence of *hashes* $h^1 \dots h^{N-1}$ corresponding to the sync committees $S^0 \dots S^{N-1}$. Subsequently, to prove the correctness of the state commitment, the prover P reveals the latest sync committee S^{N-1} assigned to epoch $N-1$ and the signatures by its members on the commitment. Upon receiving the committee S^{N-1}, the verifier checks if the hash of S^{N-1} matches h^{N-1}, and validates the signatures on the commitment.

Unfortunately, an adversarial prover P^* can claim an incorrect committee $S^{*,N-1}$, whose hash $h^{*,N-1}$ disagrees with h^{N-1} returned by P. This implies a disagreement between the two hash sequences received from P and P^*. The verifier can exploit this discrepancy to identify the truthful party that returned the correct committee. Towards this goal, the verifier iterates over the two hash sequences, and finds the *first point of disagreement*. Let j be the index of this point such that $h^j \neq h^{*,j}$ and $h^i = h^{*,i}$ for all $i < j$. The verifier then requests P to reveal the committees S^j and S^{j-1} at the preimage of h^j and h^{j-1}, and to supply a handover proof Σ^j for S^{j-1} and S^j. He also requests P^* to reveal the committees $S^{*,j}$ and $S^{*,j-1}$ at the preimage of $h^{*,j}$ and $h^{*,j-1}$, and to supply a handover proof $\Sigma^{*,j}$ for $S^{*,j-1}$ to $S^{*,j}$. As $h^{j-1} = h^{*,j-1}$ by definition, the verifier is convinced that the committees S^{j-1} and $S^{*,j-1}$ revealed by P and P^* are the same.

Finally, the verifier checks whether the committees $S^{*,j}$ and S^j were inaugurated by the previous committee S^{j-1} using the respective handover proofs Σ^j and $\Sigma^{*,j}$. Since S^{j-1} contains over $\frac{m}{2}$ honest members that signed only the correct committee S^j assigned to epoch j, adversarial prover P^* cannot create a handover proof with sufficiently many signatures inaugurating $S^{*,j}$. Hence, the handover from S^{j-1} to $S^{*,j}$ will not be ratified $\Sigma^{*,j}$, whereas the handover from S^{j-1} to S^j will be ratified by Σ^j. Consequently, the verifier will identify P as the truthful party and accept its commitment.

In the protocol above, security of the commitment obtained by the prover relies crucially on the existance of an honest prover. Indeed, when all provers are adversarial, they can collectively return the *same* incorrect state commitment and the *same* incorrect sync committee for the latest epoch. They can then provide over $\frac{m}{2}$ signatures by this committee on

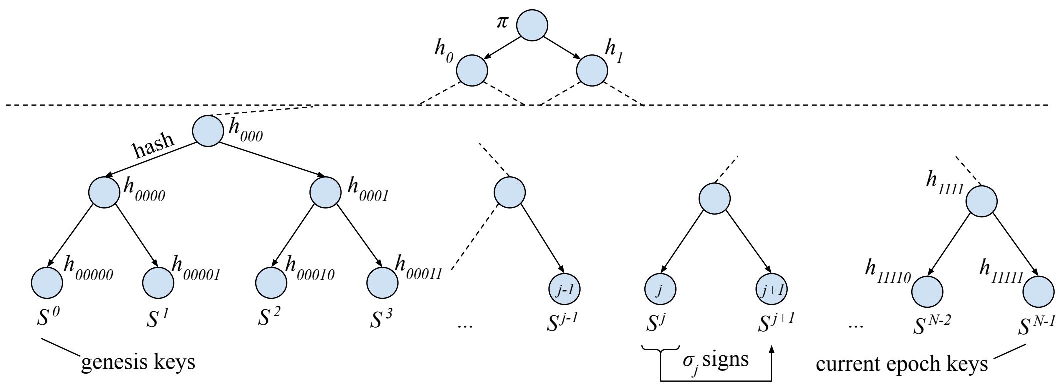

Figure 1 The *handover tree*, the central construction of our protocol. The root of the Merkle tree is the initial proof π. During the bisection game, the signatures between the challenge node j and its neighbours $j-1$ and $j+1$ are validated.

the incorrect commitment. In the absence of an honest prover to challenge the adversarial ones, the verifier would believe in the validity of an incorrect commitment.

The optimistic light client reduces the communication load of sending over the whole sync committee sequence by representing each committee with a constant size hash. However, it is still $\mathcal{O}(|\mathcal{C}|)$ as the verifier has to do a linear search on the hashes returned by the two provers to identify the first point of disagreement. To support a truly succinct verifier, we will next work towards an interactive PoPoS protocol based on bisection games.

5 The Superlight Client

Trees and mountain ranges. Before describing the succinct PoPoS protocol and the superlight client, we introduce the data structures used by the bisection games.

Suppose the number of epochs N is a power of two. The honest provers organize the committee sequences for the past epochs into a Merkle tree called the *handover tree* (Fig. 1). The j^{th} leaf of the handover tree contains the committee S^j of the j^{th} epoch. A handover tree consisting of leaves $S^0, \dots, S^{N-1}$ is said to be *well-formed* with respect to a succession $\mathbb{S}$ if it satisfies the following properties:

1. The leaves are syntactically valid. Every j^{th} leaf contains a sync committee S^j that consists of m public keys.
2. The first leaf corresponds to the known *genesis* sync committee S^0.
3. For each $j = 1 \dots N-1$, Σ^j consists of over $\frac{m}{2}$ signatures by members of S^{j-1} on (j, S^j).

Every honest prover holds a succession of handover signatures attesting to the inauguration of each sync committee in its handover tree after S^0. These successions might be different for every honest prover as any set of signatures larger than $\frac{m}{2}$ by S^j can inaugurate S^{j+1}. However, the trees are the same for all honest parties, and they are well-formed with respect to the succession held by each honest prover.

When the number N of epochs is not a power of two, provers arrange the past sync committees into Merkle mountain ranges (MMRs) [50, 23]. An MMR is a list of Merkle trees, whose sizes are decreasing powers of two. To build an MMR, a prover first obtains a binary representation $2^{q_1} + \dots + 2^{q_n}$ of N, where $q_1 > \dots > q_n$. It then divides the sequence of sync committees into n subsequences, one for each q_i. For $i \geq 1$, the i^{th} subsequence contains the committees $S^{\sum_{n=1}^{i-1} 2^{q_i}}, \dots, S^{(\sum_{n=1}^{i} 2^{q_i})-1}$. Each i^{th} subsequence is organized into a distinct Merkle tree $\mathcal{T}_i$, whose root, denoted by $\langle \mathcal{T}_i \rangle$, is called a *peak*. These peaks are all hashed together to obtain the root of the MMR. We hereafter refer to the index of each leaf in these Merkle trees with the epoch of the sync committee contained at the leaf. (For instance, if there are two trees with sizes 4 and 2, the leaf indices in the first tree are $0, 1, 2, 3$ and the leaf indices in the second tree are 4 and 5.) The MMR is said to be *well formed* if each constituent tree is well-formed (but, of course, only the first leaf of the first tree needs to contain the genesis committee). To ensure succinctness, only the peaks and a small number of leaves, with their respective inclusion proofs, will be presented to the verifier during the following bisection game.

Different state commitments. We begin our construction of the full PoPoS protocol (abbreviated SLC for *Super Light Client*) by describing the first messages exchanged between the provers $\mathcal{P}$ and the verifier. Each honest prover first shares the state commitment signed by the latest sync committee at the beginning of the last epoch $N-1$. If all commitments received by the verifier are the same, by existential honesty, the verifier rests assured this commitment is correct, *i.e.*, it corresponds to the ledger of the honest provers at the beginning of the epoch. Otherwise, the verifier requests from each prover in $\mathcal{P}$: (i) the MMR peaks $\langle \mathcal{T} \rangle_i$, $i \in [n]$ held by the prover, where n is the number of peaks, (ii) the latest sync committee S^{N-1}, (iii) a Merkle inclusion proof for S^{N-1} with respect to the last peak $\langle \mathcal{T} \rangle_n$, and (iv) signatures by the committee members in S^{N-1} on the state commitment given by the prover.

Upon receiving these messages, the verifier first checks if there are more than $\frac{m}{2}$ valid signatures by the committee members in S^{N-1} on the state commitment. It then verifies the Merkle proof for S^{N-1} with respect to $\langle \mathcal{T} \rangle_n$. As the majority of the committee members in S^{N-1} are honest, it is not possible for different state commitments to be signed by over half of S^{N-1}. Hence, if the checks above succeed for two provers P and P^* that returned different commitments, one of them (P^*) must be an adversarial prover, and must have claimed an incorrect sync committee $S^{*,N-1}$ for the last epoch. Moreover, as the Merkle proofs for both $S^{*,N-1}$ and S^{N-1} verify against the respective peaks $\langle \mathcal{T} \rangle_n$ and $\langle \mathcal{T} \rangle_n^*$, these peaks must be different. Since the two provers disagree on the roots and there is only one well-formed MMR at any given epoch, therefore one of the provers does not hold a well-formed MMR. This reduces the problem of identifying the correct state commitment to detecting the prover that has a well-formed MMR behind its peaks.

Bisection game. To identify the honest prover with the well-formed MMR, the verifier (Alg. 1) initiates a bisection game between P and P^* (Alg. 2). Suppose the number of epochs N is a power of two. Each of the two provers claims to hold a tree with size N (otherwise, since the verifier knows N by his local clock, the prover with a different size Merkle tree loses the game.) During the game, the verifier aims to locate the first point of disagreement between the alleged sync committee sequences at the leaves of the provers' Merkle trees, akin to the improved optimistic light client (Sec. 4).

The game proceeds in a binary search fashion similar to refereed delegation of computation [12, 11, 33]. Starting at the Merkle roots $\langle \mathcal{T} \rangle$ and $\langle \mathcal{T} \rangle^*$ of the two trees, the verifier traverses an identical path on both trees until reaching a leaf with the same index. This leaf corresponds to the first point of disagreement. At each step of the game, the verifier asks the provers to reveal the children of the *current* node, denoted by h_c and h_c^* on the respective trees (Alg. 2, l. 4). Initially, $h_c = \langle \mathcal{T} \rangle$ and $h_c^* = \langle \mathcal{T} \rangle^*$ (Alg. 1, l. 2). Upon receiving the alleged left and right child nodes h_0^* and h_1^* from P^*, and h_0, h_1 from P, he checks if

Algorithm 1 Run by the verifier during the bisection game to identify the first point of disagreement between the provers' leaves (*cf.* Fig. 2). Here, P and P^* denote the honest and adversarial provers, whereas $\langle\mathcal{T}\rangle$ and $\langle\mathcal{T}\rangle^*$ denote the roots of their respective Merkle trees with size ℓ.

```
1:  function FindDisagreement($P, \langle\mathcal{T}\rangle, P^*, \langle\mathcal{T}\rangle^*, \ell$)
2:      $h_c, h_c^* \leftarrow \langle\mathcal{T}\rangle, \langle\mathcal{T}\rangle^*$
3:      while $\ell > 1$ do
4:          $(h_0, h_1) \dashleftarrow P$; $(h_0^*, h_1^*) \dashleftarrow P^*$
5:          if $h_c \neq H(h_0 \,\|\, h_1)$ then return                          ▷ $P$ loses
6:          if $h_c^* \neq H(h_0^* \,\|\, h_1^*)$ then return                    ▷ $P^*$ loses
7:          if $h_0 \neq h_0^*$ then
8:              $h_c^*, h_c \leftarrow h_0^*, h_0$; $(\text{open}, 0) \dashrightarrow P$; $(\text{open}, 0) \dashrightarrow P^*$
9:          else
10:             $h_c^*, h_c \leftarrow h_1^*, h_1$; $(\text{open}, 1) \dashrightarrow P$; $(\text{open}, 1) \dashrightarrow P^*$
11:         $\ell \leftarrow \ell // 2$
12:     $S \dashleftarrow P$; $S^* \dashleftarrow P^*$
13:     return $S, S^*$
```

Algorithm 2 Run by an honest prover during the bisection game to reply to the verifier V's queries. The sequence $S^0, \ldots, S^{N-1}$ denotes the sync committees in the prover's view.

```
1:  function ReplyToVerifier($S^0, \ldots, S^{N-1}$)
2:      $\mathcal{T} \leftarrow \text{MakeMT}(S^0, \ldots, S^{N-1})$; $\mathcal{T}.\text{root} \dashrightarrow V$; $j \leftarrow 0$
3:      while $\mathcal{T}.\text{size} > 1$ do
4:          $(\mathcal{T}.\text{left.root}, \mathcal{T}.\text{right.root}) \dashrightarrow V$; $(\text{open}, i) \dashleftarrow V$
5:          if $i = 0$ then
6:              $\mathcal{T} \leftarrow \mathcal{T}.\text{left}$
7:          else
8:              $\mathcal{T} \leftarrow \mathcal{T}.\text{right}$
9:          $j \leftarrow 2j + i$
10:     $S^j \dashrightarrow V$
```

$h_c = H(h_0 \,\|\, h_1)$ and $h_c^* = H(h_0^* \,\|\, h_1^*)$, where H is the collision-resistant hash function used to construct the Merkle trees (Alg. 1, ll. 5 and 6). The verifier then compares h_0 with h_0^*, and h_1 with h_1^* to determine if the disagreement is on the left or the right child (Alg. 1, ll. 7 and 9). Finally, he descends into the first disagreeing child, and communicates this decision to the provers (Alg. 2, l. 4); so that they can update the current node that will be queried in the next step of the bisection game (Alg. 2, ll. 6 and 8).

Upon reaching a leaf at some index j, the verifier asks both provers to reveal the alleged committees S^j and $S^{*,j}$ at the pre-image of the respective leaves. If $j = 1$, he inspects whether S^j or $S^{*,j}$ matches S^0. The prover whose alleged first committee is not equal to S^0 loses the game.

If $j > 1$, the verifier also requests from the provers (i) the committees at the $(j-1)^{\text{th}}$ leaves, (ii) their Merkle proofs with respect to $\langle\mathcal{T}\rangle$ and $\langle\mathcal{T}\rangle^*$, and (iii) the handover proofs Σ^j and $\Sigma^{*,j}$. The honest prover responds with (i) S^{j-1} assigned to epoch $j-1$, (ii) its Merkle proof with respect to $\langle\mathcal{T}\rangle$, and (iii) its own view of the handover proof Σ^j (which might be different from other provers). Upon checking the Merkle proofs, the verifier is now convinced that the committees S^{j-1} and $S^{*,j-1}$ revealed by P and P^* are the same, since their hashes match. The verifier subsequently checks if Σ^j contains more than $\frac{m}{2}$ signatures by the committee members in S^{j-1} on (j, S^j), and similarly for P^*.

The prover that fails any of checks by the verifier loses the bisection game. If one prover

Algorithm 3 Run by the verifier to identify the first different peak in the MMRs of the two provers. Here, $\langle\mathcal{T}\rangle_{1,\dots,n}$ and $\langle\mathcal{T}\rangle^*_{1,\dots,n}$ denote the peaks of the honest and adversarial provers respectively.

```
1:  function BisectionGame($P, P^*$)
2:      $\langle\mathcal{T}\rangle_{1,\dots,n} \dashleftarrow P$
3:      $\langle\mathcal{T}\rangle^*_{1,\dots,n} \dashleftarrow P^*$
4:      for $i = 1$ to $n$ do
5:          if $\langle\mathcal{T}\rangle_i \neq \langle\mathcal{T}\rangle^*_i$ then
6:              $\ell \leftarrow$ size of the $i^{\text{th}}$ Merkle Tree
7:              return FindDisagreement($P, \langle\mathcal{T}\rangle_i, P^*, \langle\mathcal{T}\rangle^*_i, \ell$)
```

Algorithm 4 Tournament conducted by the verifier among provers $\mathcal{P}$ to identify the state commitment $\langle\mathsf{st}\rangle$. The verifier uses BisectionGame (*cf.* Fig. 2, Algs. 1 and 3) between two provers and deduce at most *one* winner. Here, pop removes and returns an arbitrary element of a set.

```
1:  function Tournament($\mathcal{P}$)
2:      $P \leftarrow \text{pop}(\mathcal{P})$; good $\leftarrow \{P\}$; $\langle\mathsf{st}\rangle \leftarrow P.\langle\mathsf{st}\rangle$
3:      for $P \in \mathcal{P}$ do
4:          if $\langle\mathsf{st}\rangle = P.\langle\mathsf{st}\rangle$ then
5:              good $\leftarrow$ good $\cup \{P\}$
6:              continue
7:          for $P^* \in$ good do
8:              if BisectionGame($P, P^*$) $= P$ then
9:                  good $\leftarrow \{P\}$; $\langle\mathsf{st}\rangle \leftarrow P.\langle\mathsf{st}\rangle$
10:                 break
11:     return $\langle\mathsf{st}\rangle$
```

loses the game, and the other one does not fail any checks, the standing prover is designated the winner. If neither prover fails any of the checks, then the verifier concludes that there are over $\frac{m}{2}$ committee members in S^{j-1} that signed different future sync committees (*i.e.*, signed both (j, S^j) and $(j, S^{*,j})$, where $(j, S^j) \neq (j, S^{*,j})$). This implies S^{j-1} is not the correct sync committee assigned to epoch $j-1$, and both provers are adversarial. In this case, both provers lose the bisection game. In any case, at most one prover can win the bisection game.

Bisection games on Merkle mountain ranges. When the number of epochs N is not a power of two, the verifier first obtains the binary decomposition $\sum_{i=1}^{n} 2^{q_i} = N$, where $q_1 > \ldots > q_n$. Then, for each prover P, he checks if there are n peaks returned. For two provers P and P^* that have n peaks but returned different commitments, the verifier compares the peaks $\langle\mathcal{T}\rangle_i$ of P with $\langle\mathcal{T}\rangle^*_i$ of P^*, and identifies the first different peak (Alg. 3). It then plays the bisection game as described above on the identified Merkle trees. The only difference with the game above is that if the disagreement is on the first leaf j of a later tree, then the Merkle proof for the previous leaf $j-1$ is shown with respect to the peak of the previous tree.

Prover complexity. Given all past sync committees, the prover constructs the MMR in linear time. The MMR is updated in an online fashion as time evolves. Every time a new sync committee appears, it is appended to the tree in $\log N$ time. The space required to store the MMR is linear.

Tournament. When there are multiple provers, the verifier interacts with them sequentially in pairs, in a tournament fashion. It begins by choosing two provers P_1 and P_2 with different

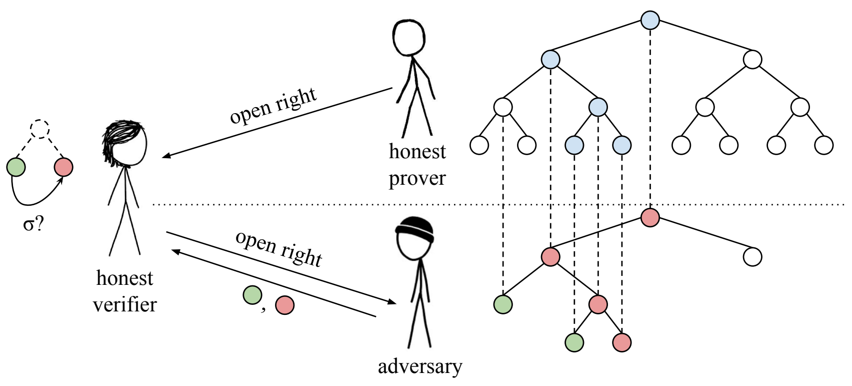

Figure 2 Honest and adversarial prover in the PoPoS bisection game (*cf.* Algs. 1, 2 and 3). The verifier iteratively requests openings of tree nodes from both provers, until the first point of disagreement is discovered.

state commitments from the set $\mathcal{P}$ (Alg. 4, l. 7). The verifier then *pits one against the other*, by facilitating a bisection game between P_1 and P_2, and decides which of the two provers loses (Alg. 4, l. 8). (There can be at most one winner at any bisection game). He then eliminates the loser from the tournament, and chooses a new prover with a different state commitment than the winner's commitment from the set $\mathcal{P}$ to compete against the winner. In the event that both provers lose, the verifier eliminates both provers, and continues the tournament with the remaining ones by sampling two new provers with different state commitments. This process continues until all provers left have the same state commitment. This commitment is adopted as the correct one. A tournament started with q provers terminates after $O(q)$ bisection games, since at least one prover is eliminated at the end of each game. In App. C, we prove the security of the tournament by showing that an honest prover never loses the bisection game and an adversarial prover loses against an honest one.

Past and future. Now that the verifier obtained the state commitment signed for the most recent epoch, and confirmed its veracity, the task that remains is to discern facts about the system's state and its history. To perform queries about the current state, such as determining how much balance one owns, the verifier simply asks for Merkle inclusion proofs into the proven state commitment.

One drawback of our protocol is that the state commitment received by the verifier is the commitment at the *beginning* of the current epoch, and therefore may be somewhat stale. In order to synchronize with the latest state within the epoch, the verifier must function as a full node for the small duration of an epoch. This functionality does not harm succinctness, since epochs have a fixed, constant duration. For example, in the case of a longest-chain blockchain, the protocol works as follows. In addition to signing the state commitment, the sync committee also signs the first stable block header of its respective epoch. The block header is verified by the verifier in a similar fashion that he verified the state commitment. Subsequently, the block header can be used as a *neon genesis* block. The verifier treats the block as a replacement for the genesis block and bootstraps from there[6].

One aspect of wallets that we have not touched upon concerns the retrieval and verification

[6] While bootstrapping, the verifier can update the state commitment by applying the transactions within the later blocks on top of the state commitment from the neon genesis block via the function $\langle\delta\rangle$.

of historical transactions. Consider a client that wishes to verify the inclusion of a particular historical transaction tx in the chain. Let's assume that tx is included in a block B of epoch j. This can be checked as follows. The verifier, as before, identifies the root of the correct handover tree. The verifier next asks the prover to provide him with the sync committee of epoch $j+1$, with the corresponding inclusion proof, as well as the first stable block header B' of that epoch signed by the committee. Subsequently, he requests the short blockchain that connects B to B'. As blockchains are hash chains, this inclusion cannot be faked by an adversary.

6 Proof-of-Stake Ethereum Light Clients

The bisection games presented in Sec. 5 can be applied to a variety of PoS consensus protocols to efficiently catch up with current consensus decisions. In this section we present an instantiation for Ethereum. We also detail how to utilize the latest epoch committee to build a full-featured Ethereum JSON-RPC. This allows for existing wallets such as MetaMask to use our construction without making any changes. Our implementation can be a drop-in replacement to obtain better decentralization and performance.

Our PoPoS protocol for Ethereum does not require any changes to the consensus layer, as Ethereum already provisions for sync committees in the way we introduced in Sec. 4.

6.1 Sync Committee Essentials

Sync committees of Ethereum contain $m = 512$ validators, sampled uniformly at random from the validator set, in proportion to their stake distribution. Every sync committee is selected for the duration of a so-called *sync committee period* [25] (which we called *epoch* in our generic construction). Each period lasts 256 Ethereum epochs (these are different from our epochs), approximately 27 hours. Ethereum epochs are further divided into *slots*, during which a new block is proposed by one validator and signed by the subset of validators assigned to the slot. At each slot, each sync committee member of the corresponding period signs the block at the tip of the chain (called the *beacon chain* [25]) according to its view. The proposer of the next slot aggregates and includes within its proposal the aggregate sync committee signature on the parent block. The sync committees are determined one period in advance, and the committee for each period is contained in the block headers of the previous period. Each block also contains a commitment to the header of the last finalized block that lies on its prefix.

6.2 Linear-Complexity Light Client

Light clients use the sync committee signatures to detect the latest beacon chain block finalized by the Casper FFG finality gadget [9, 10]. At any round, the view of a light client consists of a finalized_header, the current sync committee and the next committee. The client updates its view upon receiving a LightClientUpdate object (update for short), that contains (i) an attested_header signed by the sync committee, (ii) the corresponding aggregate BLS signature, (iii) the slot at which the aggregate signature was created, (iv) the next sync committee as stated in the attested_header, and (v) a finalized_header (called the new finalized header for clarity) to replace the one held by the client.

To validate an update, the client first checks if the aggregate signature is from a slot larger than the finalized_header in its view, and if this slot is within the current or the next period. (Updates with signatures from sync committees that are more than one period in the future are rejected.) It then verifies the inclusion of the new finalized header and the next sync committee provided by the update with respect to the state of the attested_header through Merkle inclusion proofs. Finally, it verifies the aggregate signature on the attested_header by the committee of the corresponding period. Since the signatures are either from the current period or the next one, the client knows the respective committee.

After validating the update, the client replaces its finalized_header with the new one, if the attested_header was signed by over 2/3 of the corresponding sync committee. If this header is from a higher period, the client also updates its view of the sync committees. Namely, the old next sync committee becomes the new current committee, and the next sync committee included in the attested_header is adopted as the new next sync committee.

6.3 Logarithmic Bootstrapping from Bisection Games

The construction above requires a bootstrapping light client to download at least one update per period, imposing a linear communication complexity in the life time of the chain. To reduce the communication load and complexity, the optimistic light client and superlight client constructions introduced in Secs. 4 and 5 can be applied to Ethereum.

A bootstrapping superlight client first connects to a few provers, and asks for the Merkle roots of the handover trees (*cf.* Sec. 5). The leaf of the handover tree at position j consist of all the public keys of the sync committee of period j concatenated with the period index j. If all the roots are the same, then the client accepts the sync committee at the last leaf as the most recent committee. If the roots are different, the client facilitates bisection games among conflicting provers. Upon identifying the first point of disagreement between two trees (*e.g.*, some leaf j), the client asks each prover to provide a LightClientUpdate object to justify the handover from the committee S^{j-1} to S^j. For this purpose, each prover has to provide a valid update that includes (i) an aggregate signature by 2/3 of the set S^{j-1} on an attested_header, and (ii) the set S^j as the next sync committee within the attested_header. Upon identifying the honest prover, and the correct latest sync committee, the client can ask the honest prover about the lastest update signed by the latest sync committe and containing the tip of the chain.

6.4 Superlight Client Architecture

On the completion of bootstrapping, the client has identified the latest beacon chain block-header. The blockheader contains the commitment to the state of the Ethereum universe that results from executing all transactions since genesis up to and including the present block. Furthermore, this commitment gets verified as part of consensus. The client can perform query to the fullnode about the state of Ethereum. The result of the query can be then verified against the state commitment using Merkle inclusion proofs. This allows for the client to access the state of the Ethereum universe in a trust-minimizing way.

Fig. 3 depicts the resulting architecture of the superlight client. In today's Ethereum, a user's wallet typically speaks to Ethereum JSON-RPC endpoints provided by either a centralized infrastructure provider such as Infura or by a (trusted) Ethereum full node (could be self-hosted). Instead, the centerpiece in a superlight client is a shim that provides RPC endpoints to the wallet, but where new transactions and queries to the Ethereum state are proxied to upstream full nodes, and query responses are verified w.r.t. a given commitment to the Ethereum state. This commitment is produced using two sidecar processes, which implement the prover and verifier of the bisection game. For this purpose, the server-side

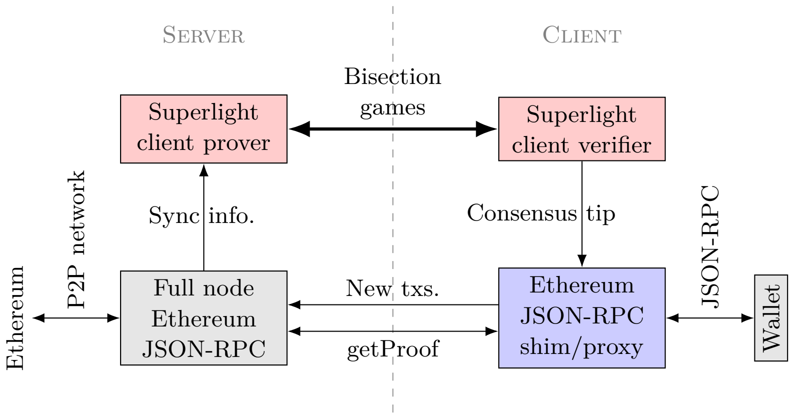

Figure 3 Ethereum superlight client architecture: On server side, an Ethereum full node feeds sync information to a bisection game prover sidecar. On client side, a bisection game verifier feeds the consensus tip into an Ethereum JSON-RPC shim/proxy, which forwards transactions coming from the wallet to the Ethereum full node, and resolves state queries with reference to the established consensus tip using Ethereum's getProof RPC endpoint.

sidecar obtains the latest sync information from a full node, using what is commonly called 'libp2p API'. The client-side sidecar feeds the block header at the consensus tip into the shim.

7 Experiments

To assess the different bootstrapping mechanisms for Ethereum (traditional light client = TLC; optimistic light client = OLC; superlight client = SLC), we implemented them in ≈ 2000 lines of TypeScript code (source code available on Github[7]). We demonstrate an improvement of SLC over TLC of 9× in time-to-completion, 180× in communication bandwidth, and 30× in energy consumption, when bootstrapping after 10 years of consensus execution. SLC improves over OLC by 3× in communication bandwidth in this setting.

7.1 Setup

Our experimental scenario includes seven malicious provers, one honest prover, and a verifier. All provers run in different Heroku '`performance-m`' instances located in the '`us`' region. The verifier runs on an Amazon EC2 '`m5.large`' instance located in '`us-west-2`'. The provers' Internet access is not restricted beyond the hosting provider's limits. The verifier's down- and upload bandwidth is artificially rate-limited to 100 Mbit/s and 10 Mbit/s, respectively, using '`tc`'. We monitor to rule out spillover from RAM into swap space.

In preprocessing, we create eight valid traces of the sync committee protocol for an execution horizon of 30 years. For this purpose, we create 512 cryptographic identities per simulated day, as well as the aggregate signatures for handover from one day's sync committee to the next day's. In some experiments, we vary how much simulated time has passed since genesis, and for this purpose truncate the execution traces accordingly. One of the execution traces is used by the honest prover and understood to be the true honest

[7] The superlight client prototype is at `https://github.com/lightclients/poc-superlight-client`. The optimistic light client implementation is at `https://github.com/lightclients/kevlar` and `https://kevlar.sh/`. The RPC shim is at `https://github.com/lightclients/patronum`.

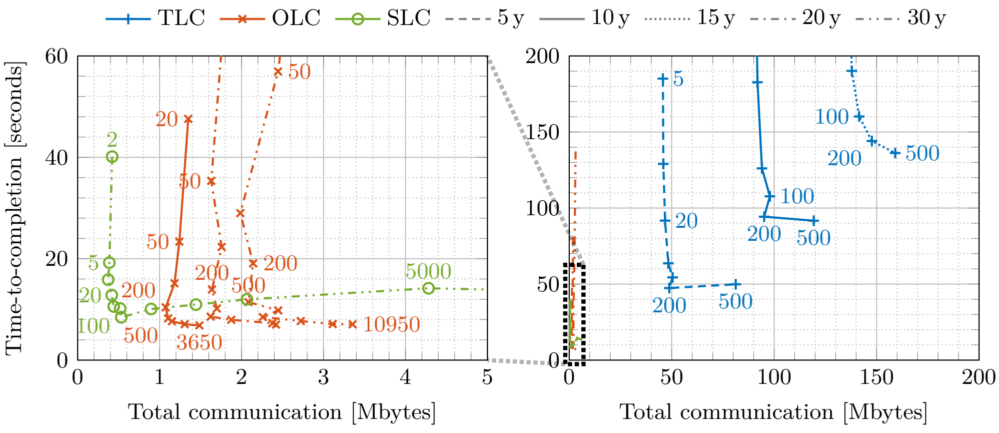

Figure 4 Time-to-completion and total communication (averaged over 5 trials) incurred by different light clients for varying internal parameters (marker labels; TLC/OLC: batch size b, SLC: Merkle tree degree d) and varying consensus execution horizon. For SLC, the curves for the considered execution horizons are virtually identical; thus, only the curve for 30 years (most challenging scenario) is shown. Pareto-optimal tradeoffs are at 'tip' of resulting 'L-shape': for 10 years execution, at $b \approx 200$ (TLC), $b \approx 500$ (OLC), and $d \approx 100$ (SLC), respectively. OLC and SLC vastly outperform TLC, *e.g.*, for 10 years execution: 9× in time-to-completion, 180× in bandwidth. In this setting, SLC has similar time-to-completion as OLC, and 3× lower communication.

execution. Adversarial provers each pick a random point in time, and splice the honest execution trace up to that point together with one of the other execution traces for the remaining execution time, *without regenerating handover signatures*, so that the resulting execution trace used by adversarial provers has invalid handover at the point of splicing. We also vary the internal parameters of the (super-)light client protocols (*i.e.*, batch size b of TLC and OLC, Merkle tree degree d of SLC).

7.2 Time-To-Completion & Total Verifier Communication

The average time-to-completion (TTC) and total communication bandwidth (TCB) required by the different light client constructions per bootstrapping occurrence is plotted in Fig. 4 for varying internal parameters (batch sizes b for TLC and OLC; Merkle tree degrees d for SLC) and varying execution horizons (from 5 to 30 years). Pareto-optimal TTC and TCB are achieved for b and d resulting 'at the tip' of the 'L-shaped' plot. For instance, for 10 years execution, TLC, OLC and SLC achieve Pareto-optimal TTC/TCB for $b \approx 200$, $b \approx 500$, and $d \approx 100$, respectively. Evidently, across a wide parameter range, OLC and SLC vastly outperform TLC in both metrics; *e.g.*, for 10 years execution and Pareto-optimal parameters, 9× in TTC, and 180× in TCB. In this setting, SLC has similar TTC as OLC, and 3× lower TCB (5× lower TCB for 30 years). For a closed-form expression describing the trade-off between latency and bandwidth, and the optimal choice of tree degree see Tas et al. [49].

The fact that both TLC and OLC have TCB linear in the execution horizon, is readily apparent from Fig. 4. The linear TTC is visible for TLC, but not very pronounced for OLC, due to the concretely low proportionality constant. In comparison, SLC shows barely any dependence of TTC or TCB on the execution horizon, hinting at the (exponentially better) logarithmic dependence. To contrast the asymptotics, we plot average TTC as a function of exponentially increasing execution horizon in Fig. 5 for OLC and SLC with internal parameters $b = 20$ and $d = 2$, respectively. Note that these are not Pareto-optimal

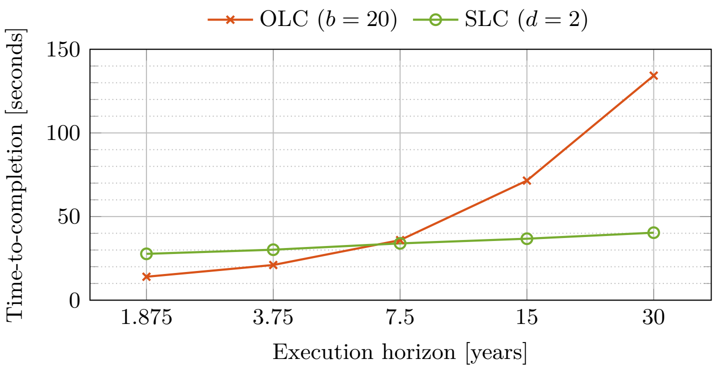

Figure 5 Time-to-completion (averaged over 5 trials) of OLC/SLC increase linearly/logarithmically with the execution horizon, respectively.

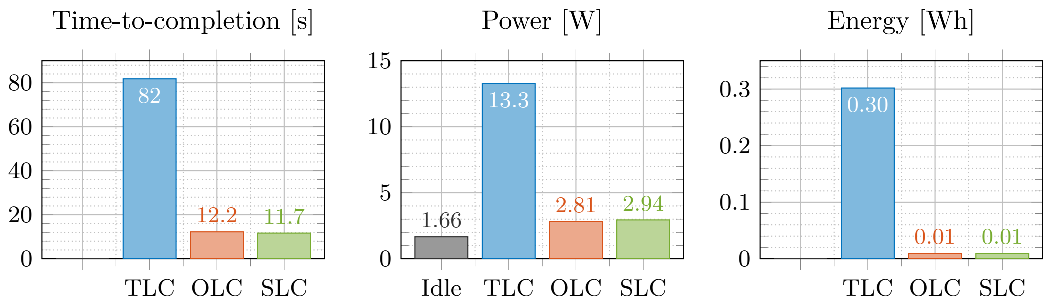

Figure 6 Energy required to bootstrap after 10 years of consensus execution using different light client constructions (averaged over 5 trials for TLC, 25 trials for OLC and SLC; internal parameters $b = 200$, $b = 500$, $d = 100$, respectively); also disaggregated into power consumption and time-to-completion. Energy required by OLC/SLC is 30× lower than TLC. Contributions $\approx 4\times$ and $\approx 7\times$ can be attributed to lower power consumption and lower time-to-completion, respectively.

parameters, but chosen here for illustration purposes. Clearly, TTC for OLC is linear in the execution horizon (plotted in Fig. 5 on an exponential scale), while for SLC it is logarithmic.

7.3 Power & Energy Consumption

A key motivation for superlight clients is their application on resource-constrained platforms such as browsers or mobile phones. In this context, computational efficiency, and as a proxy energy efficiency, is an important metric. We ran the light clients on a battery-powered System76 Lemur Pro (‘`lemp10`’) laptop with Pop!_OS 22.04 LTS, and recorded the decaying battery level using ‘`upower`’ (screen off, no other programs running, no keyboard/mouse input, WiFi connectivity; provers still on Heroku instances). From the energy consumption and wallclock time we calculated the average power consumption. As internal parameters for TLC, OLC, and SLC, we chose $b = 200$, $b = 500$, and $d = 100$, respectively (*cf.* Pareto-optimal parameters in Fig. 4). The energy required to bootstrap 10 years of consensus execution, averaged over 5 trials for TLC, and 25 trials for OLC and SLC, is plotted in Fig. 6. We disaggregate the energy consumption into power consumption and TTC for each light client, and also record the power consumption of the machine in idle. (Note, discrepancies in Figs. 4 and 6 are due to the light clients running on Amazon EC2 vs. a laptop.)

OLC and SLC have comparable TTC and power consumption, resulting in comparable energy consumption per bootstrap occurrence. The energy required by OLC and SLC is 30×

lower than the energy required by TLC per bootstrap occurrence (right panel in Fig. 6). This can be attributed to a $\approx 4\times$ lower power consumption (middle panel in Fig. 6) together with a $\approx 7\times$ lower TTC (left panel in Fig. 6). The considerably lower energy/power consumption of OLC/SLC compared to TLC is due to the lower number of signature verifications (and thus lower computational burden). Note that a sizeable fraction of OLC's/SLC's power consumption can be attributed to system idle (middle panel in Fig. 6). When comparing light clients in terms of *excess* energy consumption (*i.e.*, subtracting idle consumption) per bootstrapping, then OLC and SLC improve over SLC by $64\times$.

8 Analysis

The theorems for succinctness and security of the PoPoS protocol are provided below. Proofs are in App. C. Security consists of two components: completeness and soundness.

▶ **Theorem 4** (Succinctness). *Consider a verifier that invokes a bisection game at round r between two provers that provided different handover tree roots. Then, the game ends in $O(\log(r))$ steps of interactivity and has a total communication complexity of $O(\log(r))$.*

▶ **Theorem 5** (Completeness). *Consider a verifier that invokes a bisection game at round r between two provers that provided different handover tree roots. Suppose one of the provers is honest. Then, the honest prover wins the bisection game.*

▶ **Theorem 6** (Soundness). *Let H^s be a collision resistant hash function. Consider a verifier that invokes a bisection game executed at round r of a secure underlying PoS protocol between two provers that provided different handover tree roots. Suppose one of the provers is honest, and the signature scheme satisfies existential unforgeability. Then, for all PPT adversarial provers $\mathcal{A}$, the prover $\mathcal{A}$ loses the bisection game against the honest prover with overwhelming probability in λ.*

▶ **Theorem 7** (Tournament Runtime). *Consider a tournament ran at round r with $|\mathcal{P}|$ provers one of which is honest. The tournament ends in $O(|\mathcal{P}|\log(r))$ steps of interactivity, and has total communication complexity $O(|\mathcal{P}|\log(r))$.*

▶ **Theorem 8** (Security). *Let H^s be a collision resistant hash function. Consider a tournament executed between an honest verifier and $|\mathcal{P}|$ provers at round r. Suppose one of the provers is honest, the signature scheme satisfies existential unforgeability, and the PoS protocol is secure. Then, for all PPT adversaries $\mathcal{A}$, the state commitment obtained by the verifier at the end of the tournament satisfies state security with overwhelming probability in λ.*

References

1 Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. 2014. `https://blockstream.com/sidechains.pdf`.

2 Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *CCS*, pages 913–930. ACM, 2018.

3 Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. 2020.

4 Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Paper 2020/352, 2020. URL: `https://eprint.iacr.org/2020/352`.

5 Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus, 2018. `arXiv:1807.04938v3`.

6 Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-light clients for cryptocurrencies. In *IEEE Symposium on Security and Privacy*, pages 928–946. IEEE, 2020.

7 Vitalik Buterin. A next-generation smart contract and decentralized application platform. 2014.

8 Vitalik Buterin. Proof of Stake: How I Learned to Love Weak Subjectivity, Nov 2014. URL: `https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/`.

9 Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget, 2017. `arXiv:1710.09437v4`.

10 Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining ghost and casper, 2020. `arXiv:2003.03052v3`.

11 Ran Canetti, Ben Riva, and Guy N. Rothblum. Practical delegation of computation using multiple servers. In *CCS*, pages 445–454. ACM, 2011.

12 Ran Canetti, Ben Riva, and Guy N. Rothblum. Refereed delegation of computation. *Inf. Comput.*, 226:16–36, 2013.

13 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186. USENIX Association, 1999.

14 Pyrros Chaidos and Aggelos Kiayias. Mithril: Stake-based threshold multisignatures. Cryptology ePrint Archive, Paper 2021/916, 2021. URL: `https://eprint.iacr.org/2021/916`.

15 Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. Sok: Blockchain light clients. In *Financial Cryptography*, volume 13411 of *LNCS*, pages 615–641. Springer, 2022.

16 Jing Chen and Silvio Micali. Algorand, 2016. `arXiv:1607.01341v9`.

17 ConsenSys. MetaMask Surpasses 10 Million MAUs, Making It The World's Leading Non-Custodial Crypto Wallet, Aug 2021. URL: `https://consensys.net/blog/press-release/metamask-surpasses-10-million-maus-making-it-the-worlds-leading-non-custodial-crypto-wallet/`.

18 Rasmus Dahlberg, Tobias Pulls, and Roel Peeters. Efficient sparse merkle trees - caching strategies and secure (non-)membership proofs. In *NordSec*, volume 10014 of *LNCS*, pages 199–215, 2016.

19 Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. Cryptology ePrint Archive, Paper 2016/919, 2016. URL: `https://eprint.iacr.org/2016/919`.

20 Stelios Daveas, Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. A gas-efficient superlight bitcoin client in solidity. In *AFT*, pages 132–144. ACM, 2020.

21 Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)*, volume 10821 of *LNCS*, pages 66–98. Springer, 2018.

22 Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725, 2019.

23 Grin Developers. Merkle Mountain Ranges (MMR). URL: `https://docs.grin.mw/wiki/chain-state/merkle-mountain-range/`.

24 Ethereum Developers. Altair Light Client – Light Client, 2023. URL: `https://github.com/ethereum/consensus-specs/blob/5c64a2047af9315db4ce3bd0eec0d81194311e46/specs/altair/light-client/light-client.md`.

25 Ethereum Developers. Altair Light Client – Sync Protocol, 2023. URL: `https://github.com/ethereum/consensus-specs/blob/e9f1d56807d52aa7425f10160a45cb522345468b/specs/altair/light-client/sync-protocol.md`.

26 Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

27 Ariel Gabizon, Kobi Gurkan, Philipp Jovanovic, Georgios Konstantopoulos, Asa Oines, Marek Olszewski, Michael Straka, Eran Tromer, and Psi Vesely. Plumo: Towards scalable interoperable blockchains using ultra light validation systems. 2020. URL: `https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-plumo_celolightclient.pdf`.

28 Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. Cryptology ePrint Archive, Paper 2014/765, 2014. URL: `https://eprint.iacr.org/2014/765`.

29 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, volume 9057 of *LNCS*, pages 281–310. Springer, 2015.

30 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO (1)*, volume 10401 of *LNCS*, pages 291–323. Springer, 2017.

31 Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *IEEE Symposium on Security and Privacy*, pages 139–156. IEEE, 2019.

32 Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In *CRYPTO*, volume 2139 of *LNCS*, pages 332–354. Springer, 2001.

33 Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium*, pages 1353–1370. USENIX Association, 2018.

34 Kostis Karantias. Sok: A taxonomy of cryptocurrency wallets. Cryptology ePrint Archive, Paper 2020/868, 2020. URL: `https://eprint.iacr.org/2020/868`.

35 Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Compact storage of superblocks for nipopow applications. In *MARBLE*, pages 77–91. Springer, 2019.

36 Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Proof-of-burn. In *Financial Cryptography*, volume 12059 of *LNCS*, pages 523–540. Springer, 2020.

37 Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

38 Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In *Financial Cryptography Workshops*, volume 9604 of *LNCS*, pages 61–78. Springer, 2016.

39 Aggelos Kiayias, Nikos Leonardos, and Dionysis Zindros. Mining in logarithmic space. In *CCS*, pages 3487–3501. ACM, 2021.

40 Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work. In *Financial Cryptography*, volume 12059 of *LNCS*, pages 505–522. Springer, 2020.

41 Aggelos Kiayias, Andrianna Polydouri, and Dionysis Zindros. The velvet path to superlight blockchain clients. In *AFT*, pages 205–218. ACM, 2021.

42 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO (1)*, volume 10401 of *LNCS*, pages 357–388. Springer, 2017.

43 Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *Financial Cryptography Workshops*, volume 11599 of *LNCS*, pages 21–34. Springer, 2019.

44 Jae Kwon and Ethan Buchman. A network of distributed ledgers – cosmos whitepaper. URL: `https://v1.cosmos.network/resources/whitepaper`.

45 Rongjian Lan, Ganesha Upadhyaya, Stephen Tse, and Mahdi Zamani. Horizon: A gas-efficient, trustless bridge for cross-chain transactions, 2021. `arXiv:2101.06000v1`.

46 Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, volume 293 of *LNCS*, pages 369–378. Springer, 1987.

47 Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT (2)*, volume 10211 of *LNCS*, pages 643–673, 2017.

48 Succinct Labs. Building the end game of interoperability with zkSNARKs, 2023. URL: `https://www.succinct.xyz/`.

49 Ertem Nusret Tas, Dionysis Zindros, Lei Yang, and David Tse. Light clients for lazy blockchains. Cryptology ePrint Archive, Paper 2022/384, 2022. URL: `https://eprint.iacr.org/2022/384`.

50 Peter Todd. Merkle mountain ranges, oct 2012. URL: `https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md`.

51 Jason Wise. Metamask Statistics 2023: How Many People Use Metamask?, Mar 2023. URL: `https://earthweb.com/metamask-statistics/`.

52 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. 2014.

53 Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In *CCS*, pages 3003–3017. ACM, 2022.

54 Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356. ACM, 2019.

55 Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt. Sok: Communication across distributed ledgers. In *Financial Cryptography (2)*, volume 12675 of *LNCS*, pages 3–36. Springer, 2021.

56 Alexei Zamyatin, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Edgar R. Weippl, and William J. Knottenbelt. A wild velvet fork appears! inclusive blockchain protocol changes in practice - (short paper). In *Financial Cryptography Workshops*, volume 10958 of *LNCS*, pages 31–42. Springer, 2018.

57 Maksym Zavershynskyi. ETH-NEAR Rainbow Bridge, Aug 2020. URL: `https://near.org/blog/eth-near-rainbow-bridge/`.

A Security of Ethereum Light Clients

The following assumptions ensure the security of the optimistic light client and superlight client on Ethereum:

1. The honest Ethereum validators constitutes at least $\frac{2}{3} + \epsilon$ fraction of the validator set at all times.
2. The sync committee for each period is sampled uniformly at random from the validator set.
3. The underlying PoS consensus protocol satisfies security.
4. The attested_header of a beacon block containing a finalized_header is signed by a sync committee member *only if* the finalized_header is the header of a Casper FFG finalized PoS block in the view of the sync committee member.
5. Honest block proposers include the latest Casper FFG finalized block in their view as the finalized_header of their proposal blocks.

The assumptions (a) and (b) ensure that the honest sync committee members constitute a supermajority of the sync committee at all periods. Assumption (d) ensures that any header obtained by a light client belongs to a Casper FFG finalized block, whereas assumption (e) ensures that upon being finalized, these blocks are soon adopted by the light clients through the light client updates. Together with (c), these assumptions and Thm. 8 imply the security of our optimistic light client and superlight client constructions for Ethereum per Def. 3.

Security under adversarial network conditions. Due to network delays or temporary adversarial majorities, there might be extended periods during which the light client does not receive any updates. In this case, if the client observes that UPDATE_TIMEOUT number of slots have passed since the slot of the last finalized_header in its view, it can do a *force update*. Prior to the force update, the client replaces the finalized_header within the best valid light client update in its view with the attested_header of the same update. Note that the finalized_header of the best valid update must have had a smaller slot than the finalized_header in the client's view, as it could not prompt the client to update its view during the last UPDATE_TIMEOUT slots. Hence, treating the attested_header within the best valid update, which is by definition from a higher slot, as a finalized_header can enable the client to adopt it as the latest finalized_header block, and facilitate the client's progression into a later sync committee period.

The current Ethereum specification [25] also recommends using other use-case dependent heuristics for updates, in lieu of checking signatures, if the light client seems stalled. However, heuristics such as swapping the attested and finalized headers as described above might cause the light client to adopt block headers that are not finalized by Casper FFG. Hence, in this work, we assume that the underlying consensus protocol is not subject to disruptions like network delays, and focus on the regular update mechanism described in Sec. 6.

B Discussion and Future Work

Interactivity. The interactivity in our protocol is undesirable. Contrary to linear bootstrapping protocols in which the bottleneck in practice comes down to the bandwidth needed to download all the data, in our protocol, because the communication complexity we have achieved is so low, the practical bottleneck comes down to the interactivity. This interactivity is inherent to our construction and is difficult to remove. In contrast to standard cryptography proof protocols (such as zero-knowledge proofs), our interactivity cannot be removed using the Fiat–Shamir heuristic [26]. The reason here is that we require two provers to challenge each other under the watchful eyes of the verifier. Such behavior cannot be emulated by the random sampling that Fiat–Shamir would offer, in which a *single* prover and *single* verifier interaction is emulated. However, even though the protocol has inherent logarithmic interactivity, in practice the actual number of interaction rounds can be reduced with a couple of techniques:

1. The tournament between multiple provers can involve bisection games played in parallel. If the verifier speaks to 8 provers, once every prover commits to his tree root, the verifier can have the provers compete with each other in 4 bisection games running simultaneously over the network, to reduce latency. The 4 remaining provers can play against each other in 2 bisection games running in parallel, and so forth. Although this does not decrease the number of interactions, the actual network delay will be significantly reduced as compared to challenging the parties one-by-one.

Despite the above optimizations, the asymptotic interaction remains $\mathcal{O}(\log n)$. The problem of *Non-Interactive* Proof of Proof-of-Stake (NIPoPoS), PoPoS that can run in just a single interaction, is left for future work.

Trusted setup. Our protocol was developed in the standard model and does not require a trusted setup. The underlying proof-of-stake protocol *does* have some trusted setup assumption: The genesis epoch contains the public keys of a committee who, at the time of the epoch, were assumed to have an honest majority. The fact that our protocol does not leverage that trust to build zero knowledge or other constructions has certain advantages. We only use hashes and signatures. This makes the protocol easy to understand and straightforward to implement, with a limited attack surface, a great advantage in security-critical protocols. But also, in case the trusted setup assumption is violated, our protocol can readily provide evidence of what exactly happened, because it does not rely on that assumption: At the exact point where two provers disagree at the bisection game, the evidence presented illustrates two different epochs at the same index accompanied by their different randomnesses, a situation that should not occur. In contrast, a zero-knowledge proof in case of trusted setup failure can lead to decisions in which foul play may be undetectable, and certainly the exact point and conditions of first failure are difficult to pinpoint. Additionally, the simplicity of our construction makes it possible to implement very efficient provers and verifiers in practice. Despite the slightly worse asymptotics, we expect our construction to perform better in concrete terms than zero knowledge proof systems that feature large constants in their complexities.

Proof-of-Work extensions. Our tree-based protocol cannot be readily extended to proof-of-work. To see why, consider a "chain tree" similar to our handover tree, but storing the proof-of-work chain in its leaves. One could naïvely imagine that two provers can run a similar bisection protocol to find the first difference in this tree. But note that, contrary to proof-of-stake epochs, there can be multiple trees that are *admissible* here, and they are not prefixes of one another. For example, if a mining adversary chooses to mine a secret chain starting somewhere in the middle of the honest chain, she will end up with a "chain tree" whose first difference with the honest tree will be the first secret block. At that point, the honest verifier has nothing to see; both trees look equally valid when the leaves are revealed. Therefore, this scheme cannot be readily applied to *chains*, but must remain constructed on top of *epochs*, as we chose to do. The closest related proof-of-work construction is FlyClient [6] in which leaves are opened at random until foul play is detected, but no bisection games are played.

C Proofs

Proof of Thm. 4. Let $N \in \Theta(r)$ be the number of epochs at round r. When the handover trees have N leaves, there can be at most $\log N \in \Theta(\log r)$ steps of interactivity during the bisection game. In case an adversarial prover attempts to continue beyond $\log N$ steps of interactivity, the verifier aborts the interaction early, as the verifier expects to receive sync committees after $\log N$ queries, and the number N is known by the verifier.

At each step of the bisection game until the sync committees are revealed, the verifier receives two children (two constant size hash values) of the queried node from both provers. At the final step, the verifier receives the sync committees S^j and $S^{*,j}$ from the provers at the first point of disagreement j, and the sync committees S^{j-1} and $S^{*,j-1}$ at the preceding leaf along with their Merkle proofs. As each committee consists of a constant number m of public keys with constant size, and each Merkle proof contains $\log N$ constant size hash values, the total communication complexity of the bisection game becomes $\Theta(\log N) = \Theta(\log(r))$. ◀

Proof of Thm. 5. To show that the honest prover wins the bisection game, we will step through the conditions checked by the verifier during the bisection game.

At the start of the game, the honest prover and verifier both agree on the number of past epochs N. By synchrony, the honest prover does not time out and replies to all of the *open* queries sent by the verifier. As the honest prover's handover tree is well-formed, at each *open* query asking the honest prover to reveal the children of a node h_c on its tree, the left and the right children h_l and h_r returned by the honest prover satisfy the relation $h_c = H(h_l \parallel h_r)$. Thus, the replies are always syntactically valid and accepted by the verifier. Subsequently, upon reaching a leaf, the honest prover supplies a sync committee, as expected by the verifier.

Suppose the first point of disagreement between the leaves of the honest and the adversarial prover is identified at some index j. If $j = 0$, the honest prover returns S^0, which is validated by the verifier as the correct sync committee supplied by the genesis state st_0.

If $j > 0$, the honest prover reveals the sync committee S^j at leaf j, the committee S^{j-1} at leaf $j-1$, and the Merkle inclusion proof for S^{j-1}, which is validated by the verifier with respect to the root. By the well-formedness of the honest prover's handover tree, the prover holds a handover proof Σ^j that contains over $m/2$ signatures on (j, S^j) by unique committee members within S^{j-1}. The honest prover sends this valid handover proof to the verifier. Consequently, the honest prover passes all of the verifier's checks, and wins the bisection game. ◀

Let Verify be the verification function for Merkle proofs. It takes a proof π, a Merkle root $\langle\mathcal{T}\rangle$, the size of the tree ℓ, an index for the leaf $0 \leq i < \ell$ and the leaf v itself. It outputs 1 if π is valid and 0 otherwise. We assume that the well-formed Merkle trees built with a collision-resistant hash function satisfy the following collision-resistance property:

▶ **Proposition 9** (Merkle Security [49]). *Let H^s be a collision resistant hash function used in the binary Merkle trees. For all PPT $\mathcal{A}$:* $\Pr[(v, D, \pi, i) \leftarrow \mathcal{A}(1^\lambda) : \langle\mathcal{T}\rangle = \textsc{MakeMT}(D).\text{root} \wedge D[i] \neq v \wedge \textsc{Verify}(\pi, \langle\mathcal{T}\rangle, |D|, i, v) = 1] \leq negl(\lambda)$.

The following lemma shows that the sync committees at the first point of disagreement identified by the verifier are different, and the committees at the previous leaf are the same with overwhelming probability.

▶ **Lemma 10** (Bisection Pinpointing). *Let H^s be a collision resistant hash function. Consider the following game among an honest prover P, a verifier V and an adversarial prover P^*: The prover P receives an array D of size N from P^*, and calculates the corresponding Merkle tree $\mathcal{T}$ with root $\langle\mathcal{T}\rangle$. Then, V mediates a bisection game between P^* claiming root $\langle\mathcal{T}\rangle^*$ and P with $\langle\mathcal{T}\rangle$. Finally, V outputs $(1, D^*[j-1], D^*[j])$ if P^* wins the bisection game; otherwise, it outputs $(0, \perp, \perp)$. Here, $D^*[j-1]$ and $D^*[j]$ are the two entries revealed by P^* for the consecutive indices $j-1$ and j during the bisection game. ($D^*[-1]$ is defined as $\perp$ if $j = 0$.) Then, for all PPT adversarial provers $\mathcal{A}$,* $\Pr[D \leftarrow \mathcal{A}(1^\lambda); (1, D^*[j-1], D^*[j]) \leftarrow (V(|D|) \leftrightarrow (P(D), \mathcal{A})) \wedge (D^*[j-1] \neq D[j-1] \vee D^*[j] = D[j])] \leq \text{negl}(\lambda)$.

The above lemma resembles [49, Lemma 4] and its proof is given below:

Proof of Lem. 10. Consider an adversary $\mathcal{A}(1^\lambda)$ such that $(1, D^*[j-1], D^*[j]) \leftarrow (V(|D|) \leftrightarrow (P(D), \mathcal{A})) \wedge (D^*[j-1] \neq D[j-1] \vee D^*[j] = D[j])$. We next construct an adversary $\mathcal{A}_m$ that uses $\mathcal{A}$ as a subroutine to break Merkle security.

The verifier starts the bisection game by asking the provers to reveal the children of the roots $\langle \mathcal{T} \rangle$ and $\langle \mathcal{T} \rangle^*$ of the respective handover trees, where $\langle \mathcal{T} \rangle \neq \langle \mathcal{T} \rangle^*$. Subsequently, at every step of the bisection game, the verifier asks each prover to reveal the two children of a previously revealed node, where the queried nodes have the same position, yet different values in the respective trees. Hence, for the index j identified by the verifier as the first point of disagreement, it holds that $D^*[j] \neq D[j]$. Since $D^*[-1] = D[-1] = \perp$, for $j = 0$, $\Pr[D \leftarrow \mathcal{A}(1^\lambda); (1, D^*[j-1], D^*[j]) \leftarrow (V(|D|) \leftrightarrow (P(D), \mathcal{A})) \wedge (D^*[j-1] \neq D[j-1] \vee D^*[j] = D[j])] = 0$.

If $j > 0$, there exists a step in the bisection game, where the verifier asks the provers to open the right child of the previously queried node. Concretely, there exists a node $\tilde{h}_c$ on $\mathcal{T}$, queried by the verifier, and a node $\tilde{h}^*_c$, alleged by P^* to be at the same position as $\tilde{h}_c$, such that for the two children $\tilde{h}_l$ and $\tilde{h}_r$ of $\tilde{h}_c$ and the two children $\tilde{h}^*_l$ and $\tilde{h}^*_r$ of $\tilde{h}^*_c$ revealed to the verifier, the following holds: $\tilde{h}^*_l = \tilde{h}_l$ and $\tilde{h}^*_r \neq \tilde{h}_r$. Let's consider the last such nodes $\tilde{h}^e_c$ and $\tilde{h}^{e,*}_c$ after which, the verifier asks the provers to open only the left children of the subsequent nodes. Let $\tilde{h}^e_l$ denote the left child of $\tilde{h}^e_c$, which by definition equals the left child of $\tilde{h}^{e,*}_c$ alleged by the adversary. Let D' denote the sequence of leaves that lie within the subtree $\mathcal{T}'$ rooted at $\tilde{h}^e_l$. Note that the honest verifier knows the number of leaves, *i.e.*, $|D'|$, within the subtree $\mathcal{T}'$.

Consider the Merkle proofs π and π^* revealed for $D[j-1]$ and $D^*[j-1]$ with respect to $\langle \mathcal{T} \rangle$ and $\langle \mathcal{T} \rangle^*$ respectively. Let $b_a, b_{a-1}, \ldots, b_2, b_1$ denote the binary representation of $j-1$ from the most important bit to the least (The index of the first leaf is zero). Given $a := \log(|D|)$, the verifier can parse the Merkle proofs as $\pi = (h_1, h_2, \ldots, h_a)$ and $\pi^* = (h^*_1, h^*_2, \ldots, h^*_a)$. Since $(1, D^*[j-1], D^*[j]) \leftarrow (V(|D|) \leftrightarrow (P(D), \mathcal{A}))$, π^* verifies with respect to $\langle \mathcal{T} \rangle^*$:

- $h^{*,f}_1 := H(D^*[j-1])$.
- $h^{*,f}_{a+1} := \langle \mathcal{T} \rangle^*$.
- For $i = 1, \ldots, a$; $h^{*,f}_{i+1} := H(h^{*,f}_i, h^*_i)$ if $b_i = 0$, and $h^{*,f}_{i+1} := H(h^*_i, h^{*,f}_i)$ if $b_i = 1$.

Now, consider the prefix of the Merkle proof π^* consisting of the first $\log(|D'|)$ entries: $\pi^*_p = (h_1, \ldots, h_{\log(|D'|)})$. By definition of j, the indices $b_{\log(|D'|)}, \ldots, b_1$ are all 1, and;

- $h^{*,f}_1 = H(D^*[j-1])$.
- $h^{*,f}_{\log(|D'|)+1} = \tilde{h}^e_l$.
- For $i = 1, \ldots, \log(|D'|)$; $h^{*,f}_{i+1} = H(h^*_i, h^{*,f}_i)$.

Hence, it holds that $\textsc{Verify}(\pi^*_p, \tilde{h}^e_l, |D'|, |D'|-1, D^*[j-1]) = 1$. Moreover, $\tilde{h}^e_l = \textsc{MakeMT}(D').\text{root}$ and $D'[|D'|-1] = D[j-1] \neq D^*[j-1]$.

Finally, $\mathcal{A}_m$ uses $\mathcal{A}$ as a subroutine to generate D, π^* and $D^*[j-1]$, and outputs $(D^*[j-1], D', \pi^*_p, |D'|-1)$, which implies that $\tilde{h}^e_l := \textsc{MakeMT}(D').\text{root}$, $D'[|D'|-1] = D[j-1] \neq D^*[j-1]$ and $\textsc{Verify}(\pi^*_p, \tilde{h}^e_l, |D'|, |D'|-1, D^*[j-1]) = 1$. Consequently, by Prop. 9, for all PPT adversarial provers $\mathcal{A}$, $\Pr[D \leftarrow \mathcal{A}(1^\lambda); (1, D^*[j-1], D^*[j]) \leftarrow (V(|D|) \leftrightarrow (P(D), \mathcal{A})) \wedge (D^*[j-1] \neq D[j-1])] \leq \text{negl}(\lambda)$. As $D^*[j] \neq D[j]$, this implies that for all PPT adversarial provers $\mathcal{A}$, $\Pr[D \leftarrow \mathcal{A}(1^\lambda); (1, D^*[j-1], D^*[j]) \leftarrow (V(|D|) \leftrightarrow (P(D), \mathcal{A})) \wedge (D^*[j-1] \neq D[j-1] \vee D^*[j] = D[j])] \leq \text{negl}(\lambda)$. ◀

▶ **Definition 11** (Definition 13.1 of Boneh & Shoup [3]). *A signature scheme* $\mathbf{S} = (G, S, V)$ *is a triple of efficient algorithms,* G*,* S *and* V*, where* G *is called a key generation algorithm,* S

is called a signing algorithm, and V is called a verification algorithm. Algorithm S is used to generate signatures and algorithm V is used to verify signatures.

- *G is a probabilistic algorithm that takes no input. It outputs a pair (pk, sk), where sk is called a secret signing key and pk is called a public verification key.*
- *S is a probabilistic algorithm that is invoked as $\sigma \overset{R}{\leftarrow} S(sk, m)$, where sk is a secret key (as output by G) and m is a message. The algorithm outputs a signature σ.*
- *V is a deterministic algorithm invoked as $V(pk, m, \sigma)$. It outputs either accept or reject.*
- *We require that a signature generated by S is always accepted by V (valid for short). That is, for all (pk, sk) output by G and all messages m, we have $\Pr[V(pk, m, S(sk, m)) = \text{accept}] = 1$.*

We say that messages lie in a finite message space $\mathcal{M}$, signatures lie in some finite signature space Σ, and $\mathbf{S} = (G, S, V)$ is defined over $(\mathcal{M}, \Sigma)$.

▶ **Definition 12** (Attack Game 13.1 of Boneh & Shoup [3]). *For a given signature scheme $\mathbf{S} = (G, S, V)$, defined over $(\mathcal{M}, \Sigma)$, and a given adversary $\mathcal{A}$, the attack game runs as follows:*

- *The challenger runs $(pk, sk) \overset{R}{\leftarrow} G()$ and sends pk to $\mathcal{A}$.*
- *$\mathcal{A}$ queries the challenger several times. For $i = 1, 2, \ldots$, the i^{th} signing query is a message $m_i \in \mathcal{M}$. Given m_i, the challenger computes $\sigma_i \overset{R}{\leftarrow} S(sk, m_i)$, and then gives σ_i to $\mathcal{A}$.*
- *Eventually $\mathcal{A}$ outputs a candidate forgery pair $(m, \sigma) \in \mathcal{M} \times \Sigma$.*

We say that the adversary wins the game if the following two conditions hold:

- *$V(pk, m, \sigma) = \text{accept}$, and*
- *m is new, namely $m \notin \{m_1, m_2, \ldots\}$.*

We define $\mathcal{A}$'s advantage with respect to $\mathbf{S}$, denoted by $\text{SIGadv}[\mathcal{A}, \mathbf{S}]$, as the probability that $\mathcal{A}$ wins the game.

▶ **Definition 13** (Definition 13.2 of Boneh & Shoup [3]). *We say that a signature scheme $\mathbf{S}$ satisfies existential unforgeability under a chosen message attack (existential unforgeability for short) if for all efficient adversaries $\mathcal{A}$, the quantity $\text{SIGadv}[\mathcal{A}, \mathbf{S}]$ is negligible.*

Proof of Thm. 6. Consider the following game among an honest prover P, a verifier V and an adversarial prover $\mathcal{A}$: The prover P receives an array $D = (S^0, \ldots, S^{N-1})$ of sync committees from the underlying PoS protocol, and calculates the corresponding Merkle tree $\mathcal{T}$ with root $\langle \mathcal{T} \rangle$. Similarly, the prover P receives a succession of handover proofs $\mathbb{S} = (\Sigma^1, \Sigma^2, \ldots, \Sigma^{N-1})$, where for all $j = 1, \ldots, N-1$, Σ^j consists of over $m/2$ valid signatures on $(j+1, S^{j+1})$ by unique honest sync committee members assigned to epoch j. Then, V mediates a bisection game between $\mathcal{A}$ claiming root $\langle \mathcal{T} \rangle^*$ and P claiming root $\langle \mathcal{T} \rangle$. Finally, $\mathcal{A}$ wins the bisection game. In the subsequent proof, we will construct an adversary $\mathcal{A}_s$ that uses $\mathcal{A}$ as a subroutine to break the existential unforgeability of the signature scheme under a chosen message attack.

Let j denote the first point of disagreement between the leaves of the honest and the adversarial provers P and $\mathcal{A}$. If $j = 0$, let S^0 and $S^{*,0}$ denote the committees returned by the honest and adversarial provers respectively for the first leaf. By Lem. 10, $S^0 \neq S^{*,0}$. As the honest prover's tree is well-formed, S^0 is the sync committee within the genesis state st_0. Thus, in this case, $\mathcal{A}$ loses the bisection game, which implies $j > 0$.

During epoch $j-1$, the honest committee members assigned to epoch $j-1$ constitute over $m/2$ of the members within S^{j-1}, and create only a *single* handover signature on (j, Σ^j). After epoch $j-1$ ends, no PPT adversary can access the secret signing keys of the honest members of the committee S^{j-1} due to the use of key-evolving signatures.

Let $S^{*,j-1}$ and $S^{*,j}$ denote the sync committees revealed by $\mathcal{A}$ for the consecutive indices $j-1$ and j during the bisection game. Suppose $S^j \neq S^{*,j}$ and $S^{j-1} = S^{*,j-1}$. Since $\mathcal{A}$ wins the bisection game, it provides a handover proof $\Sigma^{*,j}$ that contains over $m/2$ signatures on $(j, S^{*,j})$ by unique committee members within $S^{*,j-1} = S^{j-1}$. Thus, there exists at least one committee member $S^{j-1}_{i^*}$ with the smallest index such that

- There is a signature $\sigma^{*,j-1}_{i^*}$ within the handover proof $\Sigma^{*,j}$ such that given the public verification key pk of $S^{j-1}_{i^*}$, $V(pk, (j, S^{*,j}), \sigma^{*,j-1}_{i^*}) = \text{accept}$.
- During epoch $j-1$, $S^{j-1}_{i^*}$ was an honest committee member assigned to epoch $j-1$.
- $S^{j-1}_{i^*}$ has created only a single handover signature $\sigma^{j-1}_{i^*}$ on (j, S^j) during epoch $j-1$.
- After epoch $j-1$ ends, no PPT adversary can access the secret signing key of $S^{j-1}_{i^*}$.

Consequently, $\mathcal{A}$ provides a signature $\sigma^{*,j-1}_{i^*}$ on $(j, S^{*,j})$ that verifies with respect to the public verification key of $S^{j-1}_{i^*}$.

Given the array of sync committees $D = (S^0, \dots, S^{N-1})$ from the underlying PoS protocol, we next construct an existential forgery adversary $\mathcal{A}_s$ that has access to the adversarial prover $\mathcal{A}$ as a subroutine. During $\mathcal{A}_s$'s interaction with $\mathcal{A}$, it receives signing queries from the adversarial and honest sync committee members within $S_0 \dots S_{N-1}$, and passes these queries to the challenger, which replies with the queried signatures. It then passes the signatures back to $\mathcal{A}$, and the succession of handover proofs $\mathbb{S} = (\Sigma^1, \Sigma^2, \dots, \Sigma^{N-1})$ to the honest prover P as specified at the beginning of the proof. Finally, $\mathcal{A}_s$ obtains the handover proofs Σ^j and $\Sigma^{*,j}$ from $\mathcal{A}$, and identifies $S^{j-1}_{i^*}$. It subsequently outputs $\sigma^{*,j-1}_{i^*}$ on the message $(j, S^{*,j})$, for which the following conditions hold:

- Given the public verification key pk of $S^{j-1}_{i^*}$, it holds that $V(pk, (j, S^{*,j}), \sigma^{*,j-1}_{i^*}) = \text{accept}$, and
- $(j, S^{*,j}) \neq (j, S^j)$, where unlike (j, S^j), the message $(j, S^{*,j})$ was not sent as a query to the challenger.

Thus, $\mathcal{A}_s$ wins the attack game in Def. 12.

Finally, if there is a PPT adversary $\mathcal{A}$ such that $\mathcal{A}$ wins the bisection game against the honest prover and the sync committees received by the verifier satisfies $S^j \neq S^{*,j}$ and $S^{j-1} = S^{*,j-1}$, $\mathcal{A}_s$ described above wins the attack game. By Lem. 10, for all PPT adversaries $\mathcal{A}$, $S^j \neq S^{*,j}$ and $S^{j-1} = S^{*,j-1}$ with overwhelming probability. Moreover, as the signature scheme satisfies existential unforgeability, for all PPT adversaries $\mathcal{A}$, the adversary $\mathcal{A}$ loses the attack game with overwhelming probability. Consequently, for all PPT adversarial provers $\mathcal{A}$, the prover $\mathcal{A}$ loses the bisection game against the honest prover with overwhelming probability in λ. ◀

Proof of Thm. 7. Consider a tournament started at round r with $|\mathcal{P}|$ provers, one of which is honest. At each step of the tournament, the verifier facilitates a bisection game between two provers with different state commitments. (Honest provers hold the same state commitment.) At the end of the game, at least one prover is designated as a loser and eliminated from the set of provers. The tournament continues until all remaining provers hold the same state commitment. Hence, it lasts at most $|\mathcal{P}| - 1$ steps. By Thm. 4, each bisection game at round r ends in $O(\log(r))$ steps of interactivity, and has a total communication complexity of $O(\log(r))$. Consequently, the tournament consists of $O(|\mathcal{P}|\log(r))$ steps of interactivity, and has a total communication complexity of $O(|\mathcal{P}|\log(r))$. ◀

Proof of Thm. 8. Consider a tournament step that involves an honest prover P and an adversarial prover P^* that have provided different state commitments $\langle \mathsf{st} \rangle$ and $\langle \mathsf{st} \rangle^*$ respectively, for the state of the blockchain at the beginning of the epoch containing round r. Let N denote the number of past epochs at round r (starting at epoch 0), and S^{N-1} denote the committee assigned to epoch $N-1$. Define n as the number of Merkle trees within the MMRs of the honest provers at epoch N, and let D_i denote the sequence of leaves within the i^{th} tree of the honest prover. Let $\langle \mathcal{T} \rangle_i^*$ and $\langle \mathcal{T} \rangle_i$, $i \in [n]$, denote the sequence of peaks revealed by P^* and P to the verifier before the bisection game. By definition, P returns S^{N-1} as the latest sync committee in its view, and let $S^{*,N-1}$ denote the latest sync committee alleged by P^*. The prover P sends over $m/2$ signatures on $\langle \mathsf{st} \rangle$ by unique committee members within S^{N-1}, whereas P^* sends over $m/2$ signatures on $\langle \mathsf{st} \rangle^*$ by unique committee members within $S^{*,N-1}$. Similarly, the prover P sends a Merkle proof π such that $\textsc{Verify}(\pi, \langle \mathcal{T} \rangle_n, |D_n|, |D_n|-1, S^{N-1}) = 1$, whereas P^* sends a Merkle proof π^* such that $\textsc{Verify}(\pi^*, \langle \mathcal{T} \rangle_n^*, |D_n|, |D_n|-1, S^{*,N-1}) = 1$. We first show that $S^{*,N-1} \neq S^{N-1}$ with overwhelming probability. We will then prove that if $S^{*,N-1} \neq S^{N-1}$, then $\langle \mathcal{T} \rangle_n^* \neq \langle \mathcal{T} \rangle_n$, with overwhelming probability.

To show that $S^{*,N-1} \neq S^{N-1}$, we construct an existential forgery adversary $\mathcal{A}_s$ that uses the adversarial prover P^* as a subroutine to break the existential unforgeability of the signature scheme under a chosen message attack. Suppose $S^{*,N-1} = S^{N-1}$. At the beginning of epoch $N-1$, the honest committee members assigned to epoch $N-1$ constitute over $m/2$ of the members within S^{N-1}, and create only a *single* signature on a state commitment, namely $\langle \mathsf{st} \rangle$. Since P^* sends over $m/2$ signatures on $\langle \mathsf{st} \rangle^*$ by unique committee members within S^{N-1}, there is at least one committee member $S_{i^*}^{N-1}$ within S^{N-1} with the smallest index such that

- There is a signature σ^* such that given the public verification key pk of $S_{i^*}^{N-1}$, it holds that $V(pk, \langle \mathsf{st} \rangle^*, \sigma^*) = \text{accept}$.
- During epoch $N-1$, $S_{i^*}^{N-1}$ is an honest committee member assigned to epoch $N-1$.
- $S_{i^*}^{N-1}$ has created only a single signature σ on a state commitment during epoch $N-1$, and that is on $\langle \mathsf{st} \rangle$.

Consequently, P^* provides a signature σ^* on $\langle \mathsf{st} \rangle^*$ that verifies with respect to the public verification key of $S_{i^*}^{N-1}$.

We next construct the adversary $\mathcal{A}_s$ that has access to the adversarial prover P^* as a subroutine. During $\mathcal{A}_s$'s interaction with P^*, it receives signing queries on state commitments from the adversarial and honest sync committee members within S_{N-1}, and passes these queries to the challenger, which replies with the queried signatures. It then passes the signatures back to P^* and P. Finally, $\mathcal{A}_s$ obtains $m/2$ signatures on the commitments $\langle \mathsf{st} \rangle$ and $\langle \mathsf{st} \rangle^*$ from $\mathcal{A}$, and identifies $S_{i^*}^{N-1}$. It subsequently outputs σ^* on the message $\langle \mathsf{st} \rangle^*$, for which the following conditions hold:

- Given the public verification key pk of $S_{i^*}^{N-1}$, it holds that $V(pk, \langle \mathsf{st} \rangle^*, \sigma^*) = \text{accept}$, and
- $\langle \mathsf{st} \rangle^* \neq \langle \mathsf{st} \rangle$, where unlike $\langle \mathsf{st} \rangle$, the message $\langle \mathsf{st} \rangle^*$ was not sent as part of a signing query to the challenger.

Thus, $\mathcal{A}_s$ wins the attack game in Def. 12.

Finally, if there is a PPT adversary P^* such that it gives $m/2$ signatures on $\langle \mathsf{st} \rangle^*$ by unique committee members within S^{N-1}, $\mathcal{A}_s$ described above wins the attack game. However, as the signature scheme satisfies existential unforgeability, for all PPT adversaries $\mathcal{A}$, the adversary $\mathcal{A}$ loses the attack game with overwhelming probability. Consequently, for all PPT adversarial provers P^*, it holds that $S^{*,N-1} \neq S^{N-1}$ with overwhelming probability.

Next, we show that if $S^{*,N-1} \neq S^{N-1}$, then $\langle \mathcal{T} \rangle_n^* \neq \langle \mathcal{T} \rangle_n$ with overwhelming probability. Towards this goal, we construct an adversary $\mathcal{A}_m$ that uses P^* as a subroutine to break Merkle security. Suppose $S^{*,N-1} \neq S^{N-1}$ and $\langle \mathcal{T} \rangle_n^* = \langle \mathcal{T} \rangle_n$. In this case, $\mathcal{A}_m$ receives from P^*, the set $S^{*,N-1}$, the sequence of leaves D_n within the last tree of the honest prover's MMR, the proof π^* and the index $|D_n| - 1$. It then outputs $(S^{*,N-1}, D_n, \pi^*, |D_n| - 1)$, for which it holds that $D_n[|D_n| - 1] = S^{N-1} \neq S^{*,N-1}$ and $\textsc{Verify}(\pi^*, \langle \mathcal{T} \rangle_n^*, |D_n|, |D_n| - 1, S^{*,N-1}) = \textsc{Verify}(\pi^*, \langle \mathcal{T} \rangle_n, |D_n|, |D_n| - 1, S^{*,N-1}) = 1$, where $\langle \mathcal{T} \rangle_n$ is the root of the last Merkle tree (that has the leaves D_n) within the honest prover's MMR. However, by Prop. 9, we know that for all PPT adversaries $\mathcal{A}$: $\Pr[(v, D, \pi, i) \leftarrow \mathcal{A}(1^\lambda) : \langle \mathcal{T} \rangle = \textsc{MakeMT}(D).\mathrm{root} \wedge D[i] \neq v \wedge \textsc{Verify}(\pi, \langle \mathcal{T} \rangle, |D|, i, v) = 1] \leq \mathrm{negl}(\lambda)$. Hence, for all PPT adversarial provers P^* with state commitment $\langle \mathsf{st} \rangle^* \neq \langle \mathsf{st} \rangle$, $S^{*,N-1} \neq S^{N-1}$, and the sequence of peaks $\langle \mathcal{T} \rangle_i^*$ revealed to the verifier by the adversarial prover is different from the sequence $\langle \mathcal{T} \rangle_i$, $i \in [n]$, revealed by the honest prover with overwhelming probability. Thus, there exists an index $d \in [n]$ such that $\langle \mathcal{T} \rangle_d^* \neq \langle \mathcal{T} \rangle_d$ and $\langle \mathcal{T} \rangle_i^* = \langle \mathcal{T} \rangle_i$ for all $i \in [n]$, $i < d$, with overwhelming probability. In this case, the verifier mediates a bisection game between P and P^* on the two alleged trees with the roots $\langle \mathcal{T} \rangle_d^*$ and $\langle \mathcal{T} \rangle_d$. By Thm. 5, P wins the game, and by Thm. 6, P^* loses the game with overwhelming probability. As a result, P^* is eliminated at this tournament step with overwhelming probability.

At each step of the tournament, at least one prover is eliminated, and the tournament continues until all remaining provers hold the same state commitment, with overwhelming probability. By assumption, there is at least one honest prover P. This prover emerges victorious from every tournament step against other provers with a different state commitment, except with negligible probability. Consequently, with overwhelming probability, P remains in the tournament until all remaining provers hold the same state commitment $\langle \mathsf{st} \rangle$ as P.

Let $\mathbb{L}$ be the ledger held by P at round r_0 corresponding to the beginning of the epoch of round r. By definition, $r_0 \leq r$ and $r - r_0 \leq C$ for some constant epoch length C. By the safety of the PoS protocol, for any honest parties P_1, P_2 and rounds $r_1 \geq r_2$: $\mathbb{L}_{r_2}^{P_2} \preccurlyeq \mathbb{L}_{r_1}^{P_1}$. Thus, for any honest party P' and rounds $r' \geq r \geq r_0$, it holds that $\mathbb{L} \preccurlyeq \mathbb{L}_{r'}^{P'}$, Similarly, for any honest party P', it holds that $\mathbb{L}_{r_0-1}^{P'} \preccurlyeq \mathbb{L}$. Consequently, there exists a latency parameter $\nu = K$, and a ledger $\mathbb{L}$ such that $\langle \mathsf{st} \rangle = \delta^*(\mathsf{st}_0, \mathbb{L})$, and $\mathbb{L}$ satisfies the following properties:

- **Safety:** For all rounds $r' \geq r + \nu$: $\mathbb{L} \preccurlyeq \mathbb{L}_{r'}^{\cup}$.
- **Liveness:** For all rounds $r' \leq r - \nu$: $\mathbb{L}_{r'}^{\cap} \preccurlyeq \mathbb{L}$.

Thus, $\langle \mathsf{st} \rangle$ satisfies state security. As the verifier accepts $\langle \mathsf{st} \rangle$ as the correct commitment at the end of the tournament with overwhelming probability, the commitment obtained by the verifier at the end of the tournament satisfies state security with overwhelming probability in λ. ◀

Safety under partial synchrony. We observe the assumption requiring at least one prover to be honest is essential for both the safety and liveness of our superlight client. Similarly proof of stake protocols without finality (e.g., Ouroboros, Ouroboros Praos) also lose safety when eclipsed. In contrast, proof of stake protocols that have finality (e.g., Ethereum's Casper FFG, Algorand, Tendermint, Streamlet, and HotStuff) maintain safety and only lose liveness when eclipsed.

D Other Proof-of-Stake Systems

We have presented our construction in a generic PoS model, and instantiated it concretely for Ethereum PoS. Our construction is quite general and can be adopted to virtually any PoS system. Many PoS systems are split into (potentially smaller) epochs in which some sampling from the underlying stake distribution is performed according to some random number. The random number generation can be performed in multiple ways. For example, all of Ouroboros [42], Ouroboros Praos [21], and Ouroboros Genesis [2] use a verifiable secret sharing mechanism, while Algorand [16] uses a multiparty computation. The stake distribution from which the sampling is performed could also have various nuances such as delegation, might require locking up one's funds, may exclude people with very small stake, or may give different weights to different stake ownership. In all of these cases, a frozen stake distribution from which the final sampling is performed is determined.

Our scheme can be generalized to any PoS scheme in which the leader can be verified from a frozen stake distribution and some randomness, no matter how it is generated, as long as the block associated with a particular slot can be uniquely determined after it stabilizes (a property that follows in any blockchain system as long as it observes the common prefix property). In the scheme we described throughout the paper, the sequence of signatures $\overline{\sigma}^{j+1}$ that are generated in an epoch j and vouch for the leaders of the next epoch sign the public key set S^{j+1} of the next epoch. To generalize our scheme to any PoS system with randomness and a stake distribution, the signatures S^{j+1} need not sign the public key sequence any more; instead, they can sign:

1. the epoch randomness η^{j+1} of the next epoch, and
2. the frozen stake distribution SD^j of the current epoch that will be used for sampling during the next epoch.

Of course, in such a scheme, a succinctness problem arises: The stake distribution SD^j might be large. However, this problem can be overcome by organizing the stake distribution SD^j into a Merkle tree. This Merkle tree contains one leaf for every satoshi (the smallest cryptocurrency denomination). The leaf's value is the public key who owns this satoshi. When sampling from SD^j according to the randomness η^{j+1}, the prover can provide a proof that the correct leader was the one that happened to be elected by opening the particular Merkle tree path at a particular index. That way, the verifier can deduce the last slot leaders of each epoch. Because the number of satoshis can be large, this Merkle tree can have a large (potentially an exponential) number of leaves. However, its root and proofs can be efficiently computed using sparse Merkle tree techniques [18] (or Merkle tries [52]) because the tree contains a polynomial number of continuous ranges in which many consecutive leaves share the same value. Even better, Merkle–Segment trees [6]) can be used. These trees are similar to Merkle trees, except that each node (internal or leaf) is also annotated with a numerical value, here the total stake under the subtree rooted at the particular node. Each internal node has the property that its annotated value is the sum of the annotated values of its children.

The above technique is quite generic, but each system has its nuances that must be accounted for.

Ouroboros/Cardano. Our construction can be implemented in Cardano/Ouroboros [42] as presented by electing a committee, but the underlying longest chain rule lends itself to better implementations for committee election and signature inclusion. One way to make use of the Cardano protocol is to extend the epoch duration R by $2k$ slots. In this manner, the randomness and leaders of the next epoch are known during the last $2k$ slots of the previous epoch (in the vanilla Cardano protocol, the leaders and randomness of the next epoch only become known at the end of the epoch). The last $2k$ slots of each epoch are then used to determine the sync committee. The committee is the leaders of these $2k$ slots, and no separate process is required to elect it. In each of the last $2k$ slots of an epoch, we add the extra requirement to the block validity rules that the block producer has included a handover signature to the correct next epoch committee; otherwise the block is rejected as invalid by full nodes.

These small changes mean that the Ouroboros protocol can be used almost as-is to support our PoPoS and do not require any additional mechanisms for electing committees or any off-chain mechanism for exchanging committee succession signatures, as the blocks themselves are used as carriers of this information. The critical property of the Ouroboros protocol that allows us to prove security in this setting is the following lemma:

▶ **Lemma 14** (Honest Subsequence). *Consider any continuous window of $2k$ slots within an epoch. If* any *$k + 1$ keys among these $2k$ are chosen, then at least one of them is guaranteed to be honest, except with negligible probability in k.*

Using the above lemma, we see that the last $2k$ slots will necessarily contain $k + 1$ honest leaders who will produce correct committee signatures, and so our PoPoS assumption that the committee has honest majority during its epoch is satisfied. The security of the protocol then follows from Thm. 8.

Ouroboros Praos and Genesis. These two protocols have some similarities to Ouroboros, but also significant differences. As Ouroboros Praos and Genesis are designed to be resilient to fully adaptive adversaries, the actual slot leader of each slot is not known *a priori*. However, a party can himself determine whether he is eligible to be the slot leader by evaluating a VRF on the epoch randomness and the current slot index using his private key. If the VRF output is below a certain threshold, determined by the candidate leader's stake, then the party is elegible to be a leader for this slot. The party's public key can then be used by others to verify a proof that the VRF computation was correct, and that he is indeed a rightful leader.

Because we cannot determine the leaders of the $j + 1^{\text{st}}$ epoch at the end of epoch j, we cannot hope to have the leaders of the j^{th} epoch sign off the public keys of the leaders of epoch $j + 1$. However, the above technique, in which signatures sign the randomness and a Merkle–Segment tree of the stake distribution, together with the VRF proof, suffices. In this construction, the signatures of epoch j sign off the randomness for epoch $j + 1$ and stake distribution Merkle tree for epoch j. At a later time, when it is revealed who is leader, the honest prover can provide the VRF proof to the verifier, and the verifier can check that the leader was indeed rightful. To obtain the VRF threshold, the prover can open the Merkle–Segment tree to the depth required to illustrate the total sum of the stake of the leader. Once the leader's stake is revealed, the threshold used in the VRF inequality is validated.

These protocols have several advantages, including security in the semi-synchronous setting as well as resilience to adaptive adversaries [21, 2].

Snow White. This protocol uses epochs and every epoch contains a randomness and a stake distribution from which leaders are sampled [19]. Therefore, our protocol can be readily adapted to it.

,

Algorand. Contrary to Ouroboros, Algorand offers immediate finality [16]. Once a block is broadcast, any transactions contained within are confirmed and can no longer be reverted. In other words, its common prefix property holds with a parameter of $k = 1$. To achieve this, Algorand runs a full Byzantine Agreement protocol for the generation of every block before moving to the next block. One way to look at it is to think of Algorand as a coin in which the epoch duration is $R = 1$. Our construction can therefore create a handover tree in which the leaves are exactly the blocks in the Algorand chain. The Algorand private sortition mechanism can be used to elect a committee large enough to ensure honest supermajority (a property required for Algorand's security). This committee, whose members can be placed in increasing order by their public key to ensure determinism, can then be used in place of our sequence of public keys, to sign off the results of the next block. Even though our handover tree now becomes slightly larger, with its number of leaves equal to the chain length $|\mathcal{C}|$, our protocol is still $\mathcal{O}(\log |\mathcal{C}|)$.