Light Clients for Lazy Blockchains

Ertem Nusret Tas[1], David Tse[1], Lei Yang[2], and Dionysis Zindros[1]

[1] Stanford University
{nusret,dntse,dionyziz}@stanford.edu
[2] MIT CSAIL
leiy@csail.mit.edu

Abstract. *Lazy* blockchains decouple consensus from transaction verification and execution to increase throughput. Although they can contain invalid transactions (*e.g.*, double spends) as a result, these can easily be filtered out by full nodes that check if there have been previous conflicting transactions. However, creating light (SPV) clients that do not see the whole transaction history becomes a challenge: A record of a transaction on the chain does not necessarily entail transaction confirmation. In this paper, we devise a protocol that enables the creation of efficient light clients for lazy blockchains. The number of interaction rounds and the communication complexity of our protocol are logarithmic in the blockchain execution time. Our construction is based on a bisection game that traverses the Merkle tree containing the ledger of all – valid or invalid – transactions. We prove that our proof system is succinct, complete and sound, and empirically demonstrate the feasibility of our scheme.

1 Introduction

A traveler in Naples saw twelve beggars lying in the sun. He offered a lira to the laziest of them. Eleven of them jumped up to claim it, so he gave it to the twelfth [41]. Towards scalable blockchains, the holy grail of cryptocurrency adoption, it has become clear that *lazy* systems will similarly win the race.

Eager blockchain protocols, such as Bitcoin and Ethereum, combine transaction verification and execution with consensus to ensure that only *valid* transactions are included in their ledger. In contrast, lazy blockchain protocols separate the consensus layer (responsible for ordering transactions) from the execution layer (responsible for interpreting them) to remove the execution bottleneck on scalability [3]. In these systems, a population of *consensus nodes* collects transactions and places them in a total order, without care for their validity. This produces a confirmed *dirty ledger*, a sequence of totally ordered, but potentially invalid, transactions – such as double spends. It is the responsibility of *full nodes* to *sanitize* the dirty ledger and ascertain which transactions are valid. This is done by executing the valid transactions one by one, and ignoring transactions that are not applicable. Examples of lazy distributed ledger protocols include Celestia (LazyLedger) [3], Prism [5], Parallel Chains [24] and Snap-and-Chat [39].

Authors are listed alphabetically. Contact author: DT

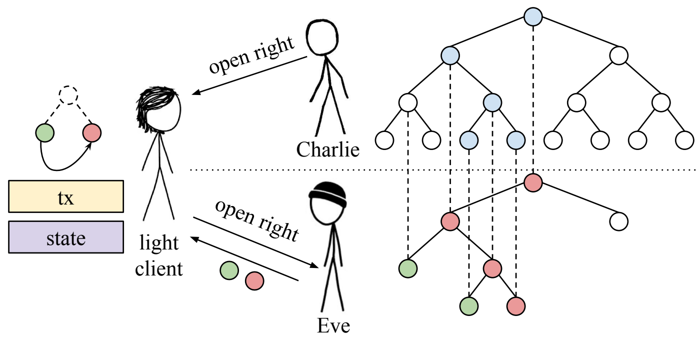

Fig. 1: Bisection Game. Charlie the challenger helps the light client iteratively traverse the tree of Eve the evil responder. A green node indicates a match, while a red node indicates a mismatch, between the two dirty trees.

As consensus nodes do not execute transactions, they also cannot find and include the state commitments in the blockchain. Hence, lazy protocols cannot easily support light clients, and techniques from the realm of eager systems, such as SPV (Simple Payment Verification [38,13]), are not applicable (*cf.* Appendix A for an attack on the succinctness of the SPV clients on lazy blockchains). For instance, in Ethereum, to prove to the light clients their current account balance, a full node presents a block header containing a state commitment, together with a Merkle inclusion proof of the account to be verified within the commitment. However, in a lazy protocol, commitments posted to the blockchain cannot be trusted since the consensus nodes do not check the validity of the state.

In this paper, we resolve this outstanding problem by introducing the first *light client for lazy blockchains*. Consider a light client, such as the mobile wallet of a vendor, wishing to confirm an incoming payment. Our construction allows it to synchronize with the network and quickly learn its latest account balance. Towards this purpose, the light client first connects to several full nodes (*e.g.*, servers by Infura, Chainlink, Alchemy), at least one of which is honest (existential honesty). It then asks them its account balance. If the answers received contradict each other, then it deduces that at least one of them is adversarial. It interactively interrogates the full nodes in order to determine which of them is truthful.

Lazy blockchains also appear in the context of optimistic rollups on Ethereum [30,23,21]. In these rollups, transactions are bundled and posted to Ethereum. Then, the rollup full nodes execute the transactions and send state commitments to a smart contract. If an invalid commitment appears on the contract, honest full nodes post fraud proofs to warn the light clients about the invalid state. To guarantee that they will see a fraud proof on-chain when the state is invalid, these clients wait for a *dispute delay* period, typically one week, before accepting a rollup state commitment. Thus, Ethereum acts like a lazy blockchain towards the pending rollup transactions and state commitments that are less than a week old. With our light client protocol used on these pending transactions, clients can sync with the latest rollup state within seconds.

Contributions. Our contributions are: (1) We put forth the first *light client* for *lazy blockchains*, achieving exponential improvement over full nodes in terms of communication and computational complexity (Section 3); (2) We show our system is *complete*, *sound*, and *succinct* with reduction-based proofs (Section 5); (3) We implement our scheme and measure its performance (Section 4).

Experiments show that our light client construction can be efficiently implemented on commodity mobile hardware, and only requires slight, incremental changes to blockchain full nodes serving these clients. Specifically, to synchronize with the network, a light client connecting to 17 full nodes distributed across the world only downloads a dozen MBs of data, as opposed to hundreds of GBs if running as a full node. The entire process takes less than 25 seconds.

1.1 Construction Overview

Consider a light client connected to two full nodes, Charlie and Eve. Charlie is honest and Eve is adversarial. The client begins by downloading the *canonical* (confirmed) header chain, each header containing the Merkle root of the transactions (valid and invalid) in its block. To find out its account balance, the client queries the full nodes. If both of them return the same answer, it rests assured that the balance reported is accurate, implying that the protocol terminates quickly in the optimistic case. Otherwise, it must identify the truthful party.

To help convince the light client, Charlie augments his dirty ledger with some extra information: Together with every transaction, he includes a state commitment after the particular transaction has been applied to the previous state. If a transaction is invalid, he does not update the state. He organizes this *augmented dirty ledger* into a binary Merkle tree, the *dirty tree*. All honest full nodes following this process will construct the same dirty tree and hold the same, correct dirty tree root (assuming they claim the same number of leaves that is a power of two). If the client somehow learns the *correct* dirty tree root, then it can be convinced of its balance with a Merkle proof. Thus, it suffices for the client to discover the correct root. (In practice, the dirty tree can be organized on the granularity of blocks rather than transactions; *cf.* Section 4.2.)

Charlie gives the correct dirty tree root to the light client, whereas Eve gives an incorrect root. Since the two roots are different, the underlying augmented dirty ledgers must differ somewhere. Charlie helps the client identify the first leaf in Eve's dirty tree that differs from his own via a *bisection game* (*cf.* Appendix B for a formal description): With reference to his own dirty tree, he guides the client through a path on Eve's dirty tree that starts at the root and ends at the first leaf of disagreement. He does this by iteratively asking Eve to reveal an increasingly deeper node at a time. Given a node revealed at a certain height, Charlie queries the left or the right child as illustrated in Figure 1. The left child is queried if it does not match the corresponding internal node of his own tree, indicating a mismatch; otherwise he selects to query the right child, since the left subtrees are identical. When the process finishes, the light client has arrived at the first point of disagreement between Charlie's and Eve's augmented dirty ledgers.

Once the augmented dirty ledger entry of disagreement is identified, the client must verify that Eve's entry is fraudulent, as claimed by Charlie: It either contains an incorrect transaction or an invalid state commitment. If the transaction within Eve's entry is different from the one in the confirmed dirty ledger at the claimed position, the client can detect this by asking for the transaction's Merkle inclusion proof with respect to the header chain client already holds (*cf.* Appendix E.2 for a formalization of this inclusion check and Appendix F for an overview of how it can be implemented on different consensus protocols). On the other hand, if the transaction is correct, the client can locally evaluate the correct state commitment at that position by applying the transaction to the previous state commitment (which is valid as Charlie agrees with it). For this purpose, the client need not download the whole previous state tree, but instead asks for a fraud proof from Charlie. Fraud proofs were first introduced by Al-Bassam et al. [4] to provide security for light clients against dishonest *majorities* that can include invalid state commitments in Ethereum. They consist of the state elements touched by the transaction and their Merkle proofs within the previous state commitment. They allow the client to obtain the new, correct state commitment by updating these state elements and the relevant inner nodes of the state Merkle tree. Therefore, any discrepancy in the state commitment can be caught by the client (*cf.* Appendix E.3 for a formalization of this state check and how it can be implemented on UTXO based protocols).

1.2 Related work

Eager light clients were first introduced by Nakamoto [38]. *Superlight* clients for eager proof-of-work blockchains were put forth as NIPoPoWs [35,12,31,37] and Mina [8] (formerly known as CODA). Improved and superlight clients for eager proof-of-stake chains were described by Gaži et al. [27] and Agrawal et al. [2][3] respectively. Our techniques are orthogonal to theirs and can be composed as discussed in Appendix D. For an overview of different light client constructions, we refer the reader to Chatzigiannis et al. [16].

Our interactive bisection game is based on the work of Canetti et al. [15] which was first applied in the blockchain setting by Arbitrum [30]. Computation over large public logs was also explored by VerSum [29]. Contrary to Canetti and Arbitrum, where bisection games are used to dispute *computation* over static data, our bisection games are administered over *ledgers*, ever-growing and with different sizes. This challenge requires us to introduce novel techniques in these refereed games such as the use of Merkle Mountain Ranges [47,22] and a *Suffix Monologue* in our construction (Section 3.3). Finally, on the multi-server case, we improve the quadratic communication complexity of Canetti [15] to linear by our multiparty tournaments (Section 3.4).

[3] The paper [2] was made public after an initial version of this work.

As an alternative to our construction, recursive compositions [7] of SNARKs [6,8] or STARKs can be used to support non-interactive lazy light clients. For example, Mina [8] relies on recursive SNARKs with trusted setup to enable verification of *all* past state transitions in constant time. Plumo [48] proposes a SNARK-based blockchain client with trusted setup that can prove months of state history with a single transition proof. Halo [9] (later formalized by [11]) introduced the first practical recursive SNARK without trusted setup. Our work also does not require a trusted setup and our provers can update their state in an online fashion within milliseconds on commodity hardware, with minimal RAM requirements (for comparison, zkBridge [50] that uses SNARK proofs incurs a proving cost of $50 million per year). We also do not require changes in the consensus layer to support pairing-friendly and ZK-friendly elliptic curves. Our construction uses simple primitives that are straightforward to implement today, and give insight to the structure of the underlying problem. Lastly, although the ZK-based solutions do not require synchrony and the existential honesty assumption for the safety of the lazy light clients (albeit requiring them for liveness), these assumptions are already needed for the clients to identify the correct header chain (consensus security) upon bootstrapping on many blockchains such as Bitcoin, Ethereum and Cardano. Therefore, our work does not introduce extra assumptions for the security of the lazy light client construction.

2 Preliminaries & Model

Notation. For a natural number n, we use $[n]$ to denote the set $\{1, \cdots, n\}$. We use ϵ for the *empty string*. Given two strings a and b, we write $a \| b$ for some unambiguous encoding of their *concatenation*. Given a sequence X, $X[i]$ represents the i^{th} element (starting from 0). Negative indices address elements from the end, so $X[-1]$ is the last element. We use $X[i{:}j]$ to denote the subsequence of X consisting of the elements indexed from i (inclusive) to j (exclusive). The notation $X[i{:}]$ means the subsequence of X from i onwards, while $X[{:}j]$ means the subsequence of X up to (but excluding) j. We use $|X|$ to denote the size of a sequence. For a non-empty sequence X, we use $(x{:}xs) \leftarrow X$ to mean that the first element of X is assigned to x, while the rest of the elements are assigned to the (potentially empty) sequence xs. 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. We use $X \preceq Y$ ($X \prec Y$) to mean that X is a (strict) prefix of Y. If either $X \preceq Y$ or $Y \preceq X$, then X and Y are said to be consistent. We use $X \mid Y$ to denote that X is a *subarray* of Y, *i.e.*, all elements in X appear in Y consecutively. We use H to denote a generic, *collision-resistant* cryptographic hash function [32].

There are three types of nodes: *consensus nodes*, *full nodes*, and *light clients*.

Consensus nodes receive *constant* size transactions from the network and run a consensus protocol to obtain chains consisting of blocks. These chains are subsequently broadcast to all other nodes. Upon receiving a *confirmed* chain from the consensus nodes, each node reads its chain and produces a sequence of transactions (with total order) called the *ledger*.

The consensus nodes are *lazy*: They treat transactions as meaningless strings, without validating them. They include in their proposed blocks *any* received transaction with some spam-resilience mechanism (*e.g.*, they typically maintain a minimal notion of state that enables transactions to pay fees for block space).

The ledgers held by different nodes satisfy two properties: (1) *Safety* mandates that the ledgers of all honest nodes are consistent with each other; (2) *Liveness* mandates that, if an honest node broadcasts a new transaction, it will eventually appear in the ledger of all honest nodes within some finite delay.

Full nodes do not execute the consensus protocol, and instead, rely on the consensus nodes to provide them with a confirmed chain and the associated ledger. Contrary to consensus nodes, full nodes execute transactions to maintain a *state* (*e.g.*, a Merkle tree of account balances) uniquely determined by the ledger. An empty ledger corresponds to a constant *genesis state*, st_0. To determine the state of a non-empty ledger, transactions from the ledger are iteratively applied on top of the state, starting at the genesis state. This is captured by a transition function $\delta(\cdot, \cdot)$ taking a state and a transaction and producing a new state. Given a dirty ledger $\mathbb{L} = \mathsf{tx}_1 \cdots \mathsf{tx}_n$, the state becomes $\delta(\delta(\cdots \delta(st_0, \mathsf{tx}_1), \cdots), \mathsf{tx}_n)$. 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, *i.e.*, $\delta^*(st_0, \overline{\mathsf{tx}}) = \delta(\delta(\cdots \delta(st_0, \mathsf{tx}_1), \cdots), \mathsf{tx}_n)$.

Some transactions may not be applicable to a particular state, in which case they are said to be *invalid* with respect to the state (*e.g.*, double spends). As we are dealing with *lazy* systems, invalid transactions may still be contained in the ledger, hence the ledger is called *dirty*. We denote by $\mathbb{L}_r^v$ the dirty ledger in the view of a full node v at time r. If safety is guaranteed, then we use $\mathbb{L}_r^\cup$ to denote the longest among all the dirty ledgers kept by honest nodes at round r. Similarly, we use $\mathbb{L}_r^\cap$ to denote the shortest among them. We skip r in this notation if it is clear from the context. By convention, if a transaction tx cannot be applied to state st, we let $\delta(\mathsf{st}, \mathsf{tx}) = \mathsf{st}$. Each state is committed to by a succinct representation called the *state commitment* (*e.g.*, a Merkle root) and denoted by $\langle \mathsf{st} \rangle$. State commitments have constant size. We denote by $\langle \cdot \rangle$ the commitment function that takes a state and produces its commitment, *i.e.*, $\langle \mathsf{st} \rangle$ is the commitment to the state st.

Light clients wish to find out a particular state element (*e.g.*, its own account balance) without downloading the whole ledger or executing the transactions. As in the SPV model, the light client downloads and verifies the *header chain* from the consensus nodes (*e.g.*, the longest chain headers containing *transaction roots*), but not the transactions themselves. Given a chain $\mathcal{C}$ with $|\mathcal{C}|$ blocks and the corresponding ledger of size $L = |\mathbb{L}|$, a full node downloads data proportional to $\mathcal{O}(|\mathcal{C}| + L)$, where $|\mathcal{C}|$ comes from the header chain and L comes from the transactions. In contrast, a light client wants to learn its desired state element by downloading asymptotically less data. We call a light client *succinct* if instead of L, it only needs to download $O(\text{poly} \log L)$ bits after obtaining the header chain.

The Prover–Verifier model. We are interested in a *light client* $\mathcal{V}$ who is booting up the network for the first time. It connects to full nodes who are fully synchronized with the rest of the network. The client acts as a *verifier*, while the full nodes act as *provers* [35]. We assume at least one of the provers that $\mathcal{V}$ connects to is honest (the standard *non-eclipsing* assumption [25,26,28,49]), but the rest can be adversarially controlled. The honest provers follow the specified protocol and the adversary can run any probabilistic polynomial-time algorithm.

Network. Time proceeds in discrete rounds. The network is *synchronous*, *i.e.*, a message sent by one honest node at the end of round r is received by all honest nodes at the beginning of round $r+1$. The adversary can inject arbitrary, but bounded number of messages to the network. She can also reorder the messages sent by honest nodes and deliver them in a different order to different honest nodes. However, she cannot censor honest messages. As popular lazy blockchain systems such as Celestia (LazyLedger) [3], Prism [5], and Parallel Chains [24] were proven secure under the synchronous network model, our construction does not impose extra requirements for these systems.

3 Construction

We next describe the protocol for ledgers of variable and dynamic lengths.

3.1 Augmented Dirty Ledgers, Dirty Trees and MMRs

The prover *augments* each element of its dirty ledger $\mathbb{L}$ and produces an *augmented dirty ledger* $\mathbb{L}_+$, where every transaction in the original dirty ledger is replaced with a pair. The pair, denoted by $(\mathsf{tx}, \langle \mathsf{st} \rangle)$, contains the original transaction tx as well as a commitment $\langle \mathsf{st} \rangle$ to the state after this transaction is applied. The first element of $\mathbb{L}_+$ is the pair $(\epsilon, \langle st_0 \rangle)$, consisting of the empty string (as there is no genesis transaction) and the genesis state commitment $\langle st_0 \rangle$. The state commitment of $\mathbb{L}_+[i+1]$ is computed by applying the transaction at $\mathbb{L}[i+1]$ to the state committed to by $\mathbb{L}_+[i]$. Concretely, if $\mathbb{L} = (\mathsf{tx}_1, \mathsf{tx}_2, \ldots)$, then $\mathbb{L}_+ = ((\epsilon, \langle st_0 \rangle), (\mathsf{tx}_1, \langle \delta(st_0, \mathsf{tx}_1) \rangle), (\mathsf{tx}_2, \langle \delta(\delta(st_0, \mathsf{tx}_1), \mathsf{tx}_2) \rangle), ..)$.

The dirty tree $\mathcal{T}$ corresponding to an augmented dirty ledger $\mathbb{L}_+$ is defined as the Merkle tree that contains $\mathbb{L}_+[i]$ as the i-th leaf.

To organize ledgers of various sizes, provers use Merkle Mountain Ranges (MMRs). Provers construct their MMRs on their augmented dirty ledgers. To build an MMR, a prover divides his augmented dirty ledger $\mathbb{L}_+$ into segments $s_1, s_2, \ldots, s_k$ with lengths $\overline{\ell} = (\ell_1, \ell_2, \ldots, \ell_k)$, where $\ell_1 = 2^{q_1} > \ell_2 = 2^{q_2} > \ldots > \ell_k = 2^{q_k}$ are unique decreasing powers of 2. Each of these k segments is then organized into a dirty tree, and those trees $\mathcal{T} = (\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k)$ are collected into an array, that is the MMR $\mathcal{T}$. The roots $\langle \mathcal{T} \rangle = (\langle \mathcal{T} \rangle_1, \langle \mathcal{T} \rangle_2, \ldots, \langle \mathcal{T} \rangle_k)$ of these dirty trees, where $\langle \mathcal{T} \rangle_i = \langle \mathcal{T}_i \rangle$, are called the *peaks*. When there is a new transaction, the provers update their MMRs in amortized constant time, worst case update time per transaction being logarithmic in the size of the dirty ledger.

3.2 Views in Disagreement

Consider a light client $\mathcal{V}$ that connects to an honest prover $\mathcal{P}$ and an adversarial prover $\mathcal{P}^*$, but does not know who is who. Let st be the current state in $\mathcal{P}$'s view at round r. Let $\mathbb{L}$ denote the dirty ledger, $\mathbb{L}_+$ denote the augmented dirty ledger, and $\mathcal{T}$ denote the MMR of $\mathcal{P}$ at round r. The last entry $\mathbb{L}_+[-1]$ of the honest augmented dirty ledger contains the commitment $\langle \mathsf{st} \rangle$ to the latest state st. The client $\mathcal{V}$ wishes to learn the value of a particular state element in st. For this purpose, $\mathcal{V}$ only needs to learn a truthful state commitment $\langle \mathsf{st} \rangle$; as from there, an inclusion proof into $\langle \mathsf{st} \rangle$ suffices to show inclusion of any state element value. So the goal of $\mathcal{P}$ is to convince $\mathcal{V}$ of the correct state commitment $\langle \mathsf{st} \rangle$. If both provers respond to $\mathcal{V}$'s request with the same commitment $\langle \mathsf{st} \rangle$, then $\mathcal{V}$ knows that the received state commitment is correct (because at least one prover is honest). If they differ, it must discover the truth. If at any point in time, one of the provers *timeouts*, *i.e.*, fails to respond in one round of receiving its query, the prover is considered adversarial and ignored thereafter (as the network is synchronous, no honest prover would timeout). In practice, this equates to a short timeout in the network connection.

Suppose the two provers claim different state commitments, $\langle \mathsf{st} \rangle$ and $\langle \mathsf{st} \rangle^*$ respectively, where $\langle \mathsf{st} \rangle \neq \langle \mathsf{st} \rangle^*$. To prove the correctness of its commitment, $\mathcal{P}$ sends to $\mathcal{V}$ the peaks $\langle \mathcal{T} \rangle = (\langle \mathcal{T} \rangle_1, \ldots, \langle \mathcal{T} \rangle_k)$ of its MMR $\mathcal{T}$, the length $\ell = |\mathbb{L}_+|$ of its augmented dirty ledger, and the Merkle proof π from the last leaf $\mathbb{L}_+[-1]$, which contains $\langle \mathsf{st} \rangle$, to the root $\langle \mathcal{T} \rangle_k$. The adversary $\mathcal{P}^*$ sends to $\mathcal{V}$ the alleged peaks $\langle \mathcal{T} \rangle^* = (\langle \mathcal{T} \rangle^*_1, \ldots, \langle \mathcal{T} \rangle^*_{k^*})$, an alleged length ℓ^*, and an alleged Merkle proof π^*. Since $\langle \mathsf{st} \rangle \neq \langle \mathsf{st} \rangle^*$, if π and π^* both verify, then we have that $\langle \mathcal{T} \rangle_k \neq \langle \mathcal{T} \rangle^*_{k^*}$. In this case, $\mathcal{V}$ mediates a *challenge game* between $\mathcal{P}$ and $\mathcal{P}^*$ to determine which of the peaks $\langle \mathcal{T} \rangle$ or $\langle \mathcal{T} \rangle^*$ were constructed honestly:

Definition 1 (Well-formed Ledgers, Trees and MMRs). *An augmented dirty ledger* $\mathbb{L}_+$ *is said to be* well-formed *at round* r *with respect to transition* δ*, genesis state* st_0*, and commitment function* $\langle\cdot\rangle$ *if:* $\mathbb{L}_+[0] = (\epsilon, \langle st_0 \rangle)$ *and,* $\forall\ i \in [|\mathbb{L}_+| - 1]$*,* $\mathbb{L}_+[i] = (\mathsf{tx}_i, \langle \mathsf{st}_i \rangle)$ *such that* $(\mathsf{tx}_{i-1}, \mathsf{tx}_i) \mid \mathbb{L}^{\cup}_r$*, and* $\mathsf{st}_i = \delta^*(st_0, \mathbb{L}[{:}i])$.

A dirty tree or MMR $\mathcal{T}$ *is said to be* well-formed *if its leaves correspond to the entries of a well-formed augmented dirty ledger.*

The augmented dirty ledger and MMR held by an honest prover are always well-formed. Hence, to determine whether $\langle \mathcal{T} \rangle_k$ or $\langle \mathcal{T} \rangle^*_{k^*}$ contain the correct state commitment, it suffices for the verifier to check if $(\langle \mathcal{T} \rangle_1, \ldots, \langle \mathcal{T} \rangle_k)$ or $(\langle \mathcal{T} \rangle^*_1, \ldots, \langle \mathcal{T} \rangle^*_{k^*})$ correspond to the peaks of a *well-formed* MMR.

3.3 Challenge Game

We now explore the challenge game that allows the verifier to compare competing claims by two provers. During the game, the prover with the larger claimed ledger length ℓ acts as the challenger while the other acts as the responder. The goal of the challenger is to identify the first point on the responder's alleged augmented dirty ledger that disagrees with his own ledger. The challenge game consists of

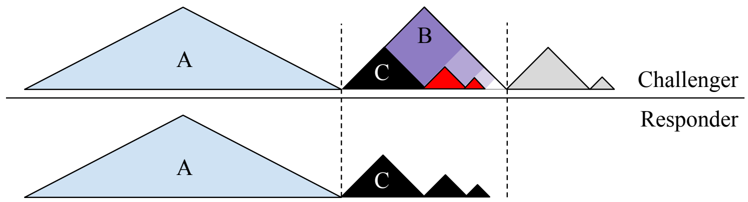

Fig. 2: The challenger's MMR (top) is compared to the responder's alleged MMR. The first two peaks (A in blue) are the same, so they are skipped by Alg. 1. The second peak of the challenger is reached (B in purple) and compared against the responder's second peak (C). When found to be different, the challenger knows that the remaining responder peaks (in black, bottom) will lie within his own current tree (B, in purple); so Alg. 1 Line 4 calls Alg. 2 to compare the black peaks against the purple tree.

two phases: During the first *zooming phase*, the challenger reduces his search of the first point of disagreement to a single tree within the responder's MMR. After this first phase is completed, the second phase consists of either the two parties playing a *bisection game* (*cf.* Section 1.1, and for a more detailed description Appendix B) or the challenger going into a *suffix monologue*.

Zooming phase. To narrow his search down to a single tree, the challenger first calls Alg. 1 to identify the earliest peak among the responder's peaks that disagrees with his own peaks. Alg. 1 iterates over the responder's peaks (Alg. 1 Line 2) until the challenger finds a peak $\langle \mathcal{T} \rangle_i^*$ among those returned by the responder, that is different from the corresponding root $\langle \mathcal{T}_i \rangle$ in his own peaks. If the number of leaves under both peaks are the same, the challenger plays a bisection game on the Merkle trees whose roots are $\langle \mathcal{T}_i \rangle$ and $\langle \mathcal{T} \rangle_i^*$ (Alg. 1 Line 7). Otherwise, if the number of leaves under $\langle \mathcal{T} \rangle_i^*$ is smaller than the number of leaves under $\langle \mathcal{T}_i \rangle$, then, all the alleged data within the responder's remaining peaks lies under the i^{th} peak of the challenger (see Figure 2). The challenger has now reduced his search to his own i^{th} tree and can compare it against the responder's remaining peaks. This is done by calling Alg. 2 on the remaining peaks of the responder (Alg. 1 Line 4).

Alg. 2 narrows the search for the first point of disagreement to one of responder's peaks, so that $\mathcal{V}$ can compare the two trees using a bisection game. Consider the responder's remaining peaks overlayed onto the challenger's tree $\mathcal{T}_i$ (dashed lines in Figure 2). They correspond to certain inner nodes within $\mathcal{T}_i$ (black, red, and red subtrees at the top of Figure 2). Alg. 2 locates the first such inner node that disagrees with the responder's corresponding peak (the left-most red subtree in Figure 2). Finally, at this point, the challenger plays the bisection game on the sub-trees under this inner node and the responder's currently inspected root (Alg. 2 Line 13). After the bisection game, either the challenger or the responder is declared the winner and the other one is declared the loser.

Suffix monologue. When the MMRs are well-formed, there is no first point of disagreement between the two alleged augmented dirty ledgers, and the ledgers form a prefix of one another. In that case, the provers will not enter into a

Algorithm 1 The algorithm run by the challenger to identify the first peak in the responder's MMR that is different from that of the challenger. The variables $\mathcal{T}$ and $\overline{\ell}$ denote the challenger's sequence of Merkle trees and a sequence of their respective sizes, whereas $\langle\mathcal{T}\rangle^*$ and $\overline{\ell}^*$ denote the responder's sequence of peaks and the corresponding number of leaves respectively. The algorithm BisectionGame initiates a bisection game between the challenger's tree and the responder's alleged tree with the same size.

function PeaksVsPeaks($\mathcal{T}, \overline{\ell}, \langle\mathcal{T}\rangle^*, \overline{\ell}^*$)
  for $i = 0$ to $|\langle\mathcal{T}\rangle^*| - 1$ **do**
    if $\overline{\ell}[i] \neq \overline{\ell}^*[i]$ **then**
      return TreeVsPeaks($\mathcal{T}[i], \langle\mathcal{T}\rangle^*[i{:}], \overline{\ell}^*[i{:}]$)
    end if
    if $\mathcal{T}[i]$.root $\neq \langle\mathcal{T}\rangle^*[i]$ **then**
      return BisectionGame($\mathcal{T}[i], \overline{\ell}[i]$)
    end if
  end for
end function

bisection game, and it is the challenger's turn to present his augmented dirty ledger entries extending the responder's ledger with size ℓ^*. Specifically, the challenger presents the *suffix* $\mathbb{L}_+[\ell^*{:}\min(\ell, \ell^* + \psi)]$ and the verifier checks the transitions within this suffix. Concretely, for every consecutive $(\mathsf{tx}_j, \langle\mathsf{st}_j\rangle)$ and $(\mathsf{tx}_{j+1}, \langle\mathsf{st}_{j+1}\rangle)$, for $\ell^* \leq j < \min(\ell, \ell^* + \psi)$, the verifier checks the inclusion of tx_j and tx_{j+1} in the header chain as before, and verifies that the state $\langle\mathsf{st}_{j+1}\rangle$ has been computed correctly using δ. The verifier also checks the transition from $\mathbb{L}^*_+[\ell^* - 1] = \mathbb{L}^*_+[-1]$, *i.e.*, the responder's last entry, to $\mathbb{L}_+[\ell^*]$, *i.e.*, the first entry in the challenger's suffix, since the challenger, by starting the suffix monologue, claims that his augmented dirty ledger is a suffix of the responder's.

The parameter ψ is a constant selected in accordance with the chain growth and liveness parameters of the blockchain (*cf.* Appendix E). The bound ψ on the number of transitions to check prevents the suffix monologue from violating succinctness. By the Common Prefix property [25], discrepancy in the lengths of two honest provers' ledgers is bounded when there is an upper bound on the chain growth rate, which is the case for our protocols of interest (*cf.* the ledger Lipschitz property in Section E.1). Hence, if the challenger presents ψ or more extra entries with consecutive transactions and correct state transitions, then the responder is declared a loser, as he presented too short a ledger to possibly be honest. In other words, if an adversarial responder presents a much shorter ledger, then the honest challenger sends ψ entries, proving to the verifier that the adversary's ledger is too short. On the contrary, an adversarial challenger cannot present a well-formed ledger much longer than an honest responder's ledger, without breaking the underlying consensus protocol. If the challenger fails to present a well-formed suffix, then the responder is declared the winner, while the challenger is declared the loser. Otherwise, if the suffix presented is well-formed and has length less than ψ, then *both* provers are declared winners of

Algorithm 2 The algorithm run by the challenger to identify the first subtree, under one of the challenger's larger Merkle trees, that is different from the responder's peak. The variable $\mathcal{T}$ denotes the challenger's larger Merkle tree whereas $\langle\mathcal{T}\rangle^*$ and $\overline{\ell}^*$ denote the responder's sequence of peaks (with some of the first elements chopped off during the recursion) and the corresponding number of leaves respectively.

function TreeVsPeaks($\mathcal{T}$, $\langle\mathcal{T}\rangle^*$, $\overline{\ell}^*$)
    assert $\mathcal{T}$.size $> \sum_{\ell^* \in \overline{\ell}^*} \ell^*$
    if $|\langle\mathcal{T}\rangle^*| = 0$ **then**
        return ▷ Done: the MMRs are well-formed.
    end if
    (peak:peaks) $\leftarrow \langle\mathcal{T}\rangle^*$
    (reSize:reSizes) $\leftarrow \overline{\ell}^*$
    if $\lfloor \frac{\mathcal{T}.\text{size}}{2} \rfloor >$ reSize **then**
        TreeVsPeaks($\mathcal{T}$.left, $\langle\mathcal{T}\rangle^*$, $\overline{\ell}^*$)
    else if tree.left.root = peak **then**
        TreeVsPeaks($\mathcal{T}$.right, peaks, reSizes)
    else
        BisectionGame($\mathcal{T}$, reSize)
    end if
end function

the challenge game. Unlike the bisection game, at the end of the suffix monologue, *both* the challenger and the responder can win.

3.4 Multiparty Tournaments

Upon joining the network, the verifier $\mathcal{V}$ contacts a subset of all available provers[4] for queries. If all of the responses are the same, $\mathcal{V}$ accepts the response as the correct answer. If it receives different responses, $\mathcal{V}$ arbitrates a *tournament* among the provers that responded. The tournament's purpose is to select a prover whose latest claimed state is as up-to-date as the state obtained by applying the transition function iteratively on the transactions in $\mathbb{L}^\cap$.

Suppose $\mathcal{V}$ hears back from n provers. Before the tournament, $\mathcal{V}$ orders the n provers into a sequence $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ in an increasing order of their (claimed) augmented dirty ledger sizes. This sequence dictates the order in which the provers play the bisection games. Then, $\mathcal{V}$ starts the tournament that consists of n steps (*cf.* Appendix B for the algorithm run by $\mathcal{V}$). Before the first step, it initializes the set $\mathcal{S} = \emptyset$. At the end of each step t, $\mathcal{S}$ contains the provers that have engaged in at least one challenge game, and have not lost any by step t.

The tournament starts with a challenge game between $\mathcal{P}_1$ and $\mathcal{P}_2$, during which $\mathcal{P}_1$ with the larger alleged augmented dirty ledger challenges $\mathcal{P}_2$. The winners are added to $\mathcal{S}$ and the tournament moves to the second step. At each step of the tournament, the set $\mathcal{S}$ is updated to contain the winners so far.

[4] For instance, according to `https://github.com/bitcoin/bitcoin/blob/master/doc/reduce-traffic.md`, Bitcoin makes 8 outbound full-relay connections.

Algorithm 3 The algorithm ran by the responder to reply to the challenger's queries while the challenger tries to identify the first point of disagreement against the responder's MMR. The variable $\mathbb{L}_+$ denotes the responder's augmented dirty ledger. The algorithm MakeMMR returns the MMR based on the given augmented dirty ledger.

```
function Respond(𝕃+)
    trees ← MakeMMR(𝕃+)
    peaks ← {tree.root : tree ∈ trees}
    peaks --→ Challenger
    pNum ←-- Challenger
    tree ← trees[pNum]                          ▷ Enter a particular Merkle Tree
    loc ← ⊥
    while tree.size > 1 do
        (tree.left, tree.right) --→ Challenger
        dir ←-- Challenger
        if dir = 0 then
            tree ← tree.left
        else
            tree ← tree.right
        end if
        loc ← loc ‖ dir
    end while
    dirtyLedger[loc] --→ Challenger
end function
```

Players may be removed from the set $\mathcal{S}$ if they lose, and new winners can be added to $\mathcal{S}$ as they win. Each player $\mathcal{P}$ is considered in order. Let $\overline{\mathcal{P}}$ denote the prover in $\mathcal{S}$ that claims to have the largest augmented dirty ledger at a given step i. The prover among $\{\mathcal{P}, \overline{\mathcal{P}}\}$ with the larger alleged augmented dirty ledger challenges the other prover. Depending on the outcome of the challenge game, there are three cases:

1. If both provers win, $\mathcal{P}$ is added to $\mathcal{S}$ and the tournament moves to step $i+1$.
2. If $\mathcal{P}$ loses, the tournament directly moves to step $i+1$ and $\mathcal{S}$ stays the same.
3. If $\mathcal{P}$ wins and $\overline{\mathcal{P}}$ loses, $\overline{\mathcal{P}}$ is removed from $\mathcal{S}$. Then, $\mathcal{P}$ challenges the new $\overline{\mathcal{P}}$, the prover with the largest alleged size among those remaining in the set $\mathcal{S}$ (or vice versa). This case is repeated until either one of cases (1) or (2) happens, or there are no provers left in $\mathcal{S}$. If the latter happens, $\mathcal{P}$ is added to $\mathcal{S}$ and the tournament moves to step $i+1$.

The procedure above is repeated until the end of step $n-1$, after which, $\overline{\mathcal{P}}$ wins the tournament. Then, the verifier accepts the state commitment of $\overline{\mathcal{P}}$ among those remaining in $\mathcal{S}$ as the correct state.

The tournament consists of $\mathcal{O}(n)$ bisection games and has an $\mathcal{O}(n)$ running time[5]. The reason is that, after each game, one party is eliminated from the winners, either by not being added to $\mathcal{S}$, or by being removed from $\mathcal{S}$, and every

[5] In contrast, the playoff in [15, Appendix G] consists of $\mathcal{O}(n^2)$ games and has an $\mathcal{O}(n)$ running time due to games happening in parallel.

party can be eliminated at most once. Parallelizing the tournaments can also help make the runtime sublinear in the number of provers.

4 System Considerations

4.1 Running Time of Bisection Games

Implementation and experimental setup. We report on a prototype implementation of the prover and the verifier in 1000 lines of Go[6], and set up 17 provers on AWS `r5.xlarge` instances distributed across 17 data centers around the globe. All provers have ledgers of the same size, but only one of the provers has the correct augmented dirty ledger. Ledgers of the remaining 16 provers differ from the correct one at a randomly selected position. To simplify the prototype, we do not implement a state transition function δ, *e.g.*, the Ethereum Virtual Machine. (We explore the cost of proving state transitions in the next subsection.) Instead, all transactions and state commitments (*cf.*. Algorithm 4) are random byte strings. We hard code the prover and the verifier such that a state commitment is valid to the verifier only if the prover has the correct commitment in its augmented dirty ledger. We run the verifier under a residential internet connection with 300 Mbps downlink and 10 Mbps uplink bandwidth. The verifier connects to all of the 17 provers, and arbitrates the tournament (Algorithm 6) among them.

Verification latency. We first explore the duration of the tournament. So far, we have only discussed *binary* Merkle trees for use in our bisection game, but we can consider m-ary Merkle trees more generally. Increasing the degree m of a dirty tree flattens the tree, resulting in fewer rounds of interactivity in the bisection game. On the other hand, opening an inner node now requires sending m children, resulting in higher bandwidth usage. Appendix C models the latency-bandwidth trade-off realized by tuning m. Here, we experimentally explore the trade-off. For this experiment, we fix the ledger size to 10 million transactions and vary m. For each configuration, we run 10 tournaments and measure the average and the standard deviation of the duration (Figure 3a). When $m = 300$, the duration reaches the lowest, at 18.37s. Most blockchains confirm transactions with a latency of tens of seconds. In comparison, the tournament adds little extra time on top of the end-to-end latency that a light client perceives for new transactions. Under this configuration, a tournament consists of 16 games, and each game involves 4 rounds of interaction between the verifier and the provers. On average, each round of interaction lasts for 0.287s. In comparison, the average network round-trip time (RTT) from the verifier to the provers is 0.132s.

As we vary m, tournaments take longer to finish. Specifically, a smaller m makes each Merkle tree opening smaller, but increases the number of openings per game, making network propagation latency the main bottleneck of the game. In contrast, a larger m makes bandwidth the main bottleneck. As we increase

[6] The code is open source under MIT license, and is available at `https://github.com/yangl1996/super-light-client`.

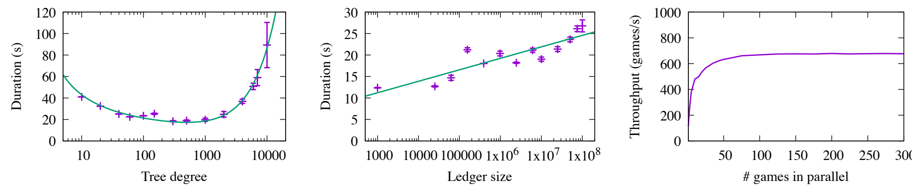

Fig. 3: In (a) and (b), we measure the time to complete a tournament of 17 geo-distributed provers. Error bars show the standard deviation. Solid lines show the trend. (a) Time when varying the tree degree m. (b) Time when varying the ledger size L. (c) Throughput of games with two provers and one verifier co-located in a data center. The verifier initiates games with variable parallelism to saturate the provers.

m to 10000, each opening of the Merkle tree becomes large enough such that message transmission is affected by the fluctuation of the internet bandwidth, causing a higher variation in the tournament duration.

Scalability. We now evaluate how our scheme scales as the ledger size grows. We fix the tree degree m to 300, vary the size of the ledger from 1000 to 100 million transactions and report the mean and the standard deviation over 10 tournaments for each datapoint (Figure 3b). As we increase the ledger size by $10^5\times$, the tournament duration grows from 13s to 26s, an increase of only $2\times$. This is because the number of interactions in a game is equal to the depth of the Merkle tree, which grows logarithmically with the ledger size.

Prover throughput. Finally, we evaluate the throughput of a prover participating in many games (Figure 3c). To minimize the network influence, we run two provers in the same datacenter. Each prover has a ledger of 10M transactions, which differ at a random location. We start a verifier in the same datacenter, which initiates a variable number of bisection games between the two provers in parallel. We gradually ramp up the parallelism to generate enough load and saturate the provers. During the process, the achieved throughput first increases due to the increased load, and then stays flat because the provers have saturated their computational resources. Specifically, a prover running our prototype can support a throughput of 680 games/second using its 4 virtual CPU cores. We expect the throughput to scale with the available CPU cores and disk IO.

4.2 Proving State Transitions

We next discuss the cost of proving and verifying state transitions, which happens when a bisection game ends with a point of disagreement. For concreteness, we use the Ethereum Virtual Machine (EVM) as an example, but the discussion applies to other state machines.

Ledger granularity. So far, we have assumed that the bisection game runs with a granularity of transactions. While the proof size is small in this natural configuration (less than 20 state elements on average for recent Ethereum transactions), an honest prover needs to maintain snapshots of ledger states as of every historical transaction to generate such proofs for arbitrary points in the dirty ledger. Maintaining these snapshots can be costly, since even blockchain nodes in "archival" mode—ones that store the most historical data—do not keep such fine-grained information. We propose that real-world deployments use a granularity of *blocks*, *i.e.*, treating an entire block as a single entry in the dirty ledger. To generate state transition proofs, provers only need access to state snapshots as of each block, which are readily available from archival nodes. A direct benefit is that provers can be implemented using public RPC APIs provided by EVMs, namely the `debug_traceBlock` RPC which lists all state elements read/written by a block. This allows provers to make use of existing archival nodes and eliminates the need to maintain separate state snapshots.

An apparent downside of this coarse-grained approach is that state transition proofs are larger, consisting of state elements touched by an entire block plus the relevant Merkle proofs. However, our experiments show that such proofs are less than 10 MB for recent Ethereum blocks, which can be downloaded within 0.3 seconds with a 300 Mbps internet connection used in previous experiments, adding little to the seconds-long duration of the bisection game.

Verification costs. Upon downloading a state transition proof (consisting of the state elements touched by the transactions within the block at the first point of disagreement and their Merkle proofs), the verifier needs to check the proof by executing the transactions locally. We implemented a verifier by forking `foundry`[7], an EVM implementation in Rust, and used it to benchmark verification costs on commodity mobile hardware.

Experimental results show that verifying state transition of recent Ethereum blocks takes less than 0.8 seconds per block on average on a M1 MacBook Pro and consumes 2.5 Joules of energy[8]. The same verification takes less than 1.5 seconds on an underpowered tablet with a 2-core Intel m5 low-power CPU manufactured in 2015. In comparison, a *full node* syncing with the latest EVM state from genesis has to execute all historical transactions, which takes at least *a full day* on a workstation with 32 GB of RAM and a 4-core Intel Xeon CPU, and uses 540 GB of SSD. Our construction saves significant time, computation, and storage because the light client only needs to locally execute the one block at the first point of disagreement.

5 Analysis

We state our security theorems informally in this section. For the rigorous theorem statements and proofs, see Appendix G. We begin by defining *State Security*, which captures the verifier's goal of obtaining a state consistent with the rest of the network: There is no disagreement with the other honest nodes (safety), and the state downloaded is recent (liveness).

Definition 2 (State Security). *An interactive Prover–Verifier protocol* (P, V) *is* state secure *with safety parameter* ν*, if there exists a ledger* $\mathbb{L}$ *such that the*

[7] https://github.com/foundry-rs/foundry

[8] Measured using `powermetrics` built into macOS.

state commitment $\langle \mathsf{st} \rangle$ *obtained by the verifier at the end of the protocol execution at round* r *satisfies* $\langle \delta^*(\mathsf{st}_0, \mathbb{L}) \rangle = \langle \mathsf{st} \rangle$*, and for all rounds* $r' \geq r + \nu$*:* $\mathbb{L}$ *is a* prefix *of* $\mathbb{L}^{\cup}_{r'}$ *(safety) and* $\mathbb{L}^{\cap}_{r}$ *is a* prefix *of* $\mathbb{L}$ *(liveness).*

The theorems for succinctness and security of the protocol are given below. Security consists of two components: completeness and soundness.

Lemma 1 (Succinctness (Informal)). *The challenge game invoked at round* r *with sizes* ℓ_1 *and* $\ell_2 > \ell_1$ *ends in* $\mathcal{O}(\log(\ell_1))$ *rounds of communication and has, considered in isolation, a total communication complexity of* $\mathcal{O}(\log r)$*.*

Theorem 1 (Completeness (Informal)). *The honest responder wins the challenge game against any PPT adversarial challenger.*

Theorem 2 (Soundness (Informal)). *Let* H *be a collision resistant hash function. For all PPT adversarial responders* $\mathcal{A}$*, an honest challenger wins the challenge game against* $\mathcal{A}$ *with overwhelming probability in* λ*.*

Theorem 3 (Tournament Runtime (Informal)). *Consider a tournament started at round* r *with* n *provers. Given at least one honest prover, for any PPT adversary* $\mathcal{A}$*, the tournament ends in* $\mathcal{O}(n \log r)$ *rounds of communication and has, considered in isolation, a total communication complexity of* $\mathcal{O}(n \log r)$*, with overwhelming probability in* λ*.*

The theorem below is a direct consequence of the above theorems.

Theorem 4 (Security (Informal)). *Consider a tournament started at round* r *with* n *provers. Given at least one honest prover, for any PPT adversary* $\mathcal{A}$*, the state commitment obtained by the prover at the end of the tournament satisfies* State Security *with overwhelming probability in* λ*.*

Acknowledgements

We thank Shresth Agrawal, Kostis Karantias, Angel Leon, Joachim Neu, and Apostolos Tzinas for several insightful discussions on this project. Ertem Nusret Tas is supported by the Stanford Center for Blockchain Research. Lei Yang is supported by a gift from the Ethereum Foundation.

References

1. Colored coins (2015), `https://en.bitcoin.it/wiki/Colored_Coins`
2. Agrawal, S., Neu, J., Tas, E.N., Zindros, D.: Proofs of proof-of-stake with sublinear complexity. In: AFT. LIPIcs, vol. 282, pp. 14:1–14:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023)
3. Al-Bassam, M.: Lazyledger: A distributed data availability ledger with client-side smart contracts. CoRR **abs/1905.09274** (2019)

4. Al-Bassam, M., Sonnino, A., Buterin, V., Khoffi, I.: Fraud and data availability proofs: Detecting invalid blocks in light clients. In: Financial Cryptography (2). Lecture Notes in Computer Science, vol. 12675, pp. 279–298. Springer (2021)
5. Bagaria, V.K., Kannan, S., Tse, D., Fanti, G., Viswanath, P.: Prism: Deconstructing the blockchain to approach physical limits. In: CCS. pp. 585–602. ACM (2019)
6. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 8617, pp. 276–294. Springer (2014)
7. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: STOC. pp. 111–120. ACM (2013)
8. Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Coda: Decentralized cryptocurrency at scale. IACR Cryptol. ePrint Arch. p. 352 (2020)
9. Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive proof composition without a trusted setup. IACR Cryptol. ePrint Arch. p. 1021 (2019)
10. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. CoRR **abs/1807.04938** (2018)
11. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Recursive proof composition from accumulation schemes. In: TCC (2). Lecture Notes in Computer Science, vol. 12551, pp. 1–18. Springer (2020)
12. Bünz, B., Kiffer, L., Luu, L., Zamani, M.: Flyclient: Super-light clients for cryptocurrencies. In: SP. pp. 928–946. IEEE (2020)
13. Buterin, V.e.a.: Light client protocol (2014), `https://eth.wiki/en/concepts/light-client-protocol`
14. Canetti, R., Riva, B., Rothblum, G.N.: Practical delegation of computation using multiple servers. In: CCS. pp. 445–454. ACM (2011)
15. Canetti, R., Riva, B., Rothblum, G.N.: Refereed delegation of computation. Inf. Comput. **226**, 16–36 (2013)
16. Chatzigiannis, P., Baldimtsi, F., Chalkias, K.: Sok: Blockchain light clients. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 13411, pp. 615–641. Springer (2022)
17. Dahlberg, R., Pulls, T., Peeters, R.: Efficient sparse merkle trees - caching strategies and secure (non-)membership proofs. In: NordSec. Lecture Notes in Computer Science, vol. 10014, pp. 199–215 (2016)
18. Daian, P., Pass, R., Shi, E.: Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 11598, pp. 23–41. Springer (2019)
19. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 10821, pp. 66–98. Springer (2018)
20. Developers, B.: Developer Guide - Bitcoin, `https://bitcoin.org/en/developer-guide`
21. Developers, F.: Fuel - Beyond Monolithic, `https://www.fuel.network/`
22. Developers, G.: Merkle Mountain Ranges (MMR), `https://docs.grin.mw/wiki/chain-state/merkle-mountain-range/`
23. Developers, O.: Optimism, `https://www.optimism.io/`
24. Fitzi, M., Gazi, P., Kiayias, A., Russell, A.: Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. IACR Cryptol. ePrint Arch. p. 1119 (2018)

25. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 9057, pp. 281–310. Springer (2015)
26. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol with chains of variable difficulty. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 10401, pp. 291–323. Springer (2017)
27. Gazi, P., Kiayias, A., Zindros, D.: Proof-of-stake sidechains. In: IEEE Symposium on Security and Privacy. pp. 139–156. IEEE (2019)
28. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: USENIX Security Symposium. pp. 129–144. USENIX Association (2015)
29. van den Hooff, J., Kaashoek, M.F., Zeldovich, N.: Versum: Verifiable computations over large public logs. In: CCS. pp. 1304–1316. ACM (2014)
30. Kalodner, H.A., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: USENIX Security Symposium. pp. 1353–1370. USENIX Association (2018)
31. Karantias, K., Kiayias, A., Zindros, D.: Compact storage of superblocks for nipopow applications. In: MARBLE. pp. 77–91. Springer Proceedings in Business and Economics, Springer (2019)
32. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. CRC Press (2014)
33. Kiayias, A., Lamprou, N., Stouka, A.: Proofs of proofs of work with sublinear complexity. In: Financial Cryptography Workshops. Lecture Notes in Computer Science, vol. 9604, pp. 61–78. Springer (2016)
34. Kiayias, A., Leonardos, N., Zindros, D.: Mining in logarithmic space. In: CCS. pp. 3487–3501. ACM (2021)
35. Kiayias, A., Miller, A., Zindros, D.: Non-interactive proofs of proof-of-work. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 12059, pp. 505–522. Springer (2020)
36. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 10401, pp. 357–388. Springer (2017)
37. Kiayias, A., Zindros, D.: Proof-of-work sidechains. In: Financial Cryptography Workshops. Lecture Notes in Computer Science, vol. 11599, pp. 21–34. Springer (2019)
38. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008), `https://bitcoin.org/bitcoin.pdf`
39. Neu, J., Tas, E.N., Tse, D.: Snap-and-chat protocols: System aspects. CoRR **abs/2010.10447** (2020)
40. Neu, J., Tas, E.N., Tse, D.: Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 446–465. IEEE (2021)
41. Russell, B.: In Praise of Idleness. Unwin (1935)
42. Sompolinsky, Y., Lewenberg, Y., Zohar, A.: SPECTRE: A fast and scalable cryptocurrency protocol. IACR Cryptol. ePrint Arch. p. 1159 (2016)
43. Sompolinsky, Y., Wyborski, S., Zohar, A.: PHANTOM GHOSTDAG: a scalable generalization of nakamoto consensus: September 2, 2021. In: AFT. pp. 57–70. ACM (2021)
44. Sompolinsky, Y., Zohar, A.: PHANTOM: A scalable blockdag protocol. IACR Cryptol. ePrint Arch. p. 104 (2018)

45. Tas, E.N.: Woods Attack on Celestia (2021), https://forum.celestia.org/t/woods-attack-on-celestia/59
46. Tas, E.N., Tse, D., Gai, F., Kannan, S., Maddah-Ali, M.A., Yu, F.: Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities. In: SP. pp. 126–145. IEEE (2023)
47. Todd, P.: Merkle mountain ranges (2012), https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md
48. Vesely, P., Gurkan, K., Straka, M., Gabizon, A., Jovanovic, P., Konstantopoulos, G., Oines, A., Olszewski, M., Tromer, E.: Plumo: An ultralight blockchain client. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 13411, pp. 597–614. Springer (2022)
49. Wüst, K., Gervais, A.: Ethereum eclipse attacks (2016), https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/121310/eth-49728-01.pdf
50. Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., Song, D.: zkbridge: Trustless cross-chain bridges made practical. In: CCS. pp. 3003–3017. ACM (2022)

A Attack on SPV Clients on Lazy Blockchains

We examine a hypothetical attack on the succinctness of the communication complexity of the Bitcoin's SPV [38] on a lazy blockchain protocol with UTXO-based execution model. Suppose at round r, the confirmed sequence of blocks contain the transactions tx_i, $i \in [n]$ in the reverse order: for $i \in \{2, \ldots, n-1\}$, tx_i appears in the prefix of tx_{i-1}. Each transaction tx_i, $i \in [n]$, spends two UTXO's, UTXO^p_i and UTXO^l_i, such that for $i = 2, .., n$, $\mathsf{UTXO}^l_i = \mathsf{UTXO}^p_{i-1}$, and UTXO^l_1 UTXO^p_i $i \in \{2, \ldots, n-1\}$, and UTXO^p_n are all distinct UTXOs. Thus, if tx_{i-1} appears in the prefix of tx_i, it *invalidates* tx_i as tx_i will be double-spending $\mathsf{UTXO}^l_i = \mathsf{UTXO}^p_{i-1}$ already spent by tx_{i-1}. However, tx_i, $i \in \{3, \ldots, n\}$, does not invalidate tx_j for any $j < i - 1$. We assume that no transaction outside the set tx_i, $i \in [n]$, invalidates tx_i for any $i \in [n-1]$.

If n is even, tx_1 would be invalid, since each transaction tx_i with an odd index $i \in \{1, 3, \ldots, 2n-1\}$, will be invalidated by the transaction tx_{i-1}. However, if n is odd, each transaction tx_i, with an even index $i \in \{2, 4, \ldots, 2n-1\}$, will be invalidated by the transaction tx_{i-1}, which has an odd index. Hence, tx_2 would be invalid, implying that tx_1 would be valid as no transaction other than tx_2 can invalidate tx_1 by our assumption.

Consider a light client whose goal is to learn whether tx_1 is valid with respect to the transactions in its prefix. Suppose n is even, *i.e.*, tx_1 is invalid. Towards its goal, the client asks full nodes it is connected to, whether tx_1 is valid with respect to the transactions in its prefix. Then, to convince the light client that tx_1 is invalid, an honest full node shows tx_2, which invalidates tx_1, along with its inclusion proof. However, in this case, an adversarial full node can show tx_3 to the client, which in turn invalidates tx_2, and gives the impression that tx_1 is valid. Through an inductive reasoning, we observe that the adversarial nodes can force the honest full node to show all of the transactions tx_i with even indices $i \in \{2, \ldots, n\}$, to the light client, such that no adversarial full node can verifiably claim tx_1's validity anymore. However, since n can be arbitrarily large, *e.g.*, a constant fraction of the confirmed ledger length, light client in this case would have to download and process linear number of transactions in the ledger length.

A related attack on rollups that use a lazy blockchain as its parent chain is described by [45].

B Bisection Game

The game is terminated by the verifier as soon as the victory of one of the provers becomes certain.

Challenger wins. The challenger wins the bisection game in the verifier's view whenever one of the following conditions fails:

1. The responder must not timeout, *i.e.*, must reply to a query within one round of receiving it from the verifier.
2. The response of the responder must be syntactically valid according to the expectations of the verifier, *e.g.*, if the challenger has asked for the two children of a Merkle tree inner node, these must be two hashes.
3. For the two nodes h_l and h_r returned by the responder as the children of a node h on its dirty tree, $h = H(h_l \,\|\, h_r)$.
4. If $j \geq 1$, the Merkle proof for the $(j-1)^{\mathrm{st}}$ leaf of the responder's dirty tree is valid.
5. If $j \geq 1$, $(\mathsf{tx}_{j-1}, \mathsf{tx}_j) \mid \mathbb{L}^{\cup}$.
6. If $j \geq 1$, for the claimed state commitments $\langle \mathsf{st} \rangle_{j-1}, \langle \mathsf{st} \rangle_j$, there is an underlying state st_{j-1} such that $\langle \mathsf{st}_{j-1} \rangle = \langle \mathsf{st} \rangle_{j-1}$ and $\langle \mathsf{st} \rangle_j = \langle \delta(\mathsf{st}_{j-1}, \mathsf{tx}_j) \rangle$.
7. If $j = 0$, the claimed state commitment $\langle \mathsf{st} \rangle_0$ matches the genesis state commitment $\langle \mathsf{st}_0 \rangle$ known to the verifier and $\mathsf{tx}_0 = \epsilon$.

To check condition 5, the verifier consults the already downloaded header chain (*cf.* Appendix E.2). To check condition 6, the verifier requests a *proof* from the responder that illustrates the correct state transition from $\langle \mathsf{st} \rangle_{j-1}$ to $\langle \mathsf{st} \rangle_j$, *e.g.*, the balances that were updated by tx_j (*cf.* Appendix E.3).

Responder wins. The responder wins and the challenger loses in the verifier's view if one of the following conditions fails:

1. The challenger must send valid queries. A valid query is a root number on the first round (if the responder holds multiple Merkle trees), or a single bit in any next round.
2. The challenger must not timeout, *i.e.*, must send a query within a round of being asked by the verifier.

If both parties respond according to these rules, then the responder wins.

Fig. 4: The alg. ran by the verifier to determine the winner of the bisection game.

The bisection game is played between two provers, a *challenger* and a *responder*. The challenger sends queries to the responder through the verifier. The responder replies through the same channel (*cf.* Figure 1). The challenger sends his first query to the verifier. The verifier forwards this query to the responder. The responder sends his response back to the verifier. Lastly, the verifier forwards this response to the challenger. Subsequently, the challenger follows up with more queries. As the verifier forwards queries and responses, it checks they are well-formed and the responses correspond to the queries. All communication between the two provers passes through the verifier. The challenger's goal is to convince the verifier that the responder's dirty tree root does *not* correspond to the root of a well-formed tree. The responder's goal is to defend his claim that his dirty tree root *is* the root of a well-formed tree. We design the game so that an honest challenger always wins against an adversarial responder, and an honest responder always wins against an adversarial challenger.

Algorithm 4 The algorithm ran by an honest challenger to identify the first point of disagreement against the responder's dirty tree, given that the trees have the same size ℓ, but different roots. The variable $\mathcal{T}$ represents the challenger's dirty tree.

```
function BisectionGame(𝒯, ℓ)
    if ℓ = 1 then
        return                                ▷ We are done; let the verifier check the leaf
    end if
    (h_l*, h_r*) ⇠ Responder                  ▷ Ask to open inner node
    (h_l, h_r) ← 𝒯.left.root, 𝒯.right.root
    if h_l = h_l* then
        1 ⇢ Responder
        BisectionGame(𝒯.right, ⌊ℓ/2⌋)
    else
        0 ⇢ Responder
        BisectionGame(𝒯.left, ⌊ℓ/2⌋)
    end if
end function
```

The game proceeds as a *binary search* [15,14,30]. For simplicity, let us for now assume that $\ell = \ell^*$ and they are a power of two. If $\langle \mathcal{T} \rangle \neq \langle \mathcal{T} \rangle^*$, then there must be a *first point of disagreement* between the two underlying augmented dirty ledgers $\mathbb{L}_+$ and $\mathbb{L}_+^*$ alleged by the two provers. During the game, an honest challenger tries to identify the first point of disagreement between his augmented dirty ledger $\mathbb{L}_+$ and the one the adversarial responder claims to hold. Let j be the index of that first point of disagreement pinpointed by the honest challenger. Then, the challenger asks the responder to reveal the $(j-1)^{\text{st}}$ and j^{th} entries of his augmented dirty ledger. Upon observing that the revealed entries violate the well-formedness conditions of Definition 1, the verifier concludes that the responder's tree is not well-formed. On the other hand, the honest responder replies to the adversarial challenger's queries truthfully. Therefore, the adversarial challenger cannot pinpoint any violation.

The honest challenger runs Algorithm 4, whereas the honest responder runs Algorithm 5. The verifier forwards and verifies exactly up to $\log \ell$ inner node queries and one leaf query. Then, at the end of the algorithm, the challenger arrives at the first point j of disagreement, and the honest responder reveals the leaf data $(\mathsf{tx}_j, \langle \mathsf{st}_j \rangle)$ (Algorithm 5 Line 14). Finally, if $j \geq 1$, the honest responder also sends the leaf $(\mathsf{tx}_{j-1}, \langle \mathsf{st}_{j-1} \rangle)$ at index $j-1$, along with its Merkle proof π within its dirty tree in a single round of interaction (if $j = 0$, then the verifier already knows the contents of the first leaf). This last response is only checked by the verifier and does not need to be forwarded to the challenger. For brevity, we omit this portion from the responder's algorithm.

Algorithm 5 The algorithm ran during the bisection game by the responder to reply to the challenger's queries. The variable $\mathbb{L}_+$ denotes the responder's augmented dirty ledger. The algorithm MakeMerkleTree returns the Merkle tree based on the given augmented dirty ledger.

1: **function** Respond($\mathbb{L}_+$)
2: $\quad \mathcal{T}^* \leftarrow$ MakeMerkleTree($\mathbb{L}_+$)
3: $\quad \mathcal{T}^*$.root $\dashrightarrow$ Challenger
4: $\quad$ **while** $\mathcal{T}^*$.size > 1 **do**
5: $\quad\quad (h_l^*, h_r^*) \leftarrow (\mathcal{T}^*$.left.root, $\mathcal{T}^*$.right.root$)$
6: $\quad\quad (h_l^*, h_r^*) \dashrightarrow$ Challenger
7: $\quad\quad$ dir $\dashleftarrow$ Challenger
8: $\quad\quad$ **if** dir $= 0$ **then**
9: $\quad\quad\quad \mathcal{T}^* \leftarrow \mathcal{T}^*$.left
10: $\quad\quad$ **else**
11: $\quad\quad\quad \mathcal{T}^* \leftarrow \mathcal{T}^*$.right
12: $\quad\quad$ **end if**
13: $\quad$ **end while**
14: $\quad \mathcal{T}^*$.data $\dashrightarrow$ Challenger
15: **end function**

If the responder is adversarial, she could send malformed responses. We use the notation $\langle \mathsf{st} \rangle_j$ to denote the *claimed* j^{th} state commitment by the responder, but this may be malformed and does not necessarily correspond to an actual commitment $\langle \mathsf{st}_j \rangle$, where st_j is the j^{th} state of an honest party's augmented dirty ledger. In fact, it may not be a commitment at all. Similarly, the claimed tree root $\langle \mathcal{T} \rangle^*$, provided by the adversary may not necessarily be a correctly generated Merkle tree.

C Latency-Bandwidth Trade-off for Bisection Games

This section models the latency-bandwidth trade-off realized by tuning the degree m of dirty trees, and complements the experimental results in Section 4.1. In an m-ary dirty tree representing an L-sized ledger, the tree height decreases logarithmically as the degree m of the tree increases, making the number of

Algorithm 6 The tournament among the provers administered by the verifier. It takes a sequence of provers $\mathcal{P}$, ordered from the one with the largest alleged augmented dirty ledger size to the smallest. The algorithm Challenge initiates a challenge game with the first given prover as the challenger and the second one as the responder.

```
function Tournament(P)
    sizes ← { }
    for p ∈ P do
        sizes[p] ← p.getsize()
    end for
    S ← {P[0]}
    largest ← P[0]
    for i = 1 to |P| − 1 do
        do
            if largest.getsize() > sizes[i] then
                result ← Challenge(largest, P[i])
            else
                result ← Challenge(P[i], largest)
            end if
            if result is "nested MMRs" then
                S ← S ∪ {P[i]}
            else if result is "largest loses" then
                S ← S \ {largest}
                largest ← arg max_{p∈S} sizes
            end if
            ▷ The set S is not updated if P[i] loses.
        while result is "largest loses" ∧ S ≠ ∅
        if S = ∅ then
            S ← {P[i]}
        end if
    end for
    return largest
end function
```

rounds of interactivity in the bisection game $\log_m L$. However, the challenger must now indicate the index of the child to open, making its messages $\log m$ bits in size. Similarly, when the responder opens up an inner node and reveals its children, m children need to be sent over the network. If the hash used is H bits long, then the messages sent by the responder are mH bits. There is therefore a *latency/bandwidth* tradeoff in the parameter m. A large m incurs less interactivity, but larger network messages, while a small m incurs more interactivity but shorter messages. In this section, we calculate the optimal m, given the respective network parameters on bandwidth and latency.

Let Δ be the network latency between the prover and verifier, measured in seconds, and C be the communication bandwidth of the channels connecting each prover to the verifier. We assume that, upon downloading any given message, the prover and the verifier compute the corresponding reply instantly (network latency dominates computational latency). At each round of the bisection game, $\log m$ and mH bits are downloaded by the responder and the challenger respectively. Moreover, each message sent between the responder and challenger takes 2Δ seconds to reach its destination, because it has to be forwarded through the verifier. Hence, each round is completed in $4\Delta+(mH+\log m)/C$ seconds. As the bisection game lasts for $\log_m L$ rounds, the total running time of the game becomes $(4\Delta + mH/C + \log m/C)\log_m L = \frac{\log L}{\log m}(4\Delta + mH/C) + \log L/C$. This expression is minimized for m that satisfies the expression $m(\log m - 1) = 4\Delta C/H$, *i.e.*, $m = \exp(W_1(4\Delta\frac{C}{eH}) + 1)$, where W_1 is the Lambert W function. The different optimal m for common bandwidths and latencies are plotted in Figure 5.

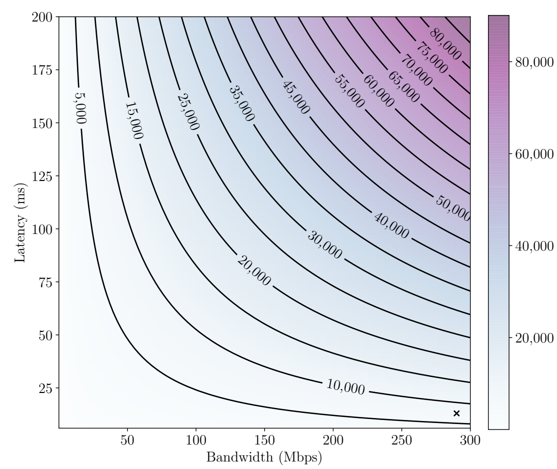

Fig. 5: Optimal Merkle tree degree m (isolines) for a given network connection bandwidth (x-axis, in Mbps) and latency (y-axis, in ms). The $\times$ marker marks the particular example described in the text.

Given[9] $C = 290$ Mbps, $\Delta = 13$ ms, $H = 256$ bits, the optimal m is 7,442, yielding dirty trees that have quite a large degree. The optimal m only depends on the network parameters and not the ledger size. Given the Ethereum ledger of $L = 1.5 \cdot 10^9$ transactions at the time of writing[10], using the optimal m for the given network parameters gives an estimate of 0.96 seconds, where 0.86 seconds of the time is due to the network delay Δ. This captures the duration of the whole bisection game with its $\log_m L$ rounds of interactivity.

D Superlight Clients

In our construction, we abstracted the checking of transaction order that the verifier performs into a consensus oracle, and discussed how this can be realized in the blockchain setting using the standard SPV technique, achieving communication complexity of $\mathcal{O}(C) = \mathcal{O}(r)$, where C is the chain size and r is the round during which the light client is booting up. This gives a total of $\mathcal{O}(C + \log L)$ communication complexity for our lazy light client protocol. However, the consensus oracle can be replaced with a *superlight* client that does not download the whole header chain, and instead samples a small portion of it. Such examples include interactive [33] or non-interactive PoPoWs and PoPoS, a primitive which can be constructed using either superblocks [35,34], FlyClient [12] or bisection games [2], and brings down the consensus oracle communication complexity to a succinct $\mathcal{O}(\text{poly} \log C) = \mathcal{O}(\text{poly} \log r)$. When composed with our protocol for identifying lazy ledger disagreements, the total communication complexity then becomes $\mathcal{O}(\text{poly} \log C + \log L) = \mathcal{O}(\text{poly} \log r)$, which is the desirable succinctness. We highlight the different roles of each protocol here: On the one hand, the superlight client, such as FlyClient or superblocks, plays the role of the *consensus oracle* and is used to answer queries about *which transaction succeeds another on the chain*; on the other hand, the interactive verification game is administered to determine the current *state of the world*, given access to such a consensus oracle. The two protocols are orthogonal and can be composed to achieve an overall performant system.

The interactivity and communication complexity for synchronization times for lazy light clients composed with different consensus oracles is illustrated in Table 1. A *Full Node* (left-most column) downloads the whole header chain of size C and every transaction of size L, thus does not need to play any interactive games, achieving constant interactivity but large communication complexity. A *Custodian Node* (right-most column) is a wallet that trusts a server to deliver correct data and does not verify it (e.g., MetaMask); this has the best performance in both complexity and interactivity. These were the only two previously known means of constructing clients for lazy blockchains. The two protocols

[9] Typical conditions for the network connection in Stanford university graduate student residences.

[10] Google Cloud Platform BigQuery table `bigquery-public-data:crypto_ethereum.transactions` as of March 6th, 2022

	Full Node	**Light Client**	**Superlight Client**	**Custodian Wallet**
Interactivity	$\mathcal{O}(1)$	$\mathcal{O}(\log L)$	$\mathcal{O}(\log L)$	$\mathcal{O}(1)$
Communication	$\mathcal{O}(C + L)$	$\mathcal{O}(C + \log L)$	$\mathcal{O}(\log CL)$	$\mathcal{O}(1)$
Decentralized	✓	✓	✓	×

Table 1: Comparison of different client types on a lazy blockchain.

titled *Light Client* and *Superlight Client* in the middle columns are clients composed with the lazy light clients explored in this work. In the light client case, an SPV client is used for the consensus oracle, while in the super light client case, a NIPoPoW superblock client is used for the consensus oracle. The C or $\log C$ term stems from the underlying consensus oracle, while the $\log L$ term stems from our lazy protocol.

E Generalizing the Model

So far, we assumed the underlying consensus protocol is blockchain-based and the computation of state mimicks Ethereum's EVM. There exist different architectures for consensus (*e.g.*, DAG-based constructions [24,44,42]) and execution (*e.g.*, UTXO [20]). Our protocol is generic and agnostic to these details. To enable the formal analysis of our construction in a generic manner, in this section we axiomatize the protocol requirements regarding consensus and execution.

E.1 Consensus Protocol

We consider a consensus protocol $\mathcal{C}$ executed by honest full nodes. Each honest full node P exposes a *read ledger* functionality which, at round r, returns a finite sequence $\mathbb{L}_r^P$ of stable transactions as the dirty ledger. They also expose a *write transaction* functionality which, given a transaction $\texttt{tx}$ at round r, attempts to include the transaction into the ledger.

Our consensus protocol must satisfy the following properties.

Definition 3 (Ledger Safety). *A consensus protocol $\mathcal{C}$ is* safe *if for all honest parties P_1, P_2 at rounds $r_1 < r_2$, $\mathbb{L}_{r_1}^{P_1}$ is a prefix of $\mathbb{L}_{r_2}^{P_2}$.*

Definition 4 (Ledger Liveness). *A consensus protocol $\mathcal{C}$ is* live *with liveness parameter u if, when an honest party P attempts to write a transaction $\texttt{tx}$ to the ledger at round r, the transaction appears in $\mathbb{L}_{r+u}^P$.*

Proof-of-work (in static and variable difficulty) and proof-of-stake protocols satisfy the above properties [25,26,36,19,18].

Primitive	Axiom	Param	Requirement
Ledger	*Safety*	-	$\mathbb{L}^{P_1}_{r_1} \preceq \mathbb{L}^{P_2}_{r_2}$
	Liveness	u	tx in $\mathbb{L}^{P}_{r+u}$
	Lipschitz	α	$\lvert\mathbb{L}^{P}_{r_2}\rvert - \lvert\mathbb{L}^{P}_{r_1}\rvert \leq \alpha(r_2 - r_1)$
Consensus	*Completeness*	-	$(\mathsf{tx}, \mathsf{tx}')$ in $\mathbb{L}^{\cup}_{r} \to \mathcal{CO}_r(\mathsf{tx}, \mathsf{tx}')$
	Soundness	ν	$(\mathsf{tx}, \mathsf{tx}')$ not in $\mathbb{L}^{\cup}_{r+\nu} \to \neg\mathcal{CO}_r(\mathsf{tx}, \mathsf{tx}')$
	Succinctness	f	$f(r) \in \mathcal{O}(\operatorname{poly}\log r)$
Execution	*Completeness*	-	$\delta(\mathsf{st}, \mathsf{tx}) = \mathsf{st}' \to \langle\delta\rangle\left(\langle\mathsf{st}\rangle, \mathsf{tx}, \pi\right) = \langle\mathsf{st}'\rangle$
	Soundness	-	$\langle\delta(\mathsf{st}, \mathsf{tx})\rangle \neq \langle\delta\rangle\left(\langle\mathsf{st}\rangle, \mathsf{tx}, \pi\right)$ is hard
	Succinctness	g	$g(r) \in \mathcal{O}(\operatorname{poly}\log r)$

Table 2: The 9 axioms required to construct a succinct light client.

Definition 5 (Ledger Lipschitz). *A consensus protocol $\mathcal{C}$ is* Lipschitz *with parameter α if for every honest party P and rounds $r_1 \leq r_2$, we have that $\lvert\mathbb{L}^{P}_{r_2}\rvert - \lvert\mathbb{L}^{P}_{r_1}\rvert < \alpha(r_2 - r_1)$.*

The above requirement states that ledgers grow at a bounded rate. For protocols that have a longest chain component such as Prism, Snap-and-Chat, Ouroboros, Babylon and Bitcoin, this follows from the fact that chains have an upper bound in their growth rate [25, Lemma 13], and that the number of transactions in each block is limited by a constant. Such a chain growth upper bound is also present for Tendermint, which Celestia is based on, as each Tendermint round has a duration of at least Δ [10]. Given the definitions of α, u and ν in the section below, we express the parameter ψ used in the suffix monologue as $\psi = \alpha(u + \nu)$.

E.2 Consensus Oracle

To enable queries about the order of transactions on the dirty ledger, we assume that the lazy blockchain protocol provides access to a *consensus oracle $\mathcal{CO}$*. The consensus oracle is a single-round black box interactive protocol executed among the verifier and the provers. The verifier invokes the oracle with input two transactions $(\mathsf{tx}, \mathsf{tx}')$ and receives a boolean response. The goal of the verifier is to determine whether a transaction tx' *immediately follows* another transaction tx on $\mathbb{L}^{\cup}$ (*i.e.*, $(\mathsf{tx}, \mathsf{tx}') \mid \mathbb{L}^{\cup}$).

We require that the consensus oracle satisfies the following properties.

Definition 6 (Consensus Oracle Security). *An consensus oracle is* secure *if it satisfies:*

– ***Completeness.*** $(\mathsf{tx}, \mathsf{tx}') \mid \mathbb{L}_r^{\cup} \Rightarrow \mathcal{CO}_r(\mathsf{tx}, \mathsf{tx}')$.
– ***Soundness.*** *The consensus oracle is sound with* delay parameter ν *if for any PPT adversary* $\mathcal{A}$*,* $\Pr[(\mathsf{tx}, \mathsf{tx}', r) \leftarrow \mathcal{A}(1^\lambda); \mathcal{CO}_r(\mathsf{tx}, \mathsf{tx}') \wedge (\mathsf{tx}, \mathsf{tx}') \nmid \mathbb{L}_{r+\nu}^{\cup}] \leq \mathrm{negl}(\lambda)$.

Definition 7 (Consensus Oracle Communication Complexity). *A consensus oracle has communication complexity $f(r)$ if the total size of the query and response messages exchanged during an oracle query invoked at round r is $f(r) \in \mathcal{O}(r)$.*

We assume that every transaction in the dirty ledger is unique and there are no duplicate transactions. Under this assumption, a consensus oracle on top of the Nakamoto longest chain consensus protocol can be instantiated as follows. Since the construction below also applies to protocols that output a chain of blocks, we will refer to the longest chain as the *canonical* chain.

The blockchain consists of a header chain, each header containing the Merkle tree root, *i.e.* the transaction root, of the transactions organized within the associated block. The ordering of the blocks by the header chain together with the ordering of the transactions by each Merkle tree determine the total order across all transactions. Thus, to query the consensus oracle with the two transactions tx and $\mathsf{tx}' \neq \mathsf{tx}$, the verifier first downloads all the block headers from the honest provers and determines the canonical stable header chain. Then, it asks a prover if tx immediately precedes tx' on its dirty ledger. To affirm, the prover replies with (a) the positions i and i' of the transactions tx and tx' within their respective Merkle trees, (b) the Merkle proofs π and π' from the transactions tx and tx' to the transaction roots, (c) the positions $j \leq j'$ of the block headers containing these transaction roots, on the canonical stable header chain.

Then, the verifier checks that the Merkle proofs are valid, and accepts the prover's claim iff either of (1) the two blocks are the same, *i.e.*, $j' = j$, and $i' = i + 1$, or (2) otherwise, the two blocks are consecutive, *i.e.*, $j' = j + 1$, and i is the index of the last leaf in the tree of block j while i' is the index of the first leaf in the tree of block $j' = j + 1$.

If no prover is able to provide such a proof, the oracle returns *false* to the verifier. The oracle's soundness follows the ledger safety.

The above is one example instantiation of a consensus oracle. Appendix F gives proofs of completeness and soundness for the consensus oracle as well as notes on how it can be implemented on different blockchain protocols.

To relax the uniqueness assumption for the transactions in the dirty ledger, each augmented dirty ledger entry containing a transaction tx can be extended by adding the index j_{tx} of the header of the block containing tx, and the index of tx within the Merkle tree of that block. In this case, the verifier queries the consensus oracle not only with transactions tx and $\mathsf{tx}' \neq \mathsf{tx}$, but also with the corresponding block header and Merkle tree indices j_{tx}, i_{tx} and $j_{\mathsf{tx}'}$, $i_{\mathsf{tx}'}$. Hence, during the query, the verifier also checks if the block and transaction indices for tx and tx', *e.g.*, j, i and j', i', received from the prover matches the claimed indices: $j = j_{\mathsf{tx}}$, $j' = j_{\mathsf{tx}'}$, $i = i_{\mathsf{tx}}$, $i' = i_{\mathsf{tx}'}$.

E.3 Execution Oracle

To enable queries about the validity of state execution, we assume that the lazy blockchain protocol provides access to an *execution oracle*. The execution oracle is a single-round black box interactive protocol executed among the verifier and the provers. The verifier invokes the oracle with a transaction tx and two state commitments, $\langle \mathsf{st} \rangle$ and $\langle \mathsf{st} \rangle'$ as input, and receives a boolean response. The goal of the verifier is to determine whether there exists a state st such that $\langle \mathsf{st} \rangle$ is the commitment of st and $\langle \mathsf{st} \rangle' = \langle \delta(\mathsf{tx}, \mathsf{st}) \rangle$.

The execution oracle is parametrized by a triplet $(\delta, \langle \cdot \rangle, \langle \delta \rangle)$ consisting of an efficiently computable *transition function* $\delta(\cdot, \cdot)$, a *commitment scheme* $\langle \cdot \rangle$, and a *succinct transition function* $\langle \delta \rangle (\cdot, \cdot, \cdot)$. The succinct transition function $\langle \delta \rangle$ accepts a state commitment $\langle \mathsf{st} \rangle$, a transaction tx, and a proof π, and produces a new state commitment $\langle \mathsf{st} \rangle'$ which corresponds to the commitment of the updated state.

To query the execution oracle on tx, $\langle \mathsf{st} \rangle$ and $\langle \mathsf{st} \rangle'$, the verifier first asks a prover for a proof π. If the prover claims that he knows a state st such that $\langle \mathsf{st} \rangle$ is the commitment of st and $\langle \mathsf{st} \rangle' = \langle \delta(\mathsf{tx}, \mathsf{st}) \rangle$, it gives a proof π. Then, the verifier accepts the prover's claim if $\langle \mathsf{st} \rangle' = \langle \delta \rangle (\langle \mathsf{st} \rangle, \mathsf{tx}, \pi)$. Otherwise, if $\langle \delta \rangle$ throws an error or outputs a different commitment, the verifier rejects the claim.

Definition 8 (Execution Oracle Security). *An execution oracle is* secure *if it satisfies:*
Completeness. *Execution oracle is* complete *with respect to a proof-computing PPT machine M if for any state* st *and transaction* tx*, it holds that* $M(\mathsf{st}, \mathsf{tx})$ *outputs a π that satisfies* $\langle \delta \rangle (\langle st \rangle, \mathsf{tx}, \pi) = \langle \delta(\mathsf{st}, \mathsf{tx}) \rangle$.
Soundness. *For any PPT adversary $\mathcal{A}$:*

$$\Pr[(\mathsf{st}, \mathsf{tx}, \pi) \leftarrow \mathcal{A}(1^\lambda); \langle \delta(\mathsf{st}, \mathsf{tx}) \rangle \neq \langle \delta \rangle (\langle \mathsf{st} \rangle, \mathsf{tx}, \pi)] \leq \mathrm{negl}(\lambda) .$$

Definition 9 (Execution Oracle Communication Complexity). *An execution oracle has communication complexity $g(r)$ if the total size of the query and response messages exchanged during an oracle query invoked at round r is $g(r) \in \mathcal{O}(r)$.*

In the account based model [4], the state is a Sparse Merkle Tree (SMT) [17] representing a key-value store. The values constitute the leaves of the SMT and the keys denote their indices. The state commitment corresponds to the root.

The verifier queries the execution oracle with tx, $\langle \mathsf{st} \rangle$ and $\langle \mathsf{st} \rangle'$. Suppose there is a state st with commitment $\langle \mathsf{st} \rangle$ and $\langle \mathsf{st} \rangle' = \langle \delta(\mathsf{st}, \mathsf{tx}) \rangle$. Let D denote the leaves of the SMT st. Let $\mathcal{S}_{\mathsf{tx}}$ be the keys of the SMT that the transaction tx reads from or writes to[11]. We assume that the number of leaves touched by a particular transaction is constant. Then, the proof required by $\langle \delta \rangle$ consists of:

- The key-value pairs $(i, D[i])$ for $i \in \mathcal{S}_{\mathsf{tx}}$ within st.
- The Merkle proofs π_i, $i \in \mathcal{S}_{\mathsf{tx}}$, from the leaves $D[i]$ to the root $\langle \mathsf{st} \rangle$.

[11] In Ethereum, these can be obtained by the verifier via the eth_createAccessList RPC.

Given the components above, $\langle\delta\rangle$ verifies the proofs π_i and the validity of tx with respect to the pairs $(i, D[i])$, *e.g.*, tx should not be spending from an account with zero balance. If there are pairs read or modified by tx that have not been provided by the prover, then $\langle\delta\rangle$ outputs $\perp$. If all such key-value pairs are present and tx is invalid with respect to them, $\langle\delta\rangle$ outputs $\langle\mathsf{st}\rangle$, and does not modify the state commitment. Otherwise, $\langle\delta\rangle$ modifies the relevant key-value pairs covered by $\mathcal{S}_{\mathsf{tx}}$, which can be done efficiently [14]. Finally, it calculates the new SMT root, *i.e.* the new state commitment, using the modified leaves and the corresponding Merkle proofs among π_i, $i \in \mathcal{S}_{\mathsf{tx}}$.

SMTs can also be used to represent states based on the UTXO [38] model. In this case, the value at each leaf of the SMT is a UTXO. Thus, the execution oracle construction above generalizes to the UTXO model.

F Consensus Oracle Constructions

Consensus oracle constructions for Celestia (LazyLedger) [3], Prism [5], and Snap-and-Chat [40,39] follow the same paradigm described in Section E.2.

F.1 Celestia

Celestia uses Tendermint [10] as its consensus protocol, which outputs a chain of blocks containing transactions. Blocks organize the transactions as namespaced Merkle trees, and the root of the tree is included within the block header. Hence, the construction of Section E.2 can be used to provide a consensus oracle for Celestia.

Celestia is designed as a data availability and consensus layer for multiple rollups. However, as Celestia is a lazy blockchain, each rollup on Celestia (called 'sovereign rollups') also need a mechanism for their *rollup light clients* to discover the correct latest rollup state. Towards this goal, our succinct light client construction can be utilized by the rollup nodes to support these light clients. For instance, as rollups are maintained by full nodes that execute the rollup-specific transactions (ignoring other transactions) posted to Celestia, these nodes can aid the rollup light clients by creating a dirty ledger of rollup-specific transactions, the corresponding dirty trees and MMRs, in the same way as the full nodes of a lazy blockchain with a single state transition function would help its light clients discover the correct latest state.

F.2 Prism

In Prism, a transaction tx is first included within a transaction block. This block is, in turn, referred by a proposer block. Once the proposer block is confirmed in the view of a prover $\mathcal{P}$ at round r, tx enters the ledger $\mathbb{L}^{\mathcal{P}}_r$. Hence, the proof of inclusion for tx consists of two proofs: one for the inclusion of tx in a transaction block B_T, the other for the inclusion of the header of B_T in a proposer block B_P. If transactions and transaction blocks are organized as Merkle trees, then, the proof of inclusion for tx would be two Merkle proofs: one from tx to the Merkle root in the header of B_T, the other from the header of B_T to the Merkle root in the header of B_P.

The construction of Section E.2 can be generalized to provide a consensus oracle for Prism. In this case, to query the consensus oracle with two transactions tx and $\mathsf{tx}' \neq \mathsf{tx}$, the verifier first downloads all the proposal block headers from the honest provers and determines the longest stable header chain. Then, it asks a prover if tx immediately precedes tx' on its dirty ledger.

To affirm, the prover replies with:

- the positions i_t and i'_t of the transactions tx and tx' within their respective Merkle trees contained in the respective transaction blocks B_T and B'_T.
- the Merkle proofs π_t and π'_t from the transactions tx and tx' to the corresponding Merkle roots within the headers of B_T and B'_T,
- the positions i_p and i'_p of the headers of the transaction blocks B_T and B'_T within their respective Merkle trees contained in the respective proposal blocks B_P and B'_P.
- the Merkle proofs π_p and π'_p from the headers of B_T and B'_T to the corresponding Merkle roots within the headers of B_P and B'_P,
- the positions $j \leq j'$ of the headers of B_P and B'_P on the longest stable header chain.

Then, the verifier checks that the Merkle proofs are valid, and accepts the prover's claim if and only if either of the following cases hold:

1. if $j = j'$ and $i'_p = i_p$, then $i'_t = i_t + 1$.
2. if $j = j'$ and $i'_p = i_p + 1$, then i_t is the index of the last leaf in the tree of B_T while i'_t is the index of the first leaf in the tree of B'_T.
3. if $j' > j$, then i_p is the index of the last leaf in the tree of B_P while i'_p is the index of the first leaf in the tree of B'_P. Similarly, i_t is the index of the last leaf in the tree of B_T while i'_t is the index of the first leaf in the tree of B'_T.

If no prover is able to provide such a proof, the oracle returns *false* to the verifier.

F.3 Snap-and-Chat

In Snap-and-Chat protocols, a transaction tx is first included within a block B_T proposed in the context of a longest chain protocol. Upon becoming k-deep within the longest chain, where k is a predetermined parameter, B_T is, in turn, included within a block B_P proposed as part of a partially-synchronous BFT protocol. Once B_P is finalized in the view of a prover $\mathcal{P}$ at round r, tx enters the *finalized ledger* $\mathbb{L}_r^{\mathcal{P}}$. There are again two Merkle proofs for verifiable inclusion, one from tx to the Merkle root included in B_T, the other from the header of B_T to the Merkle root in the header of B_P. Consequently, the consensus oracle construction for Prism also applies to Snap-and-Chat protocols.

Remark 1. We remark here that protocols that do not follow the longest chain rule, or are not even proper chains, can be utilized by our protocol. Such examples include Parallel Chains [24], PHANTOM [44] / GHOSTDAG [43], and SPECTRE [42]. The only requirement is that these systems provide a succinct means of determining whether two transactions follow one another on the ledger.

F.4 Colored Coins and Babylon

Colored coins [1] refer to assets other than Bitcoin that are maintained on the Bitcoin blockchain, and derive their security from the consensus security of Bitcoin. Babylon [46] is a protocol that checkpoints off-the-shelf PoS protocols onto Bitcoin to mitigate PoS-related problems such as non-slashable posterior corruption attacks, low liveness resilience and difficulty to bootstrap from low token valuation. To post checkpoints and other types of data, Babylon and other colored coin applications use the OP_RETURN scripting code, which allows arbitrary data to be recorded in an unspendable Bitcoin transaction. Since the miners do not check the validity of the data within the OP_RETURN transactions with respect to the corresponding application state, Bitcoin acts as a lazy blockchain towards these applications. Section F.5 describes how a consensus oracle can be instantiated for longest chain protocols such as Bitcoin.

In the case of Babylon, provers can interact with an consensus oracle after the bisection game to prove the validity of the checkpoint at the first point of disagreement. To support an consensus oracle, each checkpoint posted to Bitcoin must be augmented by the *active validator set* of the portion of the PoS protocol ledger corresponding to the checkpoint (*cf.* [46][Sections IV-C and V-B]). To verify the validity of the disputed checkpoint (via [46][Algorithms 1 and 2]), the verifier has to read only a constant-size portion of the PoS protocol ledger, namely the portion between the chain checkpointed by the earlier, common checkpoint, and the latter checkpoint at the source of disagreement.

F.5 Longest Chain

As an illustration of how the consensus oracle can be realized in a longest header chain protocol, we provide sketches for the proofs of consensus oracle completeness and soundness in the Nakamoto setting.

Even in the original Nakamoto paper [38], a description of an SPV client is provided, and it realizes our consensus oracle axioms, although these virtues were not stated or proven formally. The consensus oracle works as follows. The verifier connects to multiple provers, at least one of which is assumed to be honest. It inquires of the provers their longest chains, downloads them, verifies that they are chains and that they have valid proof-of-work, and keeps the heaviest chain. It then chops off k blocks from the end to arrive at the stable part. Upon being queried on two transactions $(\mathsf{tx}, \mathsf{tx}')$, the oracle inquires of its provers whether these transactions follow one another on the chain. To prove that they do, the honest prover reveals two Merkle proofs of inclusion for tx and tx'. These must appear in either consecutive positions within the same block header, or at the last and first position in consecutive blocks.

The terminology of *typical executions*, the Common Prefix parameter k, and the Chain Growth parameter τ are borrowed from the Bitcoin Backbone [25] line of works, where these properties are proven. We leverage these properties to show that our Consensus Oracle satisfies our desired axioms. Our proofs are in the static synchronous setting, but generalize to the Δ-bounded delay and variable difficulty settings.

Lemma 2 (Nakamoto Completeness). *In typical executions where honest majority is observed, the Nakamoto Consensus Oracle is complete.*

Sketch. We prove that, if $(\mathsf{tx}, \mathsf{tx}')$ are reported in $\mathbb{L}^{\cup}$, then an honest prover will be able to prove so. Suppose the verifier chose a longest header chain C^V. If $(\mathsf{tx}, \mathsf{tx}')$ appear consecutively in $\mathbb{L}^{\cup}$, by ledger safety, this means that they belong to the ledger of at least one honest party P who is acting as a prover. Since $(\mathsf{tx}, \mathsf{tx}')$ appear consecutively in $\mathbb{L}^P$, therefore they appear in the stable portion $C^P[: -k]$ of the chain C^P held by P. By the Common Prefix property, $C^P[: -k]$ is a prefix of C^V and therefore $(\mathsf{tx}, \mathsf{tx}')$ appear consecutively in the stable header chain adopted by the verifier. Therefore, the verifier accepts. □

Lemma 3 (Nakamoto Soundness). *In typical executions where honest majority is observed, the Nakamoto Consensus Oracle instantiated with a Merkle Tree that uses a collision resistant hash function is sound, with soundness parameter $\nu = \frac{k}{\tau}$ where k is the Common Prefix parameter and τ is the Chain Growth parameter.*

Sketch. Suppose for contradiction that $(\mathsf{tx}, \mathsf{tx}')$ are not reported in $\mathbb{L}^{\cup}_{r+\nu}$, yet the adversary convinces the verifier of this at round r. This means that the adversary has presented some header chain C^V to the verifier which was deemed to be the longest at the time, and $(\mathsf{tx}, \mathsf{tx}')$ appear in its stable portion $C^V[: -k]$. Consider an honest prover P. At time r, the honest prover holds a chain C^P_r and at round $r + \nu$, it holds a chain $C^P_{r+\nu}$. By the Common Prefix property, $C^V[: -k]$ is a prefix of C^P_r and of $C^P_{r+\nu}$. Furthermore, $(\mathsf{tx}, \mathsf{tx}')$ will appear in the same block (or consecutive blocks) in all three. By the Chain Growth property, $C^P_{r+\nu}$ contains at least k more blocks than C^P_r. Therefore, $(\mathsf{tx}, \mathsf{tx}')$ appears in $C^P_{r+\nu}[: -k]$ and are part of the stable chain at round $r + \nu$ for party P. They are hence reported in $\mathbb{L}^P_{r+\nu} \subseteq \mathbb{L}^{\cup}_{r+\nu}$, which is a contradiction. Finally, by Proposition 1, proofs of inclusion for tx and tx' cannot be forged with respect to Merkle roots in block headers other than those in C^V, that were initially shown to contain tx and tx' (except with negligible probability). □

Proofs of correctness and soundness for the consensus oracle constructions of Celestia, Prism and Snap-and-Chat follow a similar pattern to the proofs for the Nakamoto setting.

Lastly, for succinctness, one must leverage a construction such as superblock NIPoPoWs [35]. Here, proofs of the longest chain are $\text{poly} \log C$ where C denotes the chain size. Transaction inclusion proofs make use of *infix proofs* [35] in addition to Merkle Tree proofs of inclusion into block headers. As $C \in \mathcal{O}(\text{poly}\log r)$, these protocols are $\mathcal{O}(\text{poly}\log r)$ as desired. Completeness and soundness follow from the relevant security proofs of the construction.

G Proofs

Our proof structure is as follows. First, we prove some facts about the bisection game, in particular its succinctness, soundness, and completeness. We later leverage these results to show that our full game enjoys the same virtues. This section is based on the generalized model for lazy blockchains presented in Section E, and the axioms used by the proofs are given by Table 2.

Lemma 4 (Bisection Succinctness). *Consider a consensus oracle and an execution oracle with f and g communication complexity respectively. Then, the bisection game invoked at round r with trees of size ℓ ends in* $\log(\ell)$ *rounds of communication and has a total communication complexity of* $O(\log \ell + f(r) + g(r))$.

Proof. When the dirty trees have ℓ leaves, there can be at most $\log \ell$ valid queries, as the verifier aborts the game after $\log \ell$ queries. Hence, the bisection game ends in $\log \ell$ rounds of interactivity.

At each round of communication, the challenger indicates whether he wants the left or the right child to be opened (which can be designated by a constant number of bits), and the responder replies with two constant size hash values. At the final round, the responder returns $(\mathsf{tx}_{j-1}, \langle \mathsf{st} \rangle_{j-1})$ and $(\mathsf{tx}_j, \langle \mathsf{st} \rangle_j)$, the augmented dirty ledger entries at indices $j-1$ and j, along with the Merkle proof for the $j-1^{\text{st}}$ entry (Alternatively, it only returns $(\mathsf{tx}_0, \langle \mathsf{st} \rangle_0)$). The entries have constant size since transactions and state commitments are assumed to have constant sizes. The Merkle proof consists of $\log \ell$ constant size hash values. Consequently, the total communication complexity of the bisection game prior to the oracle queries becomes $O(\log \ell)$.

Finally, the verifier queries the consensus oracle on $(\mathsf{tx}_{j-1}, \mathsf{tx}_j)$ and the execution oracle on $(\langle \mathsf{st} \rangle_{j-1}, \mathsf{tx}_j, \langle \mathsf{st} \rangle_j)$ with $O(f(r))$ and $O(g(r))$ communication complexity. Hence, the total communication complexity of the bisection game becomes $O(\log(\ell) + f(r) + g(r))$. □

Lemma 5 (Bisection Completeness). *Suppose the consensus and execution oracles are complete and the ledger is safe. Then, the honest responder wins the bisection game against any PPT adversarial challenger.*

Proof. We will enumerate the conditions checked by the verifier in Algorithm 4 to show that the honest responder always wins.

The honest responder replies to each valid query from the verifier, and the replies are syntactically valid. Hence, conditions (1) and (2) of Algorithm 4 cannot fail.

By the construction of the honest responder's Merkle tree, each inner node h queried by the challenger satisfies $h = H(h_l \,\|\, h_r)$ for its children h_l, h_r returned in response to the query. For the same reason, the Merkle proof given by the responder is valid. Hence, conditions (3) and (4) cannot fail either.

If $j = 0$, by the well-formedness of the responder's dirty ledger, $(\mathsf{tx}_j, \langle \mathsf{st} \rangle_j) = (\epsilon, \langle st_0 \rangle)$, so condition (7) cannot fail.

Let r denote the round at which the bisection game was started. If $j \geq 1$, by the well-formedness of the responder's dirty tree, for any consecutive pair of leaves at indices $j-1$ and j, it holds that $(\mathsf{tx}_{j-1}, \mathsf{tx}_j) \mid \mathbb{L}_r^{\mathcal{P}} \preceq \mathbb{L}_r^{\cup}$ due to ledger safety. As the consensus oracle is complete, by Definition 6, it returns *true* on $(\mathsf{tx}_{j-1}, \mathsf{tx}_j)$, implying that the condition (5) cannot fail.

Finally, by the well-formedness of the responder's dirty tree, for any consecutive pair of leaves at indices $j-1$ and j, there exist a state st_{j-1} such that $\langle \mathsf{st} \rangle_{j-1} = \langle \mathsf{st}_{j-1} \rangle$, and $\langle \mathsf{st} \rangle_j = \langle \delta(\mathsf{st}_{j-1}, \mathsf{tx}_j) \rangle$. As the execution oracle is complete, by Definition 8, $M(\mathsf{st}_{j-1}, \mathsf{tx}_j)$ outputs a proof π such that $\langle \delta \rangle (\langle \mathsf{st}_{j-1} \rangle, \mathsf{tx}_j, \pi) = \langle \delta(\mathsf{st}_{j-1}, \mathsf{tx}_j) \rangle$. Using the observations above, $\langle \delta \rangle (\langle \mathsf{st} \rangle_{j-1}, \mathsf{tx}_j, \pi) = \langle \mathsf{st} \rangle_j$. Consequently, condition (6) cannot fail. Thus, the honest responder wins the bisection game against any adversary. □

Let $\textsc{Verify}(\pi, \langle \mathcal{T} \rangle, \ell, i, v)$ 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. The following proposition is a well-known folklore result about the security of Merkle trees, stating that it is impossible to prove proofs of inclusion for elements that were not present during the tree construction. It extends the result that Merkle trees are collision resistant [32].

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

Proof. Suppose $\mathcal{A}$ is the adversary of the statement. We will construct a hash collision adversary $\mathcal{A}'$ that calls $\mathcal{A}$ as a subroutine. The adversary $\mathcal{A}'$ works as follows. It invokes $\mathcal{A}(1^\lambda)$, and obtains v, D, π, i. Let $h_1^*, \ldots, h_{a-1}^*$ denote the hash values within π, where $a = \log|\ell| + 1$ is the height of the Merkle tree. Let $h_1, \ldots, h_{a-1}$ denote the inner nodes within the Merkle tree at the positions that correspond to those of $h_1^*, \ldots h_{a-1}^*$. Let $\tilde{h}_1, \ldots, \tilde{h}_{a-1}$ denote the siblings of $h_1, \ldots, h_{a-1}$. Define $\tilde{h}_a := \langle \mathcal{T} \rangle$. Then, $\tilde{h}_1 = H(D[i])$, and for $i = 1, \ldots, a-1$;

- If h_i is the left child of its parent, $\tilde{h}_{i+1} = H(h_i \,\|\, \tilde{h}_i)$.
- If h_i is the right child of its parent, $\tilde{h}_{i+1} = H(\tilde{h}_i \,\|\, h_i)$.

Consider the event $\textsc{Merkle-Collision}$ that $\mathcal{A}$ succeeds. In that case, there exists a sequence of hash values $\tilde{h}_1^*, \ldots, \tilde{h}_a^*$ such that $\tilde{h}_1^* = H(v)$, $\tilde{h}_a^* = \langle \mathcal{T} \rangle$, and for $i = 1, \ldots, a-1$,

- If h_i is the left child of its parent, $\tilde{h}_{i+1}^* = H(h_i^* \,\|\, \tilde{h}_i^*)$.
- If h_i is the right child of its parent, $\tilde{h}_{i+1}^* = H(\tilde{h}_i^* \,\|\, h_i^*)$.

Finally, for $i = 1, \ldots, a$, define $h_{i,m}$ and $h_{i,c}$ as follows:

- $h_{a,m} = \langle \mathcal{T} \rangle$, $h_{a,c} = \langle \mathcal{T} \rangle$.
- $h_{0,m} = v$, $h_{0,c} = D[i]$.
- If h_i is the left child of its parent, $h_{i,m} = h_i^* \parallel \tilde{h}_i^*$ and $h_{i,c} = h_i \parallel \tilde{h}_i$.
- If h_i is the right child of its parent, $h_{i,m} = \tilde{h}_i^* \parallel h_i^*$ and $h_{i,m} = \tilde{h}_i \parallel h_i$.

Finally, the adversary $\mathcal{A}'$ finds the first index p for which there is a collision

$$H(h_{i,m}) = H(h_{i,c}) \text{ and } h_{i,m} \neq h_{i,c}$$

and returns $a := h_{p,m}$ and $b := h_{p,c}$, if such an index p exists. Otherwise, it returns FAILURE.

In the case of MERKLE-COLLISION, for $i = 0, \ldots, a-1$, $h_{i+1,m} = H(h_{i,m})$, $h_{i+1,c} = H(h_{i,c})$. As $v \neq D[i]$, a collision must have been found for at least one index $p \in [h-1]$. Therefore, $\Pr[\mathcal{A}' \text{ succeeds}] = \Pr[\text{MERKLE-COLLISION}]$.

However, since $\forall$ PPT $\mathcal{A}'$:

$$\Pr[(a,b) \leftarrow \mathcal{A}'(1^\lambda) : a \neq b, H(a) = H(b)] \leq \text{negl}(\lambda)\,,$$

therefore, $\Pr[\text{MERKLE-COLLISION}] = \text{negl}(\lambda)$. □

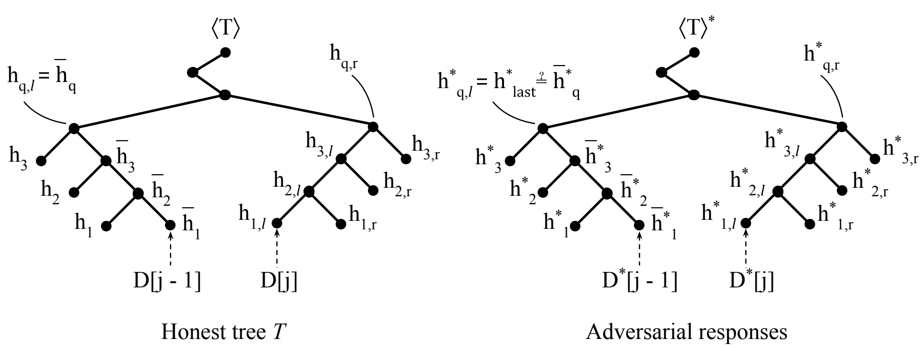

Fig. 6: The world in the view of the proof of Lemma 6. Starred quantities (right-hand side) denote adversarially provided values. Unstarred quantities (left-hand side) denote the respective honestly provided values. The inner node at height q from the leaves is the level containing the lowest common ancestor between leaves with indices j and $j-1$.

The next lemma establishes an important result for our bisection game: That the honest challenger can pinpoint the *first point of disagreement* or *last point of agreement* indices $j-1$ and j within the responder's claimed tree. The result stems from the fact that the data are organized into a Merkle tree which can be explored, moving left or right, one level at a time, ensuring the invariant that *the first point of disagreement remains within the subtree explored* at every step.

Lemma 6 (Bisection Pinpointing). *Let H^s be a collision resistant hash function. Consider the following game among an honest challenger $\mathcal{P}$, a verifier $\mathcal{V}$ and an adversarial responder $\mathcal{P}^*$: The challenger $\mathcal{P}$ receives an array D of size ℓ from $\mathcal{P}^*$, and calculates the corresponding dirty tree $\mathcal{T}$ with root $\langle\mathcal{T}\rangle$. Then, $\mathcal{P}$ plays the bisection game against $\mathcal{P}^*$ claiming root $\langle\mathcal{T}\rangle^* \neq \langle\mathcal{T}\rangle$ and size ℓ. Finally, $\mathcal{V}$ outputs $(1, D^*[j-1], D^*[j])$ if $\mathcal{P}$ wins the bisection game; otherwise, it outputs $(0, \bot, \bot)$. Here, $D^*[j-1]$ and $D^*[j]$ are the two entries revealed by $\mathcal{P}^*$ for the consecutive indices $j-1$ and j during the bisection game. ($D^*[-1]$ is defined as $\bot$ if $j = 0$.) Then, for all PPT adversarial responder $\mathcal{A}$,* $\Pr[D \leftarrow \mathcal{A}(1^\lambda); (1, D^*[j-1], D^*[j]) \leftarrow \mathcal{V} \wedge (D^*[j-1] \neq D[j-1] \vee D^*[j] = D[j])] \leq \text{negl}(\lambda)$.

Proof. Consider a PPT adversarial responder $\mathcal{P}^*$ playing the bisection game against the honest challenger $\mathcal{P}$ at some round r. Since the challenger is honest, his queries are valid and he does not time out. For the responder to win the bisection game, she must satisfy all the conditions checked by Algorithm 4.

Consider the event Bad that the responder wins. Conditioned on Bad, the responder does not timeout and her replies are syntactically valid. Let $a = \log \ell + 1$ denote the height of the challenger's dirty tree. At each round $i \in [a-1]$ of interactivity in the bisection game, the responder reveals two hash values $h^*_{a-i,l}$ and $h^*_{a-i,r}$. The subscript $a-i$ signifies the alleged height of the nodes $h^*_{a-i,l}$ and $h^*_{a-i,r}$. Let $h_{i,l}$ and $h_{i,r}$ denote the inner nodes in the honest challenger's dirty tree with the same positions as $h^*_{i,l}$ and $h^*_{i,r}$. These will always exist, as the verifier limits the rounds of interaction to a.

At the first round, the responder reveals $h^*_{a-1,l}$ and $h^*_{a-1,r}$ as the alleged left and right children of its dirty tree root $\langle\mathcal{T}\rangle^*$. By condition (3) of Algorithm 4, $H(h^*_{a-1,l} \,\|\, h^*_{a-1,r}) = \langle\mathcal{T}\rangle^*$. However, since $\langle\mathcal{T}\rangle^* \neq \langle\mathcal{T}\rangle = H(h_{a-1,l} \,\|\, h_{a-1,r})$, either $h_{a-1,l} \neq h^*_{a-1,l}$ or $h_{a-1,r} \neq h^*_{a-1,r}$ or both. Then, if $h_{a-1,l} \neq h^*_{a-1,l}$, the challenger picks $h^*_{a-1,l}$ to query next; else, he picks $h^*_{a-1,r}$.

We observe that if a node $h^* = h^*_{i,l}$ or $h^* = h^*_{i,r}$, $i \in \{2, \dots, a-1\}$, returned by the responder, is queried by the honest challenger,

- For the two children $h^*_{i-1,l}$ and $h^*_{i-1,r}$ of h^*, it holds that $h^* = H(h^*_{i-1,l} \,\|\, h^*_{i-1,r})$ by condition (3).
- For the nodes h, $h_{i-1,l}$ and $h_{i-1,r}$ in the challenger's dirty tree that have the same positions as h^*, $h^*_{i-1,l}$ and $h^*_{i-1,r}$; $h = H(h_{i-1,l} \,\|\, h_{i-1,r})$, and $h \neq h^*$.
- By implication, either $h_{i-1,l} \neq h^*_{i-1,l}$ or $h_{i-1,r} \neq h^*_{i-1,r}$ or both. If $h_{i-1,l} \neq h^*_{i-1,l}$, the challenger picks $h^*_{i-1,l}$ as its next query; else, it picks $h^*_{i-1,r}$ as its next query.

The queries continue until the challenger queries a node $h^* = h_{1,l}$ or $h^* = h_{1,r}$ returned by the responder, and the responder reveals the leaf $D^*[j]$ such that $H(D^*[j]) = h^*$. By induction, h^* is different from the node $h = H(D[j])$ with the same position in the challenger's dirty tree. Thus, $D^*[j] \neq D[j]$.

If $j = 0$, it must hold that $D^*[0] = (\epsilon, \langle st_0\rangle)$ by condition (7) of Algorithm 4. However, since the challenger's dirty tree is well-formed, $D[0] = (\epsilon, \langle st_0\rangle)$ as well. Hence, $D^*[0] = D[0]$, therefore necessarily $j > 0$.

(When the provers hold MMRs instead of Merkle treees, responder's augmented dirty ledger entries $D^*[j-1]$ and $D^*[j]$ can lie in different Merkle trees held by the responder. In this case, since the honest challenger did not initiate a bisection game between the responder's peak $\mathcal{T}_i^*$ containing $D^*[j-1]$ and his corresponding inner node $\mathcal{T}_i = \mathcal{T}_i^*$, $D^*[j-1] = D[j-1]$ with overwhelming probability. To show this, we construct the PPT Merkle tree adversary that outputs $D^*[j-1]$, the honest challenger's leaves under $\mathcal{T}_i$, the responder's Merkle proof π^* for $D^*[j-1]$ with respect to $\mathcal{T}_i = \mathcal{T}_i^*$ and the index of $D[j-1]$ within the subtree of $\mathcal{T}_i$, if $D^*[j-1] \neq D[j-1]$; and FAILURE otherwise. Since this adversary succeeds except with negligible probability in λ, $D^*[j-1] = D[j-1]$ with overwhelming probability, and this concludes the proof. In the rest of this section, we assume that $j-1$ and j lie in the same Merkle tree of the responder.)

As $j > 0$, there must exist a *last* node queried by the challenger such that for its children $h^*_{q,l}$ and $h^*_{q,r}$ revealed by the responder at height q, it holds that $h^*_{q,l} = h_{q,l}$ and $h^*_{q,r} \neq h_{q,r}$ (This is the last time the challenger went *right*). Define $h^*_{\text{last}} = h^*_{q,l}$ (see Figure 6).

By condition (4) of Algorithm 4, the Merkle proof π^* for $D^*[j-1]$ is valid with respect to $\langle \mathcal{T} \rangle^*$. Let $h^*_1, h^*_2, \ldots, h^*_{a-1}$ denote the sequence of nodes on π^* Let $\tilde{h}^*_1 := H(D^*[j-1])$ and define $\tilde{h}^*_{i+1}$, $i = 1, \ldots, a-1$, recursively as follows: $\tilde{h}^*_{i+1} := H(h^*_i \,\|\, \tilde{h}^*_i)$ if h^*_i is the left child of its parent, and, $\tilde{h}^*_{i+1} := H(\tilde{h}^*_i \,\|\, h^*_i)$ if h^*_i is the right child of its parent. Since π^* is valid, $\tilde{h}^*_a = \langle \mathcal{T} \rangle^*$ (The nodes $\tilde{h}^*_i$, $i \in [a-1]$, are the alleged nodes on the path connecting $D^*[j-1]$ to the root $\langle \mathcal{T} \rangle^*$, and h^*_i are their alleged siblings).

Let h_i, $i \in [a-1]$, denote the inner nodes in the challenger's dirty tree with the same positions as h^*_i. Let $\tilde{h}_i$, $i \in [a-1]$, denote the inner nodes in the challenger's dirty tree on the path from $D[j-1]$ to $\langle \mathcal{T} \rangle$. These inner nodes satisfy the following relations for $i \in [a-1]$: $\tilde{h}_a = \langle \mathcal{T} \rangle$, $\tilde{h}_1 := H(D[j-1])$, $\tilde{h}_{i+1} = H(h_i \,\|\, \tilde{h}_i)$ if h_i is the left child of its parent, and, $\tilde{h}_{i+1} = H(\tilde{h}_i \,\|\, h_i)$ if h_i is the right child of its parent.

Consider the event DISCREPANCY that $\tilde{h}^*_q \neq h^*_{\text{last}}$ and the event INVALID-PROOF that $\tilde{h}^*_q = h^*_{\text{last}} \wedge D[j-1] \neq D^*[j-1]$. Since $\Pr[D[j-1] \neq D^*[j-1] \mid \text{BAD}] \leq \Pr[\text{DISCREPANCY}] + \Pr[\text{INVALID-PROOF}]$ we next bound the probabilities of these events.

We first construct a hash collision adversary $\mathcal{A}_1$ that calls the responder as a subroutine, and show that the event DISCREPANCY implies that $\mathcal{A}_1$ succeeds. For $i \in \{q, \ldots, a\}$, define $h^*_{i,c}$ as: $h^*_{a,c} := \langle \mathcal{T} \rangle^*$ and $h^*_{i,c} := h^*_{i,l} \,\|\, h^*_{i,r}$ if $i < h$. Similarly, define $h^*_{i,m}$ as: $h^*_{a,m} := \langle \mathcal{T} \rangle^*$, $h^*_{i,m} := \tilde{h}^*_i \,\|\, h^*_i$ if h_i is the right child of its parent, and $h^*_{i,m} := h^*_i \,\|\, \tilde{h}^*_i$ if h_i is the left child of its parent.

The adversary $\mathcal{A}_1$ calls the responder as a sub-routine, and obtains the values $h^*_{i,c}$ and $h^*_{i,m}$, $i \in \{q, \ldots, h\}$. It finds the first index p for which there is a collision $H(h^*_{p,m}) = H(h^*_{p,c})$ and $h^*_{p,m} \neq h^*_{p,c}$ and returns $a := h^*_{p,m}$ and $b := h^*_{p,c}$, if such an index p exists. Otherwise, it returns FAILURE.

In the case of DISCREPANCY, $\tilde{h}^*_q \neq h^*_{\text{last}} = h^*_{q,l}$. Hence, it must be the case that $h^*_{q,m} \neq h^*_{q,c}$. However, since $h^*_{a,m} = \langle \mathcal{T} \rangle^* = h^*_{a,c}$, a collision must have been found for at least one index $i \in \{q, \ldots, a-1\}$. Consequently, Discrepancy implies that $\mathcal{A}_1$ succeeds.

We next construct a Merkle tree adversary $\mathcal{A}_2$ that calls the responder as a subroutine, and show that the event Invalid-Proof implies that $\mathcal{A}_2$ succeeds.

Let P denote the sequence of leaves in the challenger's dirty tree, *i.e.*, within D, that lie under the subtree with root $h_{q,l}$. Let π denote the sub-sequence $h^*_1, \ldots, h^*_{q-1}$ within π^*. The adversary $\mathcal{A}_2$ receives P from the responder, and constructs a well-formed dirty tree using P in time $O(\text{poly}(\ell))$. It then obtains the leaf $v := D^*[j-1]$ and the Merkle proof $\pi = (h^*_1, \ldots, h^*_{q-1})$ from the responder. Finally, it returns v, P, π and the index idx of the leaf $D[j-1]$ within the sequence P such that $P[\text{idx}] = D[j-1]$.

If Invalid-Proof, it must be the case that $\tilde{h}^*_q = h^*_{\text{last}} = h_{q,l}$ and $D[j-1] \neq D^*[j-1] = v$. Hence, π is a valid Merkle proof for v with respect to the root $h_{q,l}$ of the Merkle tree with leaves P. Moreover, $v \neq P[\text{idx}]$. Consequently, Invalid-Proof implies than $\mathcal{A}_2$ succeeds.

Finally, by the fact that H is a collision-resistant hash function and Lemma 1,

$$\begin{aligned}&\Pr[D[j-1] \neq D^*[j-1] \mid \text{Bad}] \leq \\ &\Pr[\text{Discrepancy}] + \Pr[\text{Invalid-Proof}] \leq \\ &\Pr[\mathcal{A}_1 \text{ succeeds}] + \Pr[\mathcal{A}_2 \text{ succeeds}] \leq \text{negl}(\lambda)\,.\end{aligned}$$

Hence, for any PPT adversarial responder, the probability that the responder wins and $(D^*[j-1] \neq D[j-1]) \vee (D^*[j] = D[j])$ is negligible in λ. □

The next lemma ensures that an honest challenger can win in the bisection game by leveraging sound consensus and execution oracles to resolve any disagreements at the leaf level.

Lemma 7 (Bisection Soundness). *Let H^s be a collision resistant hash function. Consider an execution that satisfies ledger safety and in which the consensus and execution oracles are sound. Then, for all PPT adversarial responders $\mathcal{A}$ claiming root $\langle \mathcal{T} \rangle^*$ and size ℓ, the honest challenger claiming $\langle \mathcal{T} \rangle \neq \langle \mathcal{T} \rangle^*$ and ℓ wins the bisection game against $\mathcal{A}$ with overwhelming probability in λ.*

Proof. Consider an adversarial PPT responder $\mathcal{P}^*$ playing against the honest challenger $\mathcal{P}$ at some round r. Since the challenger is honest, his queries are valid and he does not time out. For the responder to win the bisection game, it must satisfy all the conditions checked by Algorithm 4. Let $(\mathsf{tx}^*_{j-1}, \langle \mathsf{st} \rangle^*_{j-1})$ and $(\mathsf{tx}^*_j, \langle \mathsf{st} \rangle^*_j)$ denote the two entries revealed by $\mathcal{P}^*$ for the consecutive indices $j-1$ and j in the event that it wins.

Define Consensus-Oracle as the event that the responder wins and $(\mathsf{tx}_{j-1}, \langle \mathsf{st} \rangle_{j-1}) = (\mathsf{tx}^*_{j-1}, \langle \mathsf{st} \rangle^*_{j-1}) \wedge \mathsf{tx}^*_j \neq \mathsf{tx}_j$. We construct a consensus oracle adversary $\mathcal{A}_1$ that calls $\mathcal{P}^*$ as a subroutine and outputs $(\mathsf{tx}^*_{j-1}, \mathsf{tx}^*_j, r)$. By the well-formedness of the challenger's dirty ledger and ledger safety, it holds that $(\mathsf{tx}_{j-1}, \mathsf{tx}_j) \mid \mathbb{L}^{\mathcal{P}}_r \preceq \mathbb{L}^{\cup}_r \preceq \mathbb{L}^{\cup}_{r+\nu}$. Therefore, if Consensus-Oracle, it must be the case that $\mathsf{tx}^*_j \neq \mathsf{tx}_j$ does not immediately follow $\mathsf{tx}^*_{j-1} = \mathsf{tx}_{j-1}$ on $\mathbb{L}^{\cup}_{r+\nu}$

as every transaction on $\mathbb{L}^{\cup}_{r+\nu}$ is unique. However, as the responder wins, the consensus oracle must have outputted *true* on $(\mathsf{tx}^*_{j-1}, \mathsf{tx}^*_j, r)$ by condition (5). Hence, Consensus-Oracle implies that $\mathcal{A}_1$ succeeds.

Define Execution-Oracle as the event that the responder wins and $(\mathsf{tx}_{j-1}, \langle\mathsf{st}\rangle_{j-1}) = (\mathsf{tx}^*_{j-1}, \langle\mathsf{st}\rangle^*_{j-1}) \ \wedge\ \mathsf{tx}^*_j = \mathsf{tx}_j \ \wedge\ \langle\mathsf{st}\rangle^*_j \neq \langle\mathsf{st}\rangle_j$. By the well-formedness of the challenger's dirty tree, there exist a state st_{j-1} such that $\langle\mathsf{st}\rangle_{j-1} = \langle\mathsf{st}_{j-1}\rangle$, $\mathsf{st}_{j-1} = \delta^*(st_0, \mathbb{L}[{:}j-1])$, and, $\langle\mathsf{st}\rangle_j = \langle\delta(\mathsf{st}_{j-1}, \mathsf{tx}_j)\rangle$. Therefore, if Execution-Oracle, it holds that $\langle\mathsf{st}\rangle_{j-1} = \langle\mathsf{st}\rangle^*_{j-1}$, and $\langle\delta(\mathsf{st}_{j-1}, \mathsf{tx}_j)\rangle = \langle\mathsf{st}\rangle_j \neq \langle\mathsf{st}\rangle^*_j$. However, as the responder wins, the execution oracle must have outputted *true* on tx^*_j, $\langle\mathsf{st}\rangle^*_{j-1}$ and $\langle\mathsf{st}\rangle^*_j$ by condition (6). Thus, the responder must have given a proof π such that $\langle\delta\rangle\left(\langle\mathsf{st}\rangle^*_{j-1}, \mathsf{tx}^*_j, \pi\right) = \langle\mathsf{st}\rangle^*_j$. This implies $\langle\delta\rangle\left(\langle\mathsf{st}\rangle_{j-1}, \mathsf{tx}_j, \pi\right) = \langle\mathsf{st}\rangle^*_j \neq \langle\delta(\mathsf{st}_{j-1}, \mathsf{tx}_j)\rangle$.

Finally, we construct an execution oracle adversary $\mathcal{A}_2$ that calls $\mathcal{P}^*$ as a subroutine and receives π. Then, using $\mathbb{L}$, $\mathcal{A}_2$ finds $\mathsf{st}_{j-1} = \delta^*(st_0, \mathbb{L}[{:}j-1])$ in $O(\mathrm{poly}(\ell))$ time. It outputs $(\mathsf{st}_{j-1}, \mathsf{tx}_j, \pi)$. Observe that if Execution-Oracle, then $\mathcal{A}_2$ succeeds.

Note that the event $(\mathsf{tx}_{j-1}, \langle\mathsf{st}\rangle_{j-1}) = (\mathsf{tx}^*_{j-1}, \langle\mathsf{st}\rangle^*_{j-1}) \wedge (\mathsf{tx}_j, \langle\mathsf{st}\rangle_j) \neq (\mathsf{tx}^*_j, \langle\mathsf{st}\rangle^*_j) \wedge$ *Responder wins* is the union of the events Consensus-Oracle and Execution-Oracle:

$$\begin{aligned}
&\Pr[(\mathsf{tx}_{j-1}, \langle\mathsf{st}\rangle_{j-1}) = (\mathsf{tx}^*_{j-1}, \langle\mathsf{st}\rangle^*_{j-1}) \wedge \\
&\quad (\mathsf{tx}_j, \langle\mathsf{st}\rangle_j) \neq (\mathsf{tx}^*_j, \langle\mathsf{st}\rangle^*_j) \wedge \text{ Responder wins}] = \\
&\Pr[\textsc{Consensus-Oracle} \vee \textsc{Execution-Oracle}] \leq \\
&\quad \Pr[\mathcal{A}_1 \text{ succeeds}] + \Pr[\mathcal{A}_2 \text{ succeeds}] \leq \mathrm{negl}(\lambda)\,.
\end{aligned}$$

Moreover, by Lemma 6;

$$\begin{aligned}
&\Pr[((\mathsf{tx}_{j-1}, \langle\mathsf{st}\rangle_{j-1}) \neq (\mathsf{tx}^*_{j-1}, \langle\mathsf{st}\rangle^*_{j-1}) \vee \\
&\quad (\mathsf{tx}_j, \langle\mathsf{st}\rangle_j) = (\mathsf{tx}^*_j, \langle\mathsf{st}\rangle^*_j)) \ \wedge\ \text{Responder wins}] \leq \mathrm{negl}(\lambda)\,.
\end{aligned}$$

Consequenty, $\Pr[\text{Responder wins}] = \mathrm{negl}(\lambda)$. □

Theorem 5 (Succinctness). *Consider a consensus and execution oracle with f and g communication complexity respectively. Then, the challenge game invoked at round r with sizes ℓ_1 and $\ell_2 > \ell_1$ ends in $\log(\ell_1 + \alpha(u+\nu))$ rounds of communication and has a total communication complexity of $O(\log(\ell_1)) + \alpha(u+\nu)(f(r)+g(r)))$.*

Proof. Suppose the challenge game was invoked on augmented dirty ledgers with (alleged) sizes ℓ_1 and $\ell_2 > \ell_1$ respectively. The zooming phase of the challenge game does not require any communication among the provers and the verifier.

Suppose that at the end of the zooming phase, the provers play a bisection game on two Merkle trees with $\ell \leq \ell_1$ leaves. By Lemma 4, the bisection game ends in $\Theta(\log \ell) = \Theta(\log \ell_1)$ rounds and has a total communication complexity of $O(\log \ell + f(r) + g(r)) = O(\log \ell_1 + f(r) + g(r))$.

Suppose that the challenge game reaches the suffix monologue. Since the verifier checks for at most $\alpha(u+\nu)$ extra entries, $(\mathsf{tx}_j, \langle \mathsf{st} \rangle_j)$, $j \in \{\ell_1, \ldots, \min(\ell_2, \ell_1 + \alpha(u+\nu))\}$, at most $\alpha(u+\nu)$ entries are sent to the verifier by the challenger. These entries have constant sizes since the transactions and the state commitments are assumed to have constant sizes. Finally, the verifier can query the consensus oracle on the $\alpha(u+\nu)$ transaction pairs $(\mathsf{tx}_{j-1}, \mathsf{tx}_j)$, $j \in \{\ell_1 + 1, \min(\ell_2, \ell_1 + \alpha(u+\nu))\}$, and the execution oracle on the $\alpha(u+\nu)$ triplets $(\langle \mathsf{st} \rangle_{j-1}, \mathsf{tx}_j, \langle \mathsf{st} \rangle_j)$, $j \in \{\ell_1 + 1, \min(\ell_2, \ell_1 + \alpha(u+\nu))\}$, with $O(\alpha(u+\nu)f(r))$ and $O(\alpha(u+\nu)g(r))$ communication complexity respectively. Hence, the total communication complexity of the challenge game becomes $O(\log \ell_1 + \alpha(u+\nu)(f(r)+g(r)))$. □

By the Lipschitz property of the ledger, $|\mathbb{L}_r^{\cup}| < \alpha r$, and α, ν, u are constants. Superlight client constructions [35,12] place f in $\mathcal{O}(\text{poly} \log r)$, and g is in $\mathcal{O}(\text{poly} \log r)$ if standard Merkle constructions [4] are used and the transition function δ ensures the state grows at most linearly, as is the case in all practical constructions. In light of these quantities, the result of the above theorem establishes that our protocol is also $\mathcal{O}(\text{poly} \log r)$ and, hence, *succinct*.

Theorem 6 (Completeness). *Suppose the consensus and execution oracles are complete and the ledger is safe. Then, the honest responder wins the challenge game against any PPT adversarial challenger.*

Proof. Suppose that at the end of the zooming phase, the challenger invoked the bisection game between one of the honest responder's peaks, $\langle \mathcal{T} \rangle_i$, and a node $\langle \mathcal{T} \rangle^*$ alleged to have the same position as $\langle \mathcal{T} \rangle_i$ within the challenger's MMR. By Lemma 5, the honest responder wins the bisection game. If the challenger starts a suffix monologue instead of the bisection game at the end of the zooming phase, the responder automatically wins the challenge game. Hence, the responder wins the challenge game. □

Proposition 2. *For any honest prover $\mathcal{P}$ and round r, $|\mathbb{L}_r^{\cup}| < |\mathbb{L}_r^{\mathcal{P}}| + \alpha u$.*

Proof. Towards contradiction, suppose $|\mathbb{L}_r^{\cup}| \geq |\mathbb{L}_r^{\mathcal{P}}| + \alpha u$. By ledger safety, there exists an honest prover $\mathcal{P}'$ such that $\mathbb{L}_r^{\mathcal{P}'} = \mathbb{L}_r^{\cup}$, which implies $|\mathbb{L}_r^{\mathcal{P}'}| \geq |\mathbb{L}_r^{\mathcal{P}}| + \alpha u$. Again by ledger safety, $\mathbb{L}_r^{\mathcal{P}} \preceq \mathbb{L}_r^{\mathcal{P}'}$. By ledger liveness, every transaction that is in $\mathbb{L}_r^{\mathcal{P}'}$ and not in $\mathbb{L}_r^{\mathcal{P}}$ becomes part of $\mathbb{L}_{r+u}^{\mathcal{P}}$, for which $\mathbb{L}_r^{\mathcal{P}} \preceq \mathbb{L}_{r+u}^{\mathcal{P}}$ holds by ledger safety. Hence, $\mathbb{L}_r^{\mathcal{P}} \preceq \mathbb{L}_r^{\mathcal{P}'} \preceq \mathbb{L}_{r+u}^{\mathcal{P}}$ and, $|\mathbb{L}_{r+u}^{\mathcal{P}}| \geq |\mathbb{L}_r^{\mathcal{P}'}| \geq |\mathbb{L}_r^{\mathcal{P}}| + \alpha u$. However, this is a violation of the ledger Lipschitz property. Consequently, it should be the case that $|\mathbb{L}_r^{\cup}| < |\mathbb{L}_r^{\mathcal{P}}| + \alpha u$. □

Lemma 8 (Monologue Succinctness). *Consider an execution of a consensus protocol which is Lipschitz with parameter α and has liveness with parameter u. Consider the challenge game instantiated with a collision-resistant hash function H^s and a consensus oracle which is sound with parameter ν. For all PPT adversarial challengers $\mathcal{A}$, if the game administered by the honest verifier among $\mathcal{A}$ and the honest responder $\mathcal{P}$ at round r reaches the suffix monologue, the adversary cannot reveal $\alpha(u+\nu)$ or more entries and win the game except with negligible probability.*

Proof. Suppose the game between the challenger $\mathcal{A}$ and the honest responder $\mathcal{P}$ reaches the suffix monologue. Consider the event BAD that the challenger reveals $\beta \geq \alpha(u+\nu)$ entries and wins the game. Let $D = ((\mathsf{tx}_1, \langle \mathsf{st} \rangle_1), (\mathsf{tx}_2, \langle \mathsf{st} \rangle_2), \ldots, (\mathsf{tx}_\beta, \langle \mathsf{st} \rangle_\beta))$ denote these entries, and $(\mathsf{tx}_0, \langle \mathsf{st} \rangle_0)$ the responder's last entry prior to the monologue phase. Because $\mathcal{P}$ is in agreement with tx_0, therefore $\mathsf{tx}_0 = \mathbb{L}_r^{\mathcal{P}}[-1]$. Let $J = (\mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_\beta)$. Since the challenger wins, the verifier has invoked the consensus oracle $\alpha(u+\nu)$ times for all consecutive pairs of transactions within $K = J[:\alpha(u+\nu)]$. At each invocation, the consensus oracle has returned *true*.

We next construct a consensus oracle adversary $\mathcal{A}'$ that calls $\mathcal{A}$ as a subroutine. If $\beta \geq \alpha(u+\nu)$, $\mathcal{A}'$ identifies the first index $p \in [\alpha(u+\nu)]$ such that tx_p does not immediately follow tx_{p-1} on $\mathbb{L}_{r+\nu}^{\cup}$, and outputs $(\mathsf{tx}_{p-1}, \mathsf{tx}_p, r)$. If $\beta < \alpha(u+\nu)$, $\mathcal{A}'$ outputs FAILURE.

By the ledger Lipschitz property, $|\mathbb{L}_{r+\nu}^{\mathcal{P}}| < |\mathbb{L}_r^{\mathcal{P}}| + \alpha\nu$. Moreover, by Lemma 2, $|\mathbb{L}_{r+\nu}^{\cup}| < |\mathbb{L}_{r+\nu}^{\mathcal{P}}| + \alpha u$. Thus, $|\mathbb{L}_{r+\nu}^{\cup}| < |\mathbb{L}_r^{\mathcal{P}}| + \alpha(u+\nu)$.

Let $\ell = |\mathbb{L}_r^{\mathcal{P}}|$. By ledger safety, $\mathsf{tx}_0 = \mathbb{L}_r^{\mathcal{P}}[\ell-1] = \mathbb{L}_{r+\nu}^{\cup}[\ell-1]$. Hence, if $(\mathsf{tx}_0, \mathsf{tx}_1) \mid \mathbb{L}_{r+\nu}^{\cup}$, $\mathsf{tx}_1 = \mathbb{L}_{r+\nu}^{\cup}[\ell]$ as every transaction on $\mathbb{L}_{r+\nu}^{\cup}$ is unique. By induction, either there exists an index $i \in [\beta]$ such that tx_i does not immediately follow tx_{i-1} on $\mathbb{L}_{r+\nu}^{\cup}$, or $|\mathbb{L}_{r+\nu}^{\cup}| \geq \ell + \beta$ and $\mathsf{tx}_i = \mathbb{L}_{r+\nu}^{\cup}[\ell+i-1]$ for all $i \in [\beta]$.

Finally, if $\beta \geq \alpha(u+\nu)$, there exists an index $i \in [\alpha(u+\nu)]$ such that tx_i does not immediately follow tx_{i-1} on $\mathbb{L}_{r+\nu}^{\cup}$. Thus, $\Pr[\text{BAD}] = \Pr[\mathcal{A}'\text{succeeds}]$. However, by the soundness of the consensus oracle, $\forall$ PPT $\mathcal{A}'$, $\Pr[\mathcal{A}'\text{succeeds}] = \text{negl}(\lambda)$. Therefore, $\Pr[\text{BAD}] = \text{negl}(\lambda)$. □

Theorem 7 (Soundness). *Let H^s be a (keyed) collision resistant hash function. Suppose the consensus and execution oracles are complete and sound. Then, for all PPT adversarial responders $\mathcal{A}$, an honest challenger wins the challenge game against $\mathcal{A}$ with overwhelming probability in λ.*

Proof. Suppose that at the end of the zooming phase, the honest challenger $\mathcal{P}$ identified one of the responder $\mathcal{P}^*$'s peaks, $\langle \mathcal{T} \rangle_i^*$, as being different from a node $\langle \mathcal{T} \rangle$ within the challenger's MMR that has the same position as $\langle \mathcal{T} \rangle_i^*$. In this case, the challenger initiates a bisection game between $\langle \mathcal{T} \rangle_i^*$ and $\langle \mathcal{T} \rangle$. By Lemma 7, the honest challenger wins the bisection game with overwhelming probability.

Suppose the challenger observes that the peaks shared by the responder correspond to the peaks of a well-formed MMR. Then, at the end of the zooming phase, the honest challenger starts the suffix monologue. Let ℓ and ℓ^* denote the challenger's and the responder's (alleged) augmented dirty ledger sizes respectively. Let r denote the round at which the challenge game was started. During the suffix monologue, the challenger reveals its augmented dirty ledger entries $(\mathsf{tx}_j, \langle \mathsf{st} \rangle_j)$ at the indices $\ell^*, \ldots, \min(\ell, \ell^* + \alpha(u+\nu)) - 1$. Then, for all $j \in \{\ell^*+1, \ell^*+2, \ldots, \min(\ell, \ell^* + \alpha(u+\nu)) - 1\}$, the verifier checks the transactions and the state transitions between $(\mathsf{tx}_{j-1}, \langle \mathsf{st} \rangle_{j-1})$ and $(\mathsf{tx}_j, \langle \mathsf{st} \rangle_j)$. The verifier does the same check between the responder's last (alleged) augmented dirty ledger entry $(\mathsf{tx}^*_{\ell^*-1}, \langle \mathsf{st} \rangle^*_{\ell^*-1})$ and $(\mathsf{tx}_{\ell^*}, \langle \mathsf{st} \rangle_{\ell^*})$.

Consider the event EQUAL that $(\mathsf{tx}^*_{\ell^*-1}, \langle \mathsf{st} \rangle^*_{\ell^*-1}) = (\mathsf{tx}_{\ell^*-1}, \langle \mathsf{st} \rangle_{\ell^*-1})$. By the well-formedness of the challenger's augmented dirty ledger, for any pair of leaves at indices $j-1$ and j, $j \in \{\ell^*, \ldots, \min(\ell, \ell^* + \alpha(u+\nu)) - 1\}$, it holds that $(\mathsf{tx}_{j-1}, \mathsf{tx}_j) \mid \mathbb{L}_r^{\mathcal{P}}$, thus, on $\mathbb{L}_r^{\cup}$ by ledger safety. As the consensus oracle is complete, by Definition 6, it returns *true* on all $(\mathsf{tx}_{j-1}, \mathsf{tx}_j)$ for $j \in \{\ell^*, \ldots, \min(\ell, \ell^* \alpha(u+\nu)) - 1\}$. Similarly, by the well-formedness of the challenger's augmented dirty ledger, for any pair of leaves at indices $j-1$ and j, $j \in \{\ell^*, \ldots, \min(\ell, \ell^* + \alpha(u+\nu)) - 1\}$, there exists a state st_{j-1} such that $\langle \mathsf{st} \rangle_{j-1} = \langle \mathsf{st}_{j-1} \rangle$, and, $\langle \mathsf{st} \rangle_j = \langle \delta(\mathsf{st}_{j-1}, \mathsf{tx}_j) \rangle$. As the execution oracle is complete, by Definition 8, for all $j \in \{\ell^*, \ldots, \min(\ell, \ell^* + \alpha(u+\nu)) - 1\}$, $M(\mathsf{st}_{j-1}, \mathsf{tx}_j)$ outputs a proof π_j such that $\langle \delta \rangle (\langle \mathsf{st}_{j-1} \rangle, \mathsf{tx}_j, \pi_j) = \langle \delta(\mathsf{st}_{j-1}, \mathsf{tx}_j) \rangle$. Thus, for all $j \in \{\ell^*, \ldots, \min(\ell, \ell^* + \alpha(u+\nu)) - 1\}$, the verifier obtains a proof π_j such that $\langle \delta \rangle (\langle \mathsf{st} \rangle_{j-1}, \mathsf{tx}_j, \pi_j) = \langle \mathsf{st} \rangle_j$. In other words, if the challenge protocol reaches the suffix monologue and EQUAL, the honest challenger wins the suffix monologue.

Finally, consider the Merkle tree adversary $\mathcal{A}'$ that calls the responder $\mathcal{P}^*$ as a subroutine. Let π^* denote the Merkle proof revealed by the responder for $(\mathsf{tx}^*_{\ell^*-1}, \langle \mathsf{st} \rangle^*_{\ell^*-1})$ with respect to its last (alleged) peak $\langle \mathcal{T} \rangle^*$. Let $\langle \mathcal{T} \rangle$ denote the corresponding node in the challenger's MMR. Let D denote the sequence of augmented dirty ledger entries held by the honest challenger in the subtree rooted at $\langle \mathcal{T} \rangle$. Let $\text{idx} := |D|$ denote the size of this subtree. If the game reaches the suffix monologue and $\neg$EQUAL, $\mathcal{A}'$ returns $v := (\mathsf{tx}^*_{\ell^*-1}, \langle \mathsf{st} \rangle^*_{\ell^*-1})$, D, π and idx. Otherwise, it returns FAILURE.

If the game reaches the suffix monologue, $\langle \mathcal{T} \rangle^* = \langle \mathcal{T} \rangle$, and π is valid with respect to $\langle \mathcal{T} \rangle$. Then, if $(\mathsf{tx}^*_{\ell^*-1}, \langle \mathsf{st} \rangle^*_{\ell^*-1}) \neq (\mathsf{tx}_{\ell^*-1}, \langle \mathsf{st} \rangle_{\ell^*-1})$, therefore $\langle \mathcal{T} \rangle = \text{MAKEMT}(D).\text{root}$, $D[\text{idx}] \neq v$, and $\text{VERIFY}(\pi, \langle \mathcal{T} \rangle, \text{idx}, v) = 1$. Conditioned on the fact that the challenge game reaches the suffix monologue, by Proposition 1, $\Pr[\neg\text{EQUAL}] = \Pr[\mathcal{A}' \text{ succeeds}] = \text{negl}(\lambda)$. Thus, the honest challenger wins the challenge protocol with overwhelming probability. □

Theorem 8 (Tournament Runtime). *Suppose the consensus and execution oracles are complete and sound, and have f and g communication complexity respectively. Consider a tournament started at round r with n provers. Given at least one honest prover, for any PPT adversary $\mathcal{A}$, the tournament ends in $2n \log(|\mathbb{L}_r^{\cup}| + \alpha(u+\nu))$ rounds of communication and has a total communication complexity of $O(2n \log(|\mathbb{L}_r^{\cup}| + \alpha(u+\nu)) + 2n\alpha(u+\nu)(f(r) + g(r)))$, with overwhelming probability in λ.*

Proof. By the end of the first step, size of the set $\mathcal{S}$ can be at most 2. Afterwards, each step of the tournament adds at most one prover to $\mathcal{S}$ and the number of steps is $n-1$. Moreover, at each step, either there is exactly one challenge game played, or if $k > 1$ games are played, at least $k-1$ provers are removed from $\mathcal{S}$. Hence, the maximum number of challenge games that can be played over the tournament is at most $2n-1$.

Recall that the size alleged by $\mathcal{P}_i$ is at most the size alleged by $\mathcal{P}_{i+1}$, $i \in [n-1]$. Let i^* be the first round where an honest prover plays the challenge game. If $i^* > 1$, until round i^*, the sizes alleged by the provers are upper bounded by $|\mathbb{L}_r^{\cup}|$. From round i^* onward, at each round, the prover $\overline{\mathcal{P}}$ claiming the largest size is either honest or must have at least once won the challenge game as a *challenger* against an honest responder. During the game against the honest responder, by Lemma 8, $\overline{\mathcal{P}}$ could not have revealed $\alpha(u+\nu)$ or more entries except with negligible probability. Hence, from round i^* onward, with overwhelming probability, the size claimed by $\overline{\mathcal{P}}$ at any round can at most be $|\mathbb{L}_r^{\cup}|+\alpha(u+\nu)-1$. Thus, with overwhelming probability, by Theorem 5, each challenge game ends after at most $\log(|\mathbb{L}_r^{\cup}|+\alpha(u+\nu))$ rounds of interactivity and has total communication complexity $O(\log(|\mathbb{L}_r^{\cup}|+\alpha(u+\nu))+\alpha(u+\nu)(f(r)+g(r)))$. Consequently, with overwhelming probability, the tournament started at round r with n provers ends in at most $2n\log(|\mathbb{L}_r^{\cup}|+\alpha(u+\nu))$ rounds of interactivity and has total communication complexity $O(2n\log(|\mathbb{L}_r^{\cup}|+\alpha(u+\nu))+2n\alpha(u+\nu)(f(r)+g(r)))$. □

Lemma 9. *Consider a challenge game invoked by the verifier at some round r. If at least one of the provers $\mathcal{P}$ is honest, for all PPT adversarial $\mathcal{A}$, the state commitment obtained by the verifier at the end of the game between $\mathcal{P}$ and $\mathcal{A}$ satisfies state security with overwhelming probability.*

Proof. If the challenger is honest, by Theorem 7, he wins the challenge game with overwhelming probability and the verifier accepts his state commitment.

Suppose the responder $\mathcal{P}$ of the challenge game is honest, and it is challenged by a challenger $\mathcal{P}^*$. If $\mathcal{P}^*$ starts a bisection game, by Lemma 5, $\mathcal{P}^*$ loses the challenge game and $\mathcal{P}$ wins the game. In this case, the verifier accepts the state commitment given by the honest responder. On the other hand, if the challenge game reaches the suffix monologue and the challenger loses the monologue, the verifier again accepts the state commitment given by the honest responder. As the state commitment of an honest prover satisfies security as given by Definition 2, in all of the cases above, the commitment accepted by the verifier satisfies state security with overwhelming probability.

Finally, consider the event Win that the game reaches the suffix monologue and the challenger wins. Let ℓ and ℓ^* denote the responder's and the challenger's (alleged) augmented dirty ledger lengths respectively. During the suffix monologue, the challenger reveals its alleged entries $(\mathsf{tx}_i^*, \langle\mathsf{st}\rangle_i^*)$ at indices $i=\ell+1,\dots,\min(\ell^*,\ell+\alpha(u+\nu))-1$. Let $(\mathsf{tx}_{\ell-1}, \langle\mathsf{st}\rangle_{\ell-1})$ denote the responder's last entry. As the challenger wins, consensus oracle must have returned true on $(\mathsf{tx}_{\ell-1}, \mathsf{tx}_\ell^*)$ and $(\mathsf{tx}_{i-1}^*, \mathsf{tx}_i^*)$ for all $i \in \{\ell+1,\dots,\min(\ell^*,\ell+\alpha(u+\nu))-1\}$. Similarly, for all $i \in \{\ell,\dots,\min(\ell^*,\ell+\alpha(u+\nu))\}$, execution oracle must have outputted a proof $\pi_{i-\ell+1}$ such that $\langle\delta\rangle(\langle\mathsf{st}\rangle_{\ell-1}, \mathsf{tx}_\ell^*, \pi_1) = \langle\mathsf{st}\rangle_\ell^*$ and it holds $\langle\delta\rangle(\langle\mathsf{st}\rangle_{i-1}^*, \mathsf{tx}_i^*, \pi_{i-\ell+1}) = \langle\mathsf{st}\rangle_i^*$ for $i \in \{\ell+1,\dots,\min(\ell^*,\ell+\alpha(u+\nu))-1\}$.

Let D denote the sequence $\mathsf{tx}_{\ell-1}, \mathsf{tx}_\ell^*, \dots, \mathsf{tx}^*_{\min(\ell^*,\ell+\alpha(u+\nu))-1}$ of transactions. Consider the event Consensus-Oracle that Win holds, $\ell^* < \ell+\alpha(u+\nu)$, and there exists an index $i \in \{1,\dots,\ell^*-\ell\}$ such that $D[i]$ does not immediately follow $D[i-1]$ on $\mathbb{L}_{r+\nu}^{\cup}$. We next construct a consensus oracle adversary $\mathcal{A}_1$ that calls $\mathcal{P}^*$ as a subroutine. The adversary $\mathcal{A}_1$ identifies the first index $p>0$ such that $D[p]$ does not immediately follow $D[p-1]$ on $\mathbb{L}_{r+\nu}^{\cup}$ if such an index exists, and outputs $(D[p-1], D[p], r)$. Otherwise, $\mathcal{A}_1$ outputs Failure. Hence, Consensus-Oracle implies that $\mathcal{A}_1$ succeeds.

Let S denote the sequence $\langle \mathsf{st} \rangle_{\ell-1}, \langle \mathsf{st} \rangle^*_{\ell}, \ldots, \langle \mathsf{st} \rangle^*_{\min(\ell^*, \ell+\alpha(u+\nu))-1}$. Define $\mathsf{st}_i = \delta^*(st_0, \mathbb{L}^{\mathcal{P}}_r || (\mathsf{tx}^*_{\ell}, \ldots, \mathsf{tx}^*_{\ell+i-1}))$ for $i \in \{1, \ldots, \ell^* - \ell\}$ ($\mathsf{st}_0 = \delta^*(st_0, \mathbb{L}^{\mathcal{P}}_r)$). Consider the event EXECUTION-ORACLE that WIN holds, $\ell^* < \ell + \alpha(u+\nu)$, $\neg$CONSENSUS-ORACLE holds, and $S[i] \neq \langle \mathsf{st}_i \rangle$ for at least one index $i \in \{1, \ldots, \ell^* - \ell\}$. We next construct an execution oracle adversary $\mathcal{A}_2$ that calls $\mathcal{P}^*$ as a subroutine. Using $\mathbb{L}^{\mathcal{P}}_r$, $\mathcal{A}_2$ finds st_i for all $i \in \{0, 1, \ldots, \ell^* - \ell\}$ in $O(\text{poly}(|\mathbb{L}^{\mathcal{P}}_r|)$ time. Then, $\mathcal{A}_2$ identifies the first index $p > 0$ such that $S[p] \neq \langle \mathsf{st}_p \rangle$ if such an index exists, and outputs $\mathsf{st} = \mathsf{st}_{p-1}$, $\mathsf{tx} = D[p] = \mathsf{tx}^*_{\ell+p-1}$, and $\pi = \pi_p$. Otherwise, $\mathcal{A}_2$ outputs FAILURE. Since $\langle \delta \rangle (S[i-1], D[i], \pi_i) = S[i]$ for $i \in \{0, 1, \ldots, \min(\ell^* - \ell, \alpha(u+\nu)) - 1\}$, the EXECUTION-ORACLE implies that

$$\langle \delta(\mathsf{st}_{p-1}, D[p]) \rangle = \langle \mathsf{st}_p \rangle \neq S[p] = \langle \delta \rangle (S[p-1], D[p], \pi_p) = \langle \delta \rangle (\langle \mathsf{st}_{p-1} \rangle, D[p], \pi_p),$$

i.e., $\mathcal{A}_2$ succeeds.

Finally, if WIN $\wedge$ $\neg$CONSENSUS-ORACLE $\wedge$ $\neg$EXECUTION-ORACLE $\wedge$ $\ell^* < \ell + \alpha(u+\nu)$, the verifier accepts the commitment $\langle \mathsf{st} \rangle^*_{\ell^*-1}$, which satisfies state security by Definition 2 (Here, $\mathbb{L} = \mathbb{L}^{\mathcal{P}}_r \parallel (\mathsf{tx}^*_{\ell}, \ldots, \mathsf{tx}^*_{\ell^*-1}) \preceq \mathbb{L}^{\cup}_{r+\nu}$ and $\langle \mathsf{st} \rangle^*_{\ell^*-1} = \langle \delta^*(\mathsf{st}_0, \mathbb{L}) \rangle = \langle \mathsf{st} \rangle$). However,

$$\Pr[\text{CONSENSUS-ORACLE} \vee \text{EXECUTION-ORACLE}] \leq \Pr[\mathcal{A}_1 \text{ succeeds}] + \Pr[\mathcal{A}_2 \text{ succeeds}] \leq \text{negl}(\lambda).$$

Moreover, by Lemma 8, $\mathcal{P}^*$ cannot reveal $\alpha(u+\nu)$ or more entries and win the game except with negligible probability. Hence,

$$\Pr[\text{WIN}] = \text{negl}(\lambda) + \Pr[\text{WIN} \wedge \neg\text{CONSENSUS-ORACLE} \wedge \neg\text{EXECUTION-ORACLE} \wedge \ell^* < \ell + \alpha(u+\nu)].$$

which implies that either $\Pr[\text{WIN}] = \text{negl}(\lambda)$ or conditioned on WIN, the commitment accepted by the verifier satisfies state security except with negligible probability. Consequently, in a challenge game invoked by the verifier at some round r, if at least one of the provers is honest, the state commitment obtained by the verifier satisfies state security except with negligible probability. ☐

Theorem 9 (Security). *Suppose the consensus and execution oracles are complete and sound, and have f and g communication complexity respectively. Consider a tournament started at round r with n provers. Given at least one honest prover, for any PPT adversary $\mathcal{A}$, the state commitment obtained by the prover at the end of the tournament satisfies State Security with overwhelming probability in λ.*

Proof. Let $\mathcal{P}_{i^*}$ denote an honest prover within $\mathcal{P}$. Let $n = |\mathcal{P}| - 1$ denote the total number of rounds. By Theorems 6 and 7, $\mathcal{P}_{i^*}$ wins every challenge game and stays in $\mathcal{S}$ after step i^* with overwhelming probability.

The prover $\overline{\mathcal{P}}$ with the largest alleged MMR at the end of each step $i \geq i^*$ is either $\mathcal{P}_{i^*}$ or has a larger (alleged) MMR than the one held by $\mathcal{P}_{i^*}$. In the first case, as $\mathcal{P}_{i^*}$ is honest, its state commitment satisfies safety and liveness per

Definition 2. In the latter case, $\overline{\mathcal{P}}$ must have played the challenge game with $\mathcal{P}_{i^*}$. Then, by Lemma 9, the state commitment of $\overline{\mathcal{P}}$ satisfies safety and liveness per Definition 2 with overwhelming probability. Consequently, the state commitment obtained by the verifier at the end of the tournament, *i.e.*, the commitment of $\overline{\mathcal{P}}$ at the end of round $n \geq i^*$, satisfies safety and liveness with overwhelming probability. □

Theorem 10 (Prover Complexity). *When updating the MMR on a rolling basis, provers do constant amortized number of hash computations per transaction. Moreover, a node with an MMR of ℓ leaves can append a new leaf to its MMR with at most $O(\log \ell)$ hash computations.*

Proof. Given an augmented dirty ledger of length ℓ, a prover can construct the corresponding MMR with $O(\ell)$ operations upon entering the challenge game. This is because each prover can obtain the binary representation of ℓ with $O(\ell)$ operations, and create each of the k Merkle trees $\mathcal{T}_i$, $i \in [k]$, with $O(2^{q_i})$ hash computations, making the total compute complexity $O(\ell)$ (*cf.* Section 3.1). Hence, updating the MMR on a rolling basis, each prover can obtain an MMR with ℓ leaves with $O(1)$ amortized number of operations per transaction.

Finally, a node with an MMR of ℓ leaves can append a new leaf to its MMR with $O(\log \ell)$ hash computations; since in the worst case, it only needs to combine the existing $\log \ell$ hashes to update the MMR. Hence, per each new transaction, each prover only incurs at most logarithmic compute complexity. □