Light Clients for Lazy Blockchains

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

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

Abstract. Decoupling consensus from transaction verification and execution is an important technique to increase the throughput of blockchains, a technique known as a *lazy* blockchain. Lazy blockchains can end up containing invalid transactions such as double spends, but these can easily be filtered out by full nodes that can check if there have been previous conflicting transactions. However, creating light (SPV) clients that do not see the whole transaction history on top of these chains 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 only 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 we illustrate how it can be applied to both the UTXO as well as the account based models. Lastly, we empirically demonstrate the feasibility of our scheme by providing experimental results.

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 be the winner of 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). 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 *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 transactions one by one, as long as they are valid, and ignoring transactions that are not applicable. For instance, if a transaction is a double spend of another and they both appear on the dirty ledger, only the first one to appear on the ledger will be executed. Examples of lazy distributed ledger protocols include Celestia (LazyLedger) [1], Prism [3], Parallel Chains [22] and Snap-and-Chat [39].

The advantage of a lazy protocol is that consensus nodes do not need to perform any execution. This removes the execution bottleneck to scaling the blockchain [1]. However, because consensus nodes do not execute, they have no knowledge of the state of the system (such as account balances) and can therefore not prevent invalid transactions from appearing in the ledger. They also cannot find and include the state commitment in the blockchain.

For this reason, lazy protocols cannot easily provide support for light clients, and techniques from the realm of eager systems, such as SPV (Simple Payment Verification [38,11]), are not applicable. For instance, in Bitcoin, to help the light clients verify a transaction, a full node presents a block header containing the Merkle root of the transactions in the block, together with an inclusion proof of the transaction. However, in a lazy protocol, this is not sufficient since some transactions included in a block can be invalid. Similarly, in Ethereum, to prove to the light clients their current account balance, a full node presents a block header, together with a Merkle inclusion proof of the account to be verified within the state commitment. However, in a lazy protocol, there cannot be any such state commitment in a block.

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 them to synchronize with the network and quickly learn their latest account balance. Towards this purpose, the light client first connects to several full nodes, at least one of which is honest. 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.

Contributions. Our contributions in this paper 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.
2. We prove our system is *complete*, *sound*, and *succinct*, from 9 straightforward axioms that the underlying system must be governed under.
3. We illustrate that our full node prover is efficient in *compute* and *storage*.
4. We implement our scheme and compare the performance of our implementation against existing systems.

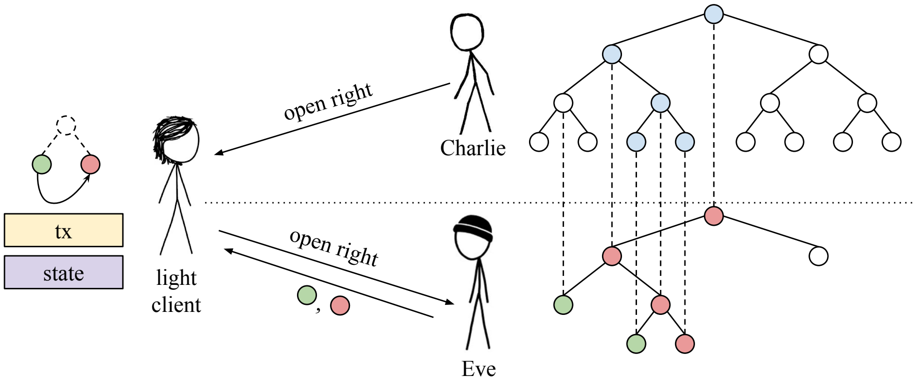

Fig. 1: Bisection Game. Charlie the challenger helps the light client iteratively traverse the tree of Eve the evil responder. A green node indicates a match, while a red node indicates a mismatch, between the two dirty trees.

Construction overview. Consider a light client connected to two full nodes, Charlie and Eve. Charlie is honest and Eve is adversarial. The light client begins by downloading the *canonical* header chain, each header containing the Merkle root of the transactions (valid and invalid) in its block. It wonders what its current account balance is. It asks this question to both full nodes. If both of them return the same answer, it rests assured that the balance reported is accurate. If they disagree, it must figure out who is truthful. Since Charlie and Eve are full nodes, they each have a copy of the current dirty ledger, and can already determine the correct balance.

To help convince the light client, Charlie takes the dirty ledger of transactions and augments it with some extra information: Together with every transaction, he includes a commitment to the state of the world 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*. Each leaf of this tree contains a transaction and its respective state commitment. All honest full nodes following this process will construct the same dirty tree and will hold the same dirty tree root. If the light client somehow learns the *correct* dirty tree root, then it can deduce its balance. To do so, it downloads a Merkle inclusion proof of the last leaf within this dirty tree. Within the last leaf lies the latest commitment to the state of the world, which, in turn, is another Merkle tree root. The light client downloads another Merkle tree proof that attests to its balance within this latest state commitment.

Charlie gives the correct dirty tree root to the light client, but Eve gives it an incorrect dirty tree root. It suffices for the light client to discover which of the two roots is truthful. Since the two roots are different, the underlying augmented dirty ledgers must differ somewhere. Charlie helps the light client identify the first leaf in Eve's dirty tree that differs from his own. With reference to his own dirty tree, he guides the light 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 selects to query 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 entry of disagreement is identified, the light client only needs to verify that it is fraudulent, as claimed by Charlie: It either contains an invalid state commitment or an invalid transaction. The light client can locally evaluate the correct state commitment for this item on its own by applying the transaction to the previous state commitment (which is valid as Charlie agrees with it). Therefore, any discrepancy in the *state commitment* will be caught by the light client. Lastly, in case of a discrepancy in the *transaction*, the light client can detect this by asking for the transaction's Merkle inclusion proof with respect to the header chain it already holds. Thus, the light client can identify the correct dirty tree root, and therefore deduce its correct balance. Here, headers are used to rule out transactions which are not in the dirty ledger at adversarially claimed positions.

Structure of the paper. We begin our paper by describing a construction that works in the setting of *blockchain* protocols, as this is easier to illustrate and the reader can refer to a concrete protocol. In our first presentation, we assume that the ledgers presented by one honest and one adversarial party have exactly the same lengths, and we discuss how any discrepancies can be resolved using a *Bisection Game* (Section 3.1). We then generalize our construction to allow for disagreements in length, which requires more complex machinery and uses a *Challenge Game* (Section 3.2) that makes use of the earlier bisection game. Next, we include more parties into the picture (Section 3.3) by introducing a *Tournament* that administers multiple instanes of our previous games. We provide some experimental results such as execution times from our prototype implementation that show our protocol is practical in Section 4. In particular, we observe that the light clients can determine the correct state on the order of seconds that matches the state-of-the-art for eager blockchains. We then generalize our construction to non-blockchain protocols by introducing the concept of a *consensus* and *execution oracle* in Section 5. This allows our construction to leverage an *axiomatic* approach in which the underlying system only needs to satisfy 9 axioms. While we provide intuition about the security of our construction early on, we leave the theoretical analysis for the end: Our security and succinctness theorems treating the axiomatic construction are stated in Section 6 and formally proven in Appendix A.

Related work. Lazy distributed ledger systems stand at the heart of any Layer-1 scalability solution. Popular such systems include Celestia (LazyLedger) [1], Prism [3], Parallel Chains [22] and Snap-and-Chat [39,40]. In the Appendix, we discuss how Celestia, Prism and Snap-and-Chat can be instantiated. *Eager* light clients were first introduced by Nakamoto [38]. *Superlight* clients for eager proof-of-work blockchains were put forth as NIPoPoWs [32,10,29,34] and CODA [8]. Improved light clients for eager proof-of-stake chains were described by Gaži et al. [25]. Our techniques are orthogonal to theirs and can be composed as discussed in Section 7. Our interactive bisection game is based on the work of Canetti et al. [13] which was first applied in the blockchain setting by Arbitrum [28]. Computation over large public logs was also explored by VerSum [27]. Contrary to Canetti and Arbitrum, where bisection games are used to dispute *com-*

putation over known data with a predetermined duration and result, our bisection games are administered over *ledgers*. Their data, due to nodes being out of synchrony for a bound duration, can be somewhat inconsistent. This challenge requires us to introduce novel techniques in these refereed games. The fraud proofs we use for our challenges are modelled after the work of Al-Bassam et al. [2]. As an alternative to our construction, recursive compositions [7] of SNARKs [6] and STARKs [5] can be used to support non-interactive lazy light clients. For example, Coda [8] relies on recursive SNARKs to enable verification of *all* past state transition in constant time, independent of the number of past transitions. Thus, on Coda, a light client can be convinced that a given state commitment is correct in constant time through a succinct proof.

2 Preliminaries & Model

Notation. For a natural number n, we use the bracket notation $[n]$ to mean the set $\{1, \cdots, n\}$. We use ϵ to denote the *empty string*. Given two strings a and b, we write $a \,\|\, b$ for some unambiguous encoding of their *concatenation*. Given a sequence X, we address it using $X[i]$ to mean the i^{th} element (starting from 0). We use negative indices to address elements from the end, so $X[-i]$ is the i^{th} element from the end, and $X[-1]$ in particular 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 not including) 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. Reversely, given an element x and a sequence xs, we use $(x{:}xs)$ to denote the sequence where the first element is x and the rest are 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 $X \prec Y$ to mean that the sequence X is a strict prefix of Y. Similarily, we use $X \preceq Y$ to mean that X is a prefix of Y. We use $X \mid Y$ to mean that X is a *subarray* of Y, meaning all the elements of X appear in Y consecutively.

Parties. Our protocol augments an existing permissionless lazy blockchain protocol of multiple participants. There are three types of nodes: *Consensus nodes*, *full nodes*, and *light clients*.

Consensus nodes. The *consensus nodes* receive and broadcast chains consisting of blocks. Each node locally maintains a chain $\mathcal{C}$ which always begins at the *genesis block* G. They validate the consensus rules of these chains [38], such as proof-of-work or proof-of-stake validity. These chains contain *constant* size transactions in some order. Each node reads its chain and produces a sequence of transactions called a *ledger*. This provides a total order of transactions. A consensus node also receives new, yet unconfirmed, transactions from the network to be introduced into the blockchain. To do this, it extends the blockchain by proposing new blocks that contain these transactions.

The consensus node is *lazy*: It treats transactions as meaningless strings, without validating them. It includes in its proposed block *any* transaction it receives with some spam-resilience mechanism.

At any point in time, the views of different consensus nodes can be different. *Honest* consensus nodes follow the specified consensus protocol. On the contrary, *adversarial* consensus nodes can arbitrarily deviate from it. The consensus protocol imposes some trust assumptions: For example, in proof-of-work [21], it is assumed that the majority of consensus nodes by *compute* are honest, and in proof-of-stake [33,14], it is assumed that the majority of the system's *money* is in honest hands. These assumptions are necessary so that the local views of the ledgers of different parties satisfy two properties: (1) *Safety* mandates that any transaction that appears in the ledger of one honest party will also eventually appear in the ledger of every honest party at the same position; (2) *Liveness* mandates that, if an honest party broadcasts a new transaction, it will eventually appear in the ledger of an honest party.

Full nodes. The *full nodes* maintain chains and ledgers just like the consensus nodes, but they do not produce new blocks. Instead, they rely on the fact that consensus nodes maintain the blockchain. Like consensus nodes, each full node also produces a ledger of transactions. Contrary to consensus nodes, full nodes execute transactions to maintain a *state*.

The current *state* of the system is 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 application of transactions on existing state is captured by a transition function δ. Consider a dirty ledger $\mathbb{L} = \mathsf{tx}_1 \cdots \mathsf{tx}_n$. Then the state of the system is $\delta(\cdots \delta(st_0, \mathsf{tx}_1), \cdots, \mathsf{tx}_n)$. 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. Namely, $\delta^*(st_0, \overline{\mathsf{tx}}) = \delta(\cdots \delta(st_0, \mathsf{tx}_1), \cdots, \mathsf{tx}_n)$.

Some transactions can be applied on top of a particular state, but others may not, because they are *invalid* with respect to the state (for example, they could be double spends). Because we are dealing with *lazy* systems, these invalid transactions are still contained in the ledger, hence the ledger is called *dirty*. We denote $\mathbb{L}_r^v$ 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 parties 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 transaction tx cannot be applied to state st because tx is invalid with respect to st, we let $\delta(\mathsf{st}, \mathsf{tx}) = \mathsf{st}$. Because the state can grow large, it is often useful to provide a succinct representation of state, which we call a *state commitment* and denote by $\langle \mathsf{st} \rangle$. We assume that state commitments have constant size. We denote by $\langle \cdot \rangle$ the commitment function that takes the state and produces the state commitment, *e.g.*, the commitment of the state st is denoted as $\langle \mathsf{st} \rangle$.

For example, in Ethereum, the state is the list of all account balances (among other data). The state is organized into a Merkle tree and its root is the state commitment. A value-transfer transaction tx modifies an *element* of the state st at two positions: the balances of an *outgoing* and the balances of an *incoming* account. The new state st' is determined as $\mathsf{st}' = \delta(\mathsf{st}, \mathsf{tx})$. The transaction is deemed invalid if the outgoing account has insufficient balance. In that case, applying the transaction leaves the balances unchanged.

Light clients. A *light client* wishes to find out a particular state element, *e.g.*, its own account balance. However, it wishes to do so without downloading the whole ledger or executing all the transactions. As in the SPV model, the light client will download and verify the *header chain*, *e.g.*, the longest chain headers containing *transaction roots*, but not every transaction (Transaction root might be the root of the Merkle tree of transactions organized within a block.) While a full node needs to download data proportional to $\mathcal{O}(|\mathcal{C}| + L)$, we want a light node to learn its desired state element by downloading asymptotically less data. We call a lazy light client *succinct* if instead of L, it only needs to download $\operatorname{poly}\log L$ transactions.

The Prover–Verifier model. We are interested in a *light client* $\mathcal{V}$ who is booting up the network for the first time. The light client holds only the *genesis block* and is always honest. It connects to full nodes who have fully synchronized with the rest of the network. The light client acts as a *verifier*, while the full nodes act as *provers* [32]. We assume at least one of the provers $\mathcal{P}_h$ that the verifier connects to is honest (the standard *non-eclipsing* assumption [23,24,26,47]), but the rest could be adversarially controlled. The honest parties follow the protocol specified here. The adversary can run any probabilistic polynomial-time algorithm.

Network. Time proceeds in discrete rounds numbered with the natural numbers $\{0, 1, \cdots\}$. Our network treatment is in the *synchronous* model, where a message sent by one honest party at the end of round r is received by all honest parties at the beginning of round $r + 1$. Our adversary can inject arbitrary messages to the network. Communication is over anonymous channels that the adversary can Sybil attack [20] by introducing as many honest-appearing messages as she wishes. She can also reorder the messages sent by honest nodes and deliver them in a different order to different honest nodes. However, the adversary cannot censor honest messages – honestly broadcast messages are guaranteed to be delivered to all honest parties.

Algorithm 1 Creates a Merkle tree using the given data.

```
function MakeMT(data)
    if $|\text{data}| = 1$ then
        return Tree(root: $H(\text{data}[0])$, data: data[0])
    end if
    mid ← $\lfloor \frac{|\text{data}|}{2} \rfloor$
    tree ← Tree(
      left: MakeMT(data[:mid]),
      right: MakeMT(data[mid:])
    )
    tree.root ← $H(\text{tree.left.root} \parallel \text{tree.right.root})$
    return tree
end function
```

Primitives. We use H to denote a generic cryptographically secure hash function. For our purposes, we require the function to be *collision-resistant* [36]. We make use of Merkle Trees [37] and their variants. Our Merkle trees always contain data whose length is a power of 2(*cf.* Algorithm 1). For data that has different lengths, we organize it into a Merkle Mountain Range [35,46,19]. In a Merkle Mountain Range, the data is split into chunks with length decreasing powers of 2. Each such chunk of data is organized into a Merkle Tree. The resulting roots, the *peaks*, are all *collected* and hashed together to produce the Merkle Mountain Range root.

3 Construction

Initially, the verifier downloads the header chain. This can later be used as a reference to check transaction inclusion and ordering. Because we are working with a lazy blockchain, there are no state commitments within the headers. The rest of the protocol will allow us to determine the state.

Augmented Dirty Ledgers and Dirty Trees. The prover needs to create a tree representing all transactions and states in the system according to its local dirty ledger. Towards this purpose, the prover *augments* each element of its dirty ledger $\mathbb{L}$ and produces an *augmented dirty ledger* $\mathbb{L}_+$. Every transaction in the original dirty ledger is replaced, in the augmented dirty ledger, with a pair. The pair contains the original transaction as well as a commitment to the state of the world after this transaction is applied. Thus, the augmented dirty ledger contains pairs in the form of $(\mathsf{tx}, \langle \mathsf{st} \rangle)$ of a transaction tx and a state commitment $\langle \mathsf{st} \rangle$, one for each transaction in the dirty ledger $\mathbb{L}$. In addition, it contains as a first element, the pair $(\epsilon, \langle st_0 \rangle)$, where the first pair entry is the empty string (because there is no genesis transaction) and the second entry is the commitment to the genesis state $\langle st_0 \rangle$. The state commitment of $\mathbb{L}_+[i+1]$ is computed by applying the transaction at $\mathbb{L}[i]$ to the state committed to by $\mathbb{L}_+[i]$. Concretely, if $\mathbb{L} = (\mathsf{tx}_1, \mathsf{tx}_2, \ldots)$, then

$$\mathbb{L}_+ = ((\epsilon, \langle st_0 \rangle), (\mathsf{tx}_1, \langle \delta(st_0, \mathsf{tx}_1) \rangle), (\mathsf{tx}_2, \langle \delta(\delta(st_0, \mathsf{tx}_1), \mathsf{tx}_2) \rangle), \ldots) \,.$$

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.

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 the view of $\mathcal{P}$ at round r. Let $\mathbb{L}$ denote the dirty ledger, $\mathbb{L}_+$ denote the augmented dirty ledger, and $\mathcal{T}$ denote the corresponding dirty tree 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 state st. The client wishes to learn the value of a particular state element in st. Towards this purpose, $\mathcal{V}$ only needs to learn a truthful state commitment $\langle \mathsf{st} \rangle$; from there, an inclusion proof, *e.g.*, a Merkle proof, into $\langle \mathsf{st} \rangle$ suffices to show inclusion of any state element value. So the goal of $\mathcal{P}$ will be 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 the client can rest assured that the received state commitment is correct (because at least one prover is honest). If they differ, the client must discover the truth. If at any point in time, one of the provers fails to respond in a timely fashion (within one round of receiving its query), the prover is considered adversarial and ignored thereafter. In practice, this equates to a short timeout in the network connection.

The two provers claim that the state commitment is $\langle \mathsf{st} \rangle$ and $\langle \mathsf{st} \rangle^*$ respectively, where $\langle \mathsf{st} \rangle \neq \langle \mathsf{st} \rangle^*$. To prove the correctness of its state commitment, $\mathcal{P}$ sends to $\mathcal{V}$ the root $\langle \mathcal{T} \rangle$ of $\mathcal{T}$, the length $\ell = |\mathbb{L}_+|$ of its augmented dirty ledger, *i.e.*, the number of leaves on $\mathcal{T}$, and the Merkle proof π from the last leaf $\mathbb{L}_+[-1]$, which contains $\langle \mathsf{st} \rangle$, to the root $\langle \mathcal{T} \rangle$. The adversary $\mathcal{P}^*$ sends to $\mathcal{V}$ an alleged root $\langle \mathcal{T} \rangle^*$, 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 $\langle \mathcal{T} \rangle \neq \langle \mathcal{T} \rangle^*$. In this case, $\mathcal{V}$ mediates a *bisection game* between $\mathcal{P}$ and $\mathcal{P}^*$ to determine whether $\langle \mathcal{T} \rangle$ or $\langle \mathcal{T} \rangle^*$ commit to $\langle \mathsf{st} \rangle$.

We define a *well-formed* ledger and tree to be those constructed according to our honest algorithm.

Definition 1 (Well-formed Ledgers and Trees). *An augmented dirty ledger* $\mathbb{L}_+$ *is said to be* well-formed *with respect to dirty ledger* $\mathbb{L}$, *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* $(\mathsf{tx}_{i-1}, \mathsf{tx}_i) \mid \mathbb{L}$, *and* $\mathsf{st}_i = \delta^*(st_0, \mathbb{L}[:i])$.

A dirty tree $\mathcal{T}$ *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 dirty tree held by an honest prover are always well-formed. Hence, to determine whether $\langle \mathcal{T} \rangle$ or $\langle \mathcal{T} \rangle^*$ commit to $\langle \mathsf{st} \rangle$, it suffices for the verifier to check if $\langle \mathcal{T} \rangle$ or $\langle \mathcal{T} \rangle^*$ correspond to the root of a *well-formed* dirty tree.

3.1 Bisection Game

The bisection game is played between two provers, one designated as a *challenger* and the other designated as a *responder*. The challenger sends queries to the responder through the verifier, upon which the responder replies through the same channel (*cf.* Figure 1). In particular, the challenger first sends his query to the verifier. The verifier then forwards this query to the responder. Upon receiving the query from the verifier, the responder produces a response and sends it back to the verifier. Lastly, the verifier forwards this response to the challenger. Subsequently, the challenger can follow up with more queries. As the verifier forwards queries and responses, it performs sanity checks on both of them to ensure that they are well-formed and that the responses correctly correspond to the queries asked. All of the communication of the bisection game between the two provers passes through the verifier. The purpose of the challenger is to convince the verifier that the responder's dirty tree root does *not* correspond to the root of a well-formed tree. The purpose of the responder is to defend his claim that his dirty tree root *is* the root of a well-formed tree. We will design the game in such a fashion that an honest challenger will always win against an adversarial responder, and an honest responder will always win against an adversarial challenger.

Algorithm 2 The algorithm ran by an honest challenger to identify the first point of disagreement against the responder's dirty tree, given that the trees of the challenger and the responder have the same size ℓ, but different roots. The variable $\mathcal{T}$ represents the challenger's dirty tree.

```
function TreeVsTree($\mathcal{T}$, $\ell$)
    if $\ell = 1$ then
        return                                    ▷ We are done; let the verifier check the leaf
    end if
    $(h_l^*, h_r^*) \dashleftarrow$ Responder               ▷ Ask to open inner node
    $(h_l, h_r) \leftarrow \mathcal{T}$.left.root, $\mathcal{T}$.right.root
    if $h_l = h_l^*$ then
        $1 \dashrightarrow$ Responder
        TreeVsTree($\mathcal{T}$.right, $\lfloor \frac{\ell}{2} \rfloor$)
    else
        $0 \dashrightarrow$ Responder
        TreeVsTree($\mathcal{T}$.left, $\lfloor \frac{\ell}{2} \rfloor$)
    end if
end function
```

The game proceeds in a *binary search* fashion similar to refereed delegation of computation [13,12,28]. 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 dirty 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 of the well-formedness in the honest augmented dirty ledger.

We will now describe how the protocol allows the challenger to discover the first point of disagreement. The honest challenger runs Algorithm 2, whereas the honest responder runs Algorithm 3. The challenger first asks the responder to reveal the left and the right children of the responder's dirty tree root $\langle \mathcal{T} \rangle^*$. Let h_l^* and h_r^* denote the children returned by the responder (*cf.* Figure 1 and Algorithm 2 Line 5 and Algorithm 3 Line 6). Let h_l and h_r be the children of the challenger's root $\langle \mathcal{T} \rangle$. The verifier then checks whether $H(h_l^* \parallel h_r^*) = \langle \mathcal{T} \rangle^*$. If the check fails, the responder is a liar, and the verifier designates him as a loser in the game (Algorithm 2). Otherwise, as $\langle \mathcal{T} \rangle \neq \langle \mathcal{T} \rangle^*$, either $h_l \neq h_l^*$ or $h_r \neq h_r^*$. If $h_l = h_l^*$, then necessarily $h_r \neq h_r^*$, and the challenger repeats the same step with the two children of h_r^*. Otherwise, $h_l \neq h_l^*$ and the challenger continues with the children of h_l^* (Lines 9 and 12 of Algorithm 2).

Algorithm 3 The algorithm ran during the bisection game 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 Merkle Tree. The variable $\mathbb{L}_+$ denotes the responder's augmented dirty ledger.

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

These queries continue until the challenger reaches a leaf with index j. The verifier forwards and verifies exactly up to $\log \ell$ inner node queries and one leaf query. Then, the challenger pinpoints the location j as the first point of disagreement, and the honest responder reveals the leaf data $(\mathsf{tx}_j, \langle \mathsf{st}_j \rangle)$ (Algorithm 3 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 of a well-formed augmented dirty ledger). 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.

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 generated by the MakeMerkleTree algorithm as $\langle \mathcal{T} \rangle^* \neq \langle \mathcal{T} \rangle$.

The verifier algorithm is described by Algorithm 2.

3.2 Ledgers of Different Sizes

In the previous section, we assumed that the sizes of the augmented dirty ledgers were powers of two, and so they could easily be organized into a Merkle tree. In reality, these could attain other values. To organize such ledgers, provers use Merkle Mountain Ranges (MMRs). An MMR is a sequence of Merkle trees of decreasing size, each of which is a power of two. Provers construct their MMRs using Algorithm 4, where the input data is their augmented dirty ledger. 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. This is always possible by writing the length $|\mathbb{L}_+|$ of the augmented dirty ledger in binary form and looking at the '1's: $|\mathbb{L}_+| = \sum_{i=1}^{k} 2^{q_i}$. Each of these k segments is then organized into a Merkle tree, and those trees $\mathcal{T} = (\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k)$ are all collected into an array, which 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 Merkle trees $\mathcal{T} = (\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k)$, where $\langle \mathcal{T} \rangle_i = \langle \mathcal{T}_i \rangle$, are called the *peaks.* Besides organizing data of different sizes, MMRs have the added benefit of being easily amendable.

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 $(j-1)^{\text{st}}$ 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.* Section 5.2). 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.* Section 5.3).

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. 2: The algorithm ran by the verifier to determine the winner of the bisection game.

Algorithm 4 Creates a Merkle Mountain Range using the given data for the leaves.

```
function MakeMMR(data)
    if |data| = 0 then
        return []
    end if
    bound ← $2^{\lfloor \log |\text{data}| \rfloor}$
    head ← MakeMT(data[:bound])
    tail ← MakeMMR(data[bound:])
    return (head:tail)
end function
```

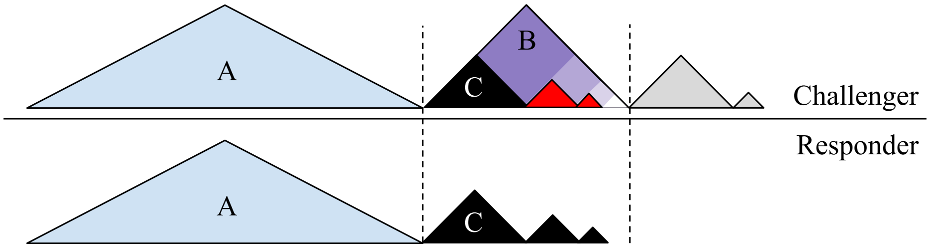

Fig. 3: The challenger's MMR (top) is compared to the responder's alleged MMR. The first two peaks (A in blue) are the same, so they are skipped by Algorithm 6. The second peak of the challenger is reached (B in purple) and compared against the responder's second peak (C). When found to be different, the challenger knows all of the remaining responder peaks (the black at the bottom) will lie within his own current tree (B in purple), so Algorithm 6 Line 4 calls Algorithm 7 to compare the black peaks against the purple tree.

The following fact will be useful and follows from the organization of the tree into powers of two:

Fact 1 (Tree Subsumption). *In any Merkle Mountain Range $\mathcal{T}$ containing trees of lengths $\{\ell_i\}_{i \in [k]}$, for any $i \in [k]$ it holds that $\ell_i > \sum_{j=i+1}^{k} \ell_j$.*

Additionally, it is possible that even two honest provers have augmented dirty ledgers of different sizes $\ell \neq \ell^*$. This is an artifact of network delays. We will now explore our full protocol that allows the verifier to compare competing claims of different sizes by two provers.

Suppose an honest prover $\mathcal{P}$ holding an augmented dirty ledger $\mathbb{L}_+^{\mathcal{P}}$ claims to a verifier $\mathcal{V}$ that the latest state commitment is $\langle \mathsf{st} \rangle$. Then, to prove its claim, $\mathcal{P}$ first organizes its augmented dirty ledger into an MMR and sends its peaks $\langle \mathcal{T} \rangle = (\langle \mathcal{T} \rangle_1, \langle \mathcal{T} \rangle_2, \ldots, \langle \mathcal{T} \rangle_k)$ to $\mathcal{V}$, together with the length $\ell = |\mathbb{L}_+^{\mathcal{P}}|$. It also sends a Merkle proof π from the last augmented dirty ledger entry $\mathbb{L}_+^{\mathcal{P}}[-1]$, which contains $\langle \mathsf{st} \rangle$, to the last Merkle tree root $\langle \mathcal{T} \rangle_k$.

Suppose another prover $\mathcal{P}^*$ also claims to $\mathcal{V}$ that the latest state commitment is $\langle \mathsf{st} \rangle^* \neq \langle \mathsf{st} \rangle$. To back his claim, $\mathcal{P}^*$ also sends to $\mathcal{V}$ an alleged sequence of peaks $\langle \mathcal{T} \rangle^* = (\langle \mathcal{T} \rangle_1^*, \langle \mathcal{T} \rangle_2^*, \ldots, \langle \mathcal{T} \rangle_{k^*}^*)$, an alleged length ℓ^*, and an alleged Merkle proof π^*. In this case, $\mathcal{V}$ first checks whether π and π^* verify with respect to $\langle \mathcal{T} \rangle_k$ and $\langle \mathcal{T} \rangle_{k^*}^*$. If the proof given by a prover cannot be verified, $\mathcal{V}$ rejects his claim. Otherwise, if $(\langle \mathcal{T} \rangle, \ell) \neq (\langle \mathcal{T} \rangle^*, \ell^*)$, then $\mathcal{V}$ initiates a *challenge game* to find which of the two provers holds a *well-formed* augmented dirty ledger.

During the challenge game, the prover that claimed to have a *larger* augmented dirty ledger $\mathbb{L}_+$ acts as the challenger while the other one, with the *smaller* augmented dirty ledger $\mathbb{L}_+^*$, 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. He starts the challenge by calling Algorithm 5.

Algorithm 5 The algorithm ran by the challenger to start the challenge game. The challenger's goal is to identify the first point of disagreement against the responder's alleged augmented dirty ledger. The variable $\mathbb{L}_+$ denotes the challenger's augmented dirty ledger. The variable $\langle\mathcal{T}\rangle^*$ denotes the peaks received from the responder, and ℓ^* denotes the augmented dirty ledger size claimed by the responder. The function GetSizes takes the length of an augmented dirty ledger and returns the sequence of number of leaves under the Merkle trees in the corresponding MMR.

```
function Challenge($\mathbb{L}_+$)
    trees $\leftarrow$ MakeMMR($\mathbb{L}_+$)
    sizes $\leftarrow$ GetSizes($|\mathbb{L}_+|$)
    $\ell^*, \langle\mathcal{T}\rangle^* \dashleftarrow$ Responder
    reSizes $\leftarrow$ GetSizes($\ell^*$)
    FindPeak(trees, sizes, $\langle\mathcal{T}\rangle^*$, reSizes)
end function
```

The challenge game consists of two phases: During the first phase, the *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*, similar to the previous section, or the challenger going into a *suffix monologue*.

Zooming phase. To narrow his search down to a single tree, the challenger first calls Algorithm 6 to identify the earliest peak in the responder's peaks that disagrees with his own peaks. Algorithm 6 iterates over the responder's peaks (Algorithm 6 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 calls Algorithm 2 to play the bisection game of the previous section on the Merkle trees whose roots are $\langle \mathcal{T}_i \rangle$ and $\langle \mathcal{T} \rangle^*_i$ (Algorithm 6 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, by Fact 1, all the data within the responder's remaining peaks allegedly lies under the i^{th} peak of the challenger (see Figure 3). The challenger has now narrowed the first point of disagreement to his own i^{th} tree and can compare it against the responder's remaining peaks. This is done by calling Algorithm 7 on the remaining peaks returned by the responder (Algorithm 6 Line 4).

Algorithm 6 The algorithm ran by the challenger to identify the first peak in the MMR of the responder 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.

```
1:  function PeaksVsPeaks($\mathcal{T}, \overline{\ell}, \langle \mathcal{T} \rangle^*, \overline{\ell}^*$)
2:      for $i = 0$ to $|\langle \mathcal{T} \rangle^*| - 1$ do
3:          if $\overline{\ell}[i] \neq \overline{\ell}^*[i]$ then
4:              return TreeVsPeaks($\mathcal{T}[i], \langle \mathcal{T} \rangle^*[i{:}], \overline{\ell}^*[i{:}]$)
5:          end if
6:          if $\mathcal{T}[i].\text{root} \neq \langle \mathcal{T} \rangle^*[i]$ then
7:              return TreeVsTree($\mathcal{T}[i], \overline{\ell}[i]$)
8:          end if
9:      end for
10: end function
```

Algorithm 6 already narrowed the challenger's search to within one of its trees (purple tree denoted B in Figure 3). Now the goal of Algorithm 7 is to narrow down the search of the first point of disagreement to one of the peaks of the *responder*, so that the verifier can compare the two trees using the earlier bisection game. Consider the *remaining* peaks of the responder overlayed onto the challenger's tree $\mathcal{T}_i$ (dashed lines in Figure 3). These correspond to certain inner nodes within $\mathcal{T}_i$ (black, red, and red subtrees at the top of Figure 3). Algorithm 7 locates the first such inner node that disagrees with the corresponding responder's peak (left-most red tree at the top of Figure 3).

To find the first disagreeing inner node, the challenger traverses a path on $\mathcal{T}_i$ that starts at $\langle \mathcal{T}_i \rangle$ and ends at the root of the desired subtree within

Algorithm 7 The algorithm ran by the challenger to identify the first subtree root under one of the challenger's larger Merkle trees that is different from a 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.

```
 1: function TreeVsPeaks($\mathcal{T}$, $\langle\mathcal{T}\rangle^*$, $\overline{\ell}^*$)
 2:     assert $\mathcal{T}$.size $> \sum_{\ell^* \in \overline{\ell}^*} \ell^*$
 3:     if $|\langle\mathcal{T}\rangle^*| = 0$ then
 4:         return                                              ▷ Done
 5:     end if
 6:     (peak:peaks) ← $\langle\mathcal{T}\rangle^*$
 7:     (reSize:reSizes) ← $\overline{\ell}^*$
 8:     if $\lfloor \frac{\mathcal{T}.\text{size}}{2} \rfloor >$ reSize then
 9:         TreeVsPeaks($\mathcal{T}$.left, $\langle\mathcal{T}\rangle^*$, reSizes)
10:     else if tree.left.root = peak then
11:         TreeVsPeaks($\mathcal{T}$.right, peaks, reSizes)
12:     else
13:         ChallengeTree($\mathcal{T}$, reSize)
14:     end if
15: end function
```

$\mathcal{T}_i$. Let $\tilde{\mathcal{T}}$ denote an inner node of $\mathcal{T}_i$ that lies on the path. Let $\tilde{\mathcal{T}}$.left and $\tilde{\mathcal{T}}$.right denote the children of $\mathcal{T}$. While exploring $\tilde{\mathcal{T}}$, the challenger checks if the number of leaves under $\tilde{\mathcal{T}}$.left exceeds the total number of leaves under the responder's remaining roots (Algorithm 7 Line 8). If this is the case, all of the responder's remaining roots must lie within $\tilde{\mathcal{T}}$.left. Thus, the challenger moves to $\tilde{\mathcal{T}}$.left (Algorithm 7 Line 9). Otherwise, the challenger checks if the root of $\tilde{\mathcal{T}}$.left, *i.e.*, the left child of $\tilde{\mathcal{T}}$, equals the first peak among the remaining ones returned by the responder (Algorithm 7 Line 10). If so, the first point of disagreement is later, and so the challenger moves to $\tilde{\mathcal{T}}$.right (Algorithm 7 Line 11). The search ends when $\tilde{\mathcal{T}}$.left differs.

Bisection game. Once the search ends, the challenger calls Algorithm 2 to play the bisection game on the sub-trees under $\tilde{\mathcal{T}}$ and the responder's currently inspected root (Algorithm 7 Line 13). At the end of the algorithm, just like before, a first point of disagreement is pinpointed. We remark here that, if the point j of disagreement is the first leaf of this tree, then the last step of the algorithm, which requires the verifier to execute the state transition, will require the responder to also open up leaf $j - 1$. Note that this opening will have to be with respect to the

Algorithm 8 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.

```
function RESPOND($\mathbb{L}_+$)
    $\mathsf{trees} \leftarrow \text{MAKEMMR}(\mathbb{L}_+)$
    $\mathsf{peaks} \leftarrow \{\mathsf{tree}.\text{root} : \mathsf{tree} \in \mathsf{trees}\}$
    $\mathsf{peaks} \dashrightarrow \text{CHALLENGER}$
    $\mathsf{pNum} \dashleftarrow \text{CHALLENGER}$
    $\mathsf{tree} \leftarrow \mathsf{trees}[\mathsf{pNum}]$                ▷ Enter a particular Merkle Tree
    while $\mathsf{tree}.\text{size} > 1$ do
        $(\mathsf{tree}.\text{left}, \mathsf{tree}.\text{right}) \dashrightarrow \text{CHALLENGER}$
        $\mathsf{dir} \dashleftarrow \text{CHALLENGER}$
        if $\mathsf{dir} = 0$ then
            $\mathsf{tree} \leftarrow \mathsf{tree}.\text{left}$
        else
            $\mathsf{tree} \leftarrow \mathsf{tree}.\text{right}$
        end if
    end while
    $\mathsf{dirtyLedger}[\mathsf{loc}] \dashrightarrow \text{CHALLENGER}$
end function
```

previous peak. After the bisection game is played, one of the challenger or the responder is declared the winner and the other one is declared the loser.

Suffix monologue. It is possible that the challenger and the responder will never enter into a bisection game, if none of the *return* statements in Algorithm 6 are reached, or if Algorithm 7 Line 4 is reached. In that case, there is no first point of disagreement between the two. Therefore, the two alleged augmented dirty ledgers form a prefix of one another: the responder's alleged $\mathbb{L}_+^*$ of length ℓ^* is a prefix of the challenger's alleged $\mathbb{L}_+$ of length $\ell > \ell^*$ (if $\ell = \ell^*$, and given that the two parties have a disagreement, then they must have played a bisection game; thus at this point, $\ell > \ell^*$). This situation can arise even when both provers are honest due to network synchronization issues. In particular, the challenger could have seen more recent transactions on the network. (Due to the Common Prefix property [23], the discrepancy between two honest provers is bounded to ledger extensions that correspond to k blocks.) Hence, it is the challenger's turn to present the augmented dirty ledger entries that extend the responder's augmented dirty ledger. Specifically, the challenger must present the *suffix* $\mathbb{L}_+[\ell^*:]$.

The verifier checks all 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 $\ell^* \leq j < |\mathbb{L}_+| - 1$, 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 augmented dirty ledger 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 entry $\mathbb{L}^*_+[-1]$ has already been revealed, together with a proof of inclusion, by the responder at the beginning of the 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, then *both* provers are declared winners of the challenge game. Unlike the bisection game, at the end of the suffix monologue, *both* the challenger and the responder can win.

Towards succinctness, the verifier checks only the first $\alpha(u + \nu)$ extra entries from the challenger's augmented dirty ledger extension. Here, α, u and ν are constants selected in accordance with the chain growth and liveness parameters of the blockchain (*cf.* Table 1). Therefore, in reality, the challenger only needs to send the portion $\mathbb{L}_+[\ell^* \colon \min(\ell, \ell^* + \alpha(u+\nu))]$ of his suffix. If the challenger is able to present more than $\alpha(u + \nu)$ extra entries with consecutive transactions and correct state transitions, then the responder is declared a loser, as he presented a ledger that is too short to possibly be honest, due to the Common Prefix property.

3.3 Multiparty Tournaments

Upon joining the network, the verifier contacts the set of all available provers[3] for queries. If all of the responses are the same, the verifier accepts the response as the correct answer. If it receives different responses, the verifier arbitrates a *tournament*, described by Algorithm 9, among the provers that responded. The purpose of the tournament is to select a prover whose latest claimed stable state is both *safe* and *live* (*cf.* Definition 9). That is, the verifier wants to obtain a state at least as up-to-date as the state obtained by applying the state transition function iteratively on the transactions in $\mathbb{L}^\cap$.

[3] In practice, the verifier contacts a small subset of them. As a concrete example, according to `https://github.com/bitcoin/bitcoin/blob/master/doc/reduce-traffic.md`, Bitcoin makes 8 outbound full-relay connections.

Suppose the verifier hears back from n provers. Before the tournament, the verifier orders the n provers into a sequence $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ in an increasing order of the augmented dirty ledger sizes claimed by the provers (*i.e.*, the size of $\mathcal{P}_1$'s alleged MMR is equal to or smaller than that of $\mathcal{P}_2$ and so on). This sequence dictates the order in which the provers play the bisection game with each other. Then, it calls Algorithm 9 to start the tournament.

The tournament consists of n steps. Before the first step, Algorithm 9 initializes $\mathcal{S} = \emptyset$. At the end of each step t, the set $\mathcal{S}$ contains each prover that has engaged in at least one challenge game, and has not lost any of those by step t.

Algorithm 9 The tournament among the provers administered by the verifier. It takes a sequence of provers $\mathcal{P}$, ordered in a decreasing fashion according to the alleged augmented dirty ledger sizes.

```
function TOURNAMENT($\mathcal{P}$)
    sizes ← { }
    for $p \in \mathcal{P}$ do
        sizes[$p$] ← $p$.getsize()
    end for
    $\mathcal{S} \leftarrow \{\mathcal{P}[0]\}$
    largest ← $\mathcal{P}[0]$
    for $i = 1$ to $|\mathcal{P}| - 1$ do
        do
            if largest.getsize() > sizes[$i$] then
                result ← CHALLENGE(largest, $\mathcal{P}[i]$)
            else
                result ← CHALLENGE($\mathcal{P}[i]$, largest)
            end if
            if result is "nested MMRs" then
                $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathcal{P}[i]\}$
            else if result is "largest loses" then
                $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\text{largest}\}$
                largest ← $\arg\max_{p \in \mathcal{S}}$ sizes
            else if result is "$\mathcal{P}[i]$ loses" then
                ▷ The set $\mathcal{S}$ does not need to be updated
            end if
        while result is "largest loses" $\wedge\, \mathcal{S} \neq \emptyset$
        if $\mathcal{S} = \emptyset$ then
            $\mathcal{S} \leftarrow \{\mathcal{P}[i]\}$
        end if
    end for
    return largest
end function
```

The tournament starts with a challenge game between $\mathcal{P}_1$ and $\mathcal{P}_2$, during which the prover with the larger alleged augmented dirty ledger challenges the other. 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. 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 round. The prover among $\{\mathcal{P}, \overline{\mathcal{P}}\}$ with the larger alleged augmented dirty ledger challenges the other prover (Lines 11 and 13 of Algorithm 9). 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 3 (Line 16 of Algorithm 9).
2. If $\mathcal{P}$ loses, the tournament directly moves to step 3 and $\mathcal{S}$ stays the same. (Line 21 of Algorithm 9).
3. If $\mathcal{P}$ wins and $\overline{\mathcal{P}}$ loses, $\overline{\mathcal{P}}$ is removed from $\mathcal{S}$ (Line 18 of Algorithm 9). 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 $\mathcal{S}$ (Line 23 of Algorithm 9). If the latter happens, $\mathcal{P}$ is added to $\mathcal{S}$ and the tournament moves to step 3 (Line 25 of Algorithm 9).

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

The above procedure consists of $\mathcal{O}(n)$ bisection games. 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 party can be eliminated at most once.

3.4 Prover Complexity

Given an augmented dirty ledger of length $\ell = \sum_{i=1}^{k} 2^{q_i}$, $q_i \in \mathbb{N}$, each prover can construct the corresponding MMR with $O(\ell)$ number of computations 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 $O(\ell)$.

Moreover, provers need not create a new MMR from scratch at each challenge game. Full nodes can maintain an MMR of the augmented dirty ledger in their memory and augmented the MMR when a new entry is added to the ledger in their view. A node that holds an MMR with ℓ leaves in total can construct the MMR containing a single new leaf with $O(\log \ell)$ hash computations since the prover only needs to combine the existing $\log \ell$ hashes to update the MMR. Hence, per each new transaction, each prover only incurs logarithmic compute complexity. In particular, updating the MMR on a rolling basis, each prover can obtain the MMR with ℓ leaves with $O(\ell \log \ell)$ number of operations. Thus, our algorithms require in total quasi-linear number of operations in the ledger size from each full node over the lifetime of the protocol.

4 Implementation

4.1 Latency-Bandwidth Trade-off

So far, we have only discussed *binary* Merkle trees for use in our bisection game. However, we can consider m-ary Merkle trees more generally. In a dirty tree representing an L-sized ledger, the dirty tree height decreases logarithmically as the degree of the tree increases, making the number of rounds of interactivity in the game $\log_m L$. At the same time, the length of each message of the challenger is now no longer a single bit, but needs to indicate the index of the child to open, making it $\log m$ bits in size. 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 network 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 can compute the corresponding reply instantly (network latency dominates computational latency).

At the beginning of the bisection game, the responder sends the m hash values corresponding to the children of the root node on its dirty tree. The challenger then selects one of them and sends back $\log m$ bits, through the verifier, to specify the selected child. This is repeated for

$\log_m L$ rounds until the challenger specifies one of the leaves on the responder's dirty tree to be opened.

At each round, $\log m$ 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, making the total running time of the bisection game $\left(4\Delta + \frac{mH+\log m}{C}\right)\log_m L = \frac{\log L}{\log m}\left(4\Delta + \frac{mH}{C}\right) + \frac{\log L}{C}$. This expression is minimized for m that satisfies the expression $m(\log m - 1) = \frac{4\Delta C}{H}$, *i.e.*, $m = \exp\left(W_1(4\Delta\frac{C}{eH}) + 1\right)$, where W_1 is the Lambert W function. The different optimal m for a range of common bandwidths and latencies are plotted in Figure 4.

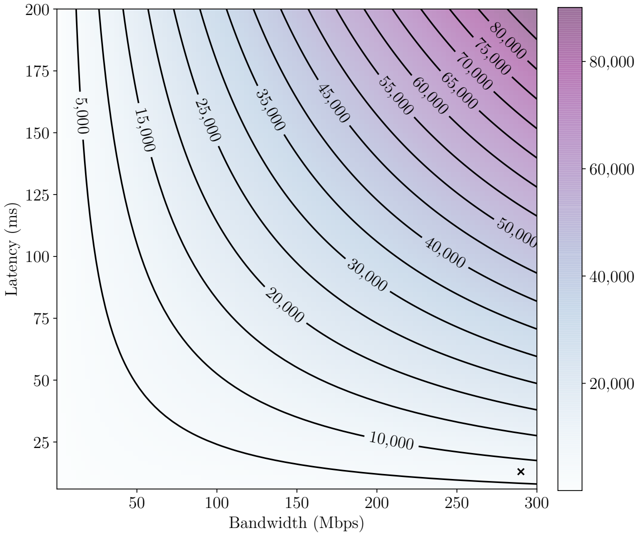

Fig. 4: Optimal Merkle tree degree m (isolines) for a given network connection bandwidth (x-axis, in Mbps) and latency (y-axis, in ms). The × marker marks the particular example described in the text.

Given[4] $C = 290$ Mbps, $\Delta = 13$ ms, $H = 256$ bits, the optimal m is 7,442, yielding dirty trees that have quite a large degree. Note that the optimal m only depends on the network parameters and not the ledger

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

size. Given the Ethereum ledger of $L = 1.5 \cdot 10^9$ transactions at the time of writing[5], 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 $\mathit{\Delta}$. This captures the duration of the whole bisection game with its $\log_m L$ rounds of interactivity.

4.2 Experiments

Implementation and experimental setup. We implement a prototype of the prover and the verifier in 1000 lines of Golang[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. Each of the remaining 16 provers has an augmented dirty ledger that differs 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). Instead, all transactions and state commitments (*cf.*. Algorithm 2) 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 9) among them.

Verification latency. We first explore the duration of the tournament, and how the degree m of the Merkle tree affects the performance. For this experiment, we fix the ledger size to 10 million transactions vary m. For each configuration, we run 10 tournaments and measure the average and the standard deviation of the duration. Figure 5a shows the results. 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.

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

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

As we vary m, tournaments take longer to finish. Specifically, a smaller m makes each Merkle tree opening shorter, but increases the number of openings per game. The propagation latency becomes the main bottleneck of the game. On the contrary, a larger m makes bandwidth the main bottleneck. As we increase 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 and vary the size of the ledger from 1000 to 100 million transactions. Similar to the previous experiment, we report the mean and the standard deviation over 10 tournaments for each datapoint. Figure 5b shows the results. 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.

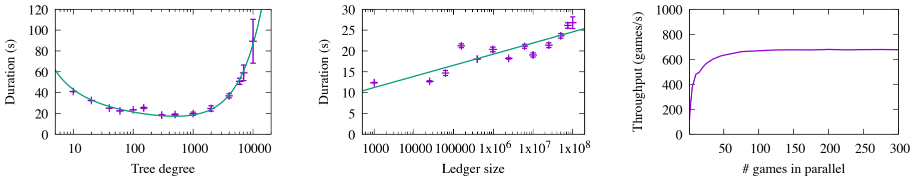

Fig. 5: In (a) and (b), we measure the time to complete a tournament of 17 geo-distributed provers. Error bars show the standard deviation. Solid lines show the trend. (a) Time when varying the tree degree m. (b) Time when varying the ledger size L. (c) Throughput of games with two provers and one verifier co-located in a data center. The verifier initiates games with variable parallelism to saturate the provers.

Prover throughput. Finally, we evaluate the throughput that a prover can achieve. To minimize the network influence, we run two provers in the same datacenter. Each prover has a ledger of 10M transactions, but their ledgers 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. Figure 5c shows the results. As we ramp up the parallelism, the achieved throughput first increases due to the increased load, and then stays flat because the provers have saturated their compute. 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.

4.3 Denial of service attacks

While our theoretical model assumes synchrony, in practice the verifiers and provers may be susceptible to *denial of service* attacks in which invalid data is sent on the network. To deal with this, provers that submit invalid data should be blacklisted by the verifier, while provers that timeout can be ignored after a few strikes. Since bisection games typically take a few seconds, avoiding denial of service attacks for our construction is not different from avoiding them in standard SPV clients or full nodes. All well known heuristic techniques for avoiding such attacks can be employed here.

5 Generalizing the Model

So far, we have assumed that the underlying consensus protocol is blockchain based and that the computation of state mimicks the Ethereum EVM. However, there exist different architectures for consensus (*e.g.*, DAG-based constructions [22,43,42]) and execution (*e.g.*, UTXO [18]). Our protocol is quite generic and agnostic to the details of how the consensus and execution layers are implemented. To enable the formal analysis of our construction in a generic manner, in this section we axiomatize the protocol requirements regarding consensus and execution.

5.1 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 tx at round r, attempts to include the transaction into the ledger.

Our consensus protocol must satisfy the following properties.

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

Table 1: The 9 axioms required to construct a succinct light client.

Definition 2 (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 3 (Ledger Liveness). *A consensus protocol $\mathcal{C}$ is* live *with liveness parameter u if, when an honest party P attempts to write a transaction* 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 [23,24,33,17,16].

Definition 4 (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 $\lvert\mathbb{L}_{r_2}^{P}\rvert - \lvert\mathbb{L}_{r_1}^{P}\rvert < \alpha(r_2 - r_1)$.*

The above requirement states that ledgers grow at a bounded rate. For blockchain based protocols, this follows from the fact that chains have an upper bound in their growth rate [48, Theorem 4], and that the number of transactions in each block is limited by a constant.

5.2 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 5 (Consensus Oracle Security). *An consensus oracle is* secure *if it satisfies:*

- ***Completeness.*** *When invoked at round r, it holds that $\mathcal{CO}_r(\mathsf{tx}, \mathsf{tx}') =$ true when $(\mathsf{tx}, \mathsf{tx}') \mid \mathbb{L}^\cup_r$.*
- ***Soundness.*** *The consensus oracle is sound with* delay parameter ν *if for any PPT adversary $\mathcal{A}$,*

$$\begin{aligned}&\Pr[(\mathsf{tx}, \mathsf{tx}', r) \leftarrow \mathcal{A}(1^\lambda);\\ &\quad \mathcal{CO}_r(\mathsf{tx}, \mathsf{tx}') \wedge (\mathsf{tx}, \mathsf{tx}') \nmid \mathbb{L}^\cup_{r+\nu}] \leq \mathrm{negl}(\lambda)\,.\end{aligned}$$

Definition 6 (Consensus Oracle Communication Complexity). *A consensus oracle has communication complexity $f(r)$ if the total size of the query and respond 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 imposed on 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:

- the positions i and i' of the transactions tx and tx' within their respective Merkle trees,
- the Merkle proofs π and π' from the transactions tx and tx' to the transaction roots,

– 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 if and only if either of the following cases hold:

1. the two blocks are the same, *i.e.*, $j' = j$, and $i' = i + 1$,
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 B 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: $j = j_{\mathsf{tx}}$, $j' = j_{\mathsf{tx}'}$, $i = i_{\mathsf{tx}}$, $i' = i_{\mathsf{tx}'}$.

5.3 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 there exists a state st such that $\langle \mathsf{st} \rangle$ is the commitment of st and $\langle \mathsf{st} \rangle' = \langle \delta(\mathsf{tx}, \mathsf{st}) \rangle$, 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 7 (Execution Oracle Security). *An execution oracle is* secure *if it satisfies:*

- ***Completeness.*** *The 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 $\mathcal{A}$:*

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

Definition 8 (Execution Oracle Communication Complexity). *An execution oracle has communication complexity $g(r)$ if the total size of the query and respond messages exchanged during an oracle query invoked at round r is $g(r) \in \mathcal{O}(r)$.*

In the account based model [2], the state is a Sparse Merkle Tree (SMT) [15] 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. 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 $\langle \mathsf{st} \rangle$.

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 [12]. 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.

6 Analysis

The security of the protocol relies on the completeness and succinctness of the consensus oracle and the execution oracle.

Definition 9 (State Security). *An interactive Prover–Verifier protocol* (P, V) *is* state secure *with safety parameter* ν*, if the state commitment* $\langle\mathsf{st}\rangle$ *obtained by the verifier at the end of the protocol execution at round* r *satisfies* safety *and* liveness *as defined below.*

There exists a ledger $\mathbb{L}$ *such that* $\langle\delta^*(\mathsf{st}_0, \mathbb{L})\rangle = \langle\mathsf{st}\rangle$*, and for all rounds* $r' \geq r + \nu$*:*

- ***Safety.*** $\mathbb{L}$ *is a* prefix *of* $\mathbb{L}^{\cup}_{r'}$*.*
- ***Liveness.*** $\mathbb{L}$ *is a* suffix *of* $\mathbb{L}^{\cap}_{r}$*.*

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

Lemma 1 (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* $O(\log(\ell_1)) + \alpha(u + \nu)(f(r) + g(r)))$*.*

Theorem 1 (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.*

Theorem 2 (Soundness). *Let H^s be a 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.*

Theorem 3 (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.*

Theorem 4 (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.*

Proofs of the lemmas and theorems above are given in Appendix A.

7 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 [30] or non-interactive PoPoWs, a primitive which can be constructed using either superblocks [32,31] or FlyClient [10], 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.

	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 2: Comparison of different client types on a lazy blockchain.

The interactivity and communication complexity for synchronization times for lazy light clients composed with different consensus oracles is illustrated in Table 2. 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 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.

Acknowledgements

We thank Shresth Agrawal, Kostis Karantias, Angel Leon and Joachim Neu for several insightful discussions on this project.

References

1. Al-Bassam, M.: Lazyledger: A distributed data availability ledger with client-side smart contracts (2019)
2. Al-Bassam, M., Sonnino, A., Buterin, V.: Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities. arXiv:1809.09044 [cs.CR] (2018)
3. Bagaria, V., Kannan, S., Tse, D., Fanti, G., Viswanath, P.: Prism: Deconstructing the blockchain to approach physical limits. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 585–602. CCS '19, ACM (2019)
4. Bagaria, V., Kannan, S., Tse, D., Fanti, G., Viswanath, P.: Prism: Deconstructing the blockchain to approach physical limits. working paper (2019)
5. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptol. ePrint Arch. **2018**, 46 (2018)
6. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. p. 326–349. ITCS '12, ACM, New York, NY, USA (2012)
7. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for snarks and proof-carrying data. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing. pp. 111–120 (2013)
8. Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive (2020)
9. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. arXiv:1807.04938 (2018), `https://arxiv.org/abs/1807.04938`
10. Bünz, B., Kiffer, L., Luu, L., Zamani, M.: Flyclient: Super-light clients for cryptocurrencies. In: 2020 IEEE Symposium on Security and Privacy (SP). IEEE (2020)
11. Buterin, V.e.a.: Light client protocol (2014), `https://eth.wiki/en/concepts/light-client-protocol`
12. Canetti, R., Riva, B., Rothblum, G.N.: Practical delegation of computation using multiple servers. In: Proceedings of the 18th ACM conference on Computer and communications security. pp. 445–454 (2011)
13. Canetti, R., Riva, B., Rothblum, G.N.: Refereed delegation of computation. Information and Computation **226**, 16–36 (2013)
14. Chen, J., Micali, S.: Algorand: A secure and efficient distributed ledger. Theoretical Computer Science **777**, 155–183 (2019)
15. Dahlberg, R., Pulls, T., Peeters, R.: Efficient sparse merkle trees. In: Nordic Conference on Secure IT Systems. pp. 199–215. Springer (2016)
16. Daian, P., Pass, R., Shi, E.: Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In: Financial Cryptography and Data Security. pp. 23–41. FC '19, Springer (2019)
17. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 66–98. Springer (2018)
18. Developers, B.: Developer guide - bitcoin. Available at: `https://bitcoin.org/en/developer-guide`, `https://bitcoin.org/en/developer-guide`

19. Developers, G.: Merkle Mountain Ranges (MMR), `https://docs.grin.mw/wiki/chain-state/merkle-mountain-range/`
20. Douceur, J.R.: The sybil attack. In: International Workshop on Peer-to-Peer Systems. pp. 251–260. Springer (2002)
21. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Annual International Cryptology Conference. pp. 139–147. Springer (1992)
22. Fitzi, M., Gaži, P., Kiayias, A., Russell, A.: Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. Cryptology ePrint Archive, Report 1119 (2018)
23. Garay, J., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: Analysis and applications. In: EUROCRYPT 2015. pp. 281–310. Springer (2015)
24. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol with chains of variable difficulty. In: Katz, J., Shacham, H. (eds.) Annual International Cryptology Conference. LNCS, vol. 10401, pp. 291–323. Springer (Aug 2017)
25. Gaži, P., Kiayias, A., Zindros, D.: Proof-of-Stake Sidechains. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 139–156. IEEE (2019)
26. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: USENIX Security Symposium. pp. 129–144 (2015)
27. van den Hooff, J., Kaashoek, M.F., Zeldovich, N.: Versum: Verifiable computations over large public logs. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 1304–1316 (2014)
28. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 1353–1370 (2018)
29. Karantias, K., Kiayias, A., Zindros, D.: Compact storage of superblocks for nipopow applications. In: The 1st International Conference on Mathematical Research for Blockchain Economy. Springer Nature (2019)
30. Kiayias, A., Lamprou, N., Stouka, A.P.: Proofs of proofs of work with sublinear complexity. In: International Conference on Financial Cryptography and Data Security. pp. 61–78. Springer, Springer (2016)
31. Kiayias, A., Leonardos, N., Zindros, D.: Mining in Logarithmic Space. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 3487–3501 (2021)
32. Kiayias, A., Miller, A., Zindros, D.: Non-Interactive Proofs of Proof-of-Work. In: International Conference on Financial Cryptography and Data Security. Springer (2020)
33. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In: Katz, J., Shacham, H. (eds.) Annual International Cryptology Conference. LNCS, vol. 10401, pp. 357–388. Springer, Springer (Aug 2017)
34. Kiayias, A., Zindros, D.: Proof-of-work sidechains. In: International Conference on Financial Cryptography and Data Security. Springer, Springer (2019)
35. Laurie, B., Langley, A., Kasper, E.: Rfc6962: Certificate transparency. Request for Comments. IETF (2013)
36. Lindell, Y., Katz, J.: Introduction to Modern Cryptography. Chapman and Hall/CRC (2014)
37. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Advances in Cryptology - CRYPTO '87. LNCS, vol. 293, pp. 369–378. Springer (1987). https://doi.org/10.1007/3-540-48184-2_32

38. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf` (2008)
39. Neu, J., Tas, E.N., Tse, D.: Snap-and-chat protocols: System aspects. arXiv preprint arXiv: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 Cryptology ePrint Archive (2016:1159)
43. Sompolinsky, Y., Zohar, A.: Phantom: A scalable blockdag protocol (2018)
44. Sompolinsky, Y., Wyborski, S., Zohar, A.: PHANTOM and GHOSTDAG: A Scalable Generalization of Nakamoto Consensus. IACR Cryptology ePrint Archive (2018), `http://eprint.iacr.org/2018/104`
45. Tas, E.N.: Woods Attack on Celestia, `https://forum.celestia.org/t/woods-attack-on-celestia/59`
46. Todd, P.: Merkle Mountain Ranges (October 2012), `https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md`
47. Wüst, K., Gervais, A.: Ethereum eclipse attacks. Tech. rep., ETH Zurich (2016)
48. Zindros, D.: Hours of Horus: Keyless Cryptocurrency Wallets. Cryptology ePrint Archive (2021)

A 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. Axioms used by the proofs are given by Table 1.

Lemma 2 (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 3 (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 2 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 2 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 5, 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 7, $M(\mathsf{st}_{j-1}, \mathsf{tx}_j)$ 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 $\textsc{Verify}(\pi, \langle \mathcal{T} \rangle, i, v)$ be the verification function for Merkle proofs. It takes a proof π, a Merkle root $\langle \mathcal{T} \rangle$, an index for the leaf i 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 [36].

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

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 $a = \log |D| + 1$ 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 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, $h_{i,m} = h_i^* \,\|\, \tilde{h}_i^*$ and $h_{i,c} = h_i \,\|\, \tilde{h}_i$.
- If h_i is the right child of its parent, $h_{i,m} = \tilde{h}_i^* \,\|\, h_i^*$ and $h_{i,m} = \tilde{h}_i \,\|\, h_i$.

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, $\Pr[\text{Merkle-Collision}] = \text{negl}(\lambda)$. □

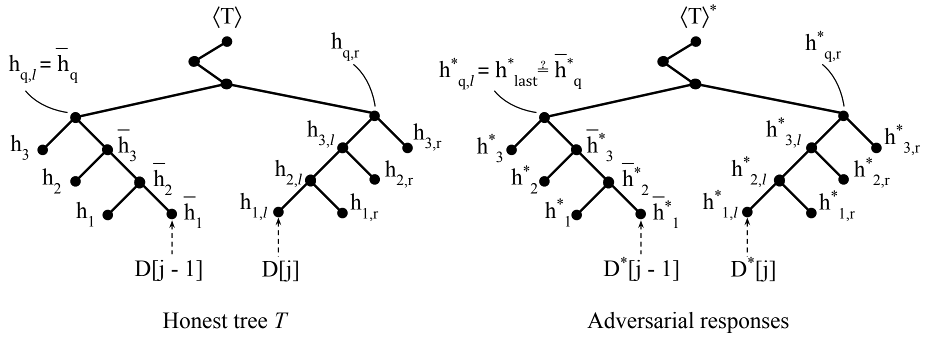

Fig. 6: The world in the view of the proof of Lemma 4. Starred quantities (right-hand side) denote adversarially provided values. Unstarred quantities (left-hand side) denote the respective honestly provided values. The inner node at height q from the leaves is the level containing the lowest common 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 4 (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 $(0, \perp, \perp)$. 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 $\perp$ 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 2.

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 2, $H(h^*_{a-1,l} \,\|\, h^*_{a-1,r}) = \langle\mathcal{T}\rangle^*$. 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 $h^* = h^*_{i,r}$, $i \in \{2, \ldots, a-1\}$, 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^*, $h^* = H(h^*_{i-1,l} \,\|\, h^*_{i-1,r})$ 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 2. 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 2, 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 $\Pr[D[j-1] \neq D^*[j-1] \mid \textsc{Bad}] \leq \Pr[\textsc{Discrepancy}] + \Pr[\textsc{Invalid-Proof}]$ 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,

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

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 5 (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 2. 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 $\langle \delta \rangle (\langle \mathsf{st} \rangle^*_{j-1}, \mathsf{tx}^*_j, \pi) = \langle \mathsf{st} \rangle^*_j$. This implies $\langle \delta \rangle (\langle \mathsf{st} \rangle_{j-1}, \mathsf{tx}_j, \pi) = \langle \mathsf{st} \rangle^*_j \neq \langle \delta(\mathsf{st}_{j-1}, \mathsf{tx}_j) \rangle$.

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 $O(\text{poly}(\ell))$ 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:

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

Moreover, by Lemma 4;

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

Consequenty, Pr[Responder wins] = negl(λ). □

Lemma 1 (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 2, 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, $(\mathsf{tx}_j, \langle\mathsf{st}\rangle_j)$, $j \in \{\ell_1, \dots, \min(\ell_2, \ell_1 + \alpha(u+\nu))\}$, 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 [32,10] place f in $\mathcal{O}(\operatorname{poly}\log r)$, and g is in $\mathcal{O}(\operatorname{poly}\log r)$ if standard Merkle constructions [2] 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 $\mathcal{O}(\operatorname{poly}\log r)$ and, hence, *succinct*.

Theorem 1 (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 3, 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 6 (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 $\mathbb{L}^{\cup}_{r+\nu}$, and outputs $(\mathsf{tx}_{p-1}, \mathsf{tx}_p, r)$. If $\beta < \alpha(u+\nu)$, $\mathcal{A}'$ outputs Failure.

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

Let $\ell = |\mathbb{L}^{\mathcal{P}}_r|$. By ledger safety, $\mathsf{tx}_0 = \mathbb{L}^{\mathcal{P}}_r[\ell-1] = \mathbb{L}^{\cup}_{r+\nu}[\ell-1]$. Hence, if $(\mathsf{tx}_0, \mathsf{tx}_1) \mid \mathbb{L}^{\cup}_{r+\nu}$, $\mathsf{tx}_1 = \mathbb{L}^{\cup}_{r+\nu}[\ell]$ as every transaction on $\mathbb{L}^{\cup}_{r+\nu}$ is unique. By induction, either there exists an index $i \in [\beta]$ such that tx_i does not immediately follow tx_{i-1} on $\mathbb{L}^{\cup}_{r+\nu}$, or $|\mathbb{L}^{\cup}_{r+\nu}| \geq \ell + \beta$ and $\mathsf{tx}_i = \mathbb{L}^{\cup}_{r+\nu}[\ell+i-1]$ 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 $\mathbb{L}^{\cup}_{r+\nu}$. 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 2 (Soundness). *Let H^s be a 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 5, 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 5, it returns *true* on all $(\mathsf{tx}_{j-1}, \mathsf{tx}_j)$, $j \in \{\ell^*, \ldots, \min(\ell, \ell^* \alpha(u+\nu)) - 1\}$. 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 7, 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 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 $\langle\mathcal{T}\rangle = \textsc{MakeMT}(D).\text{root}$, $D[\text{idx}] \neq v$, and $\textsc{Verify}(\pi, \langle\mathcal{T}\rangle, \text{idx}, v) = 1$. Conditioned on the fact that the challenge game reaches the suffix monologue, by Proposition 1, $\Pr[\neg\textsc{Equal}] = \Pr[\mathcal{A}' \text{ succeeds}] = \text{negl}(\lambda)$. Thus, the honest challenger wins the challenge protocol with overwhelming probability. □

Theorem 5 (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 6, $\overline{\mathcal{P}}$ 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 1, 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 7. *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 2, 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 3, $\mathcal{P}^*$ 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 9, 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 $i = \ell+1, \ldots, \min(\ell^*, \ell+\alpha(u+\nu)) - 1$. 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 $i \in \{\ell+1, \ldots, \min(\ell^*, \ell+\alpha(u+\nu)) - 1\}$. Similarly, for all $i \in \{\ell, \ldots, \min(\ell^*, \ell+\alpha(u+\nu))\}$, 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 $\langle\delta\rangle(\langle\mathsf{st}\rangle^*_{i-1}, \mathsf{tx}^*_i, \pi_{i-\ell+1}) = \langle\mathsf{st}\rangle^*_i$ for $i \in \{\ell+1, \ldots, \min(\ell^*, \ell+\alpha(u+\nu)) - 1\}$

Let D denote the sequence $\mathsf{tx}_{\ell-1}, \mathsf{tx}^*_\ell, \ldots, \mathsf{tx}^*_{\min(\ell^*, \ell+\alpha(u+\nu))-1}$ of transactions. Consider the event CONSENSUS-ORACLE that WIN holds, $\ell^* < \ell+\alpha(u+\nu)$, and there exists an index $i \in \{1, \ldots, \ell^*-\ell\}$ such that $D[i]$ does not immediately follow $D[i-1]$ on $\mathbb{L}^{\cup}_{r+\nu}$. 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 $\mathbb{L}^{\cup}_{r+\nu}$ 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 of state commitments $\langle\mathsf{st}\rangle_{\ell-1}, \langle\mathsf{st}\rangle^*_\ell, \ldots, \langle\mathsf{st}\rangle^*_{\min(\ell^*, \ell+\alpha(u+\nu))-1}$. Define $\mathsf{st}_i = \delta^*(st_0, \mathbb{L}^{\mathcal{P}}_r || (\mathsf{tx}^*_\ell, \ldots, \mathsf{tx}^*_{\ell+i-1}))$ for $i \in$

$\{1,\ldots,\ell^*-\ell\}$ $(\mathsf{st}_0 = \delta^*(st_0, \mathbb{L}_r^{\mathcal{P}}))$. 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 $\mathbb{L}_r^{\mathcal{P}}$, $\mathcal{A}_2$ finds st_i for all $i \in \{0,1,\ldots,\ell^*-\ell\}$ in $O(\text{poly}(|\mathbb{L}_r^{\mathcal{P}}|)$ 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 $i \in \{0,1,\ldots,\min(\ell^*-\ell, \alpha(u+\nu))-1\}$, Execution-Oracle implies that $\langle \delta(\mathsf{st}_{p-1}, D[p]) \rangle = \langle \mathsf{st}_p \rangle \neq S[p] = \langle \delta \rangle\,(S[p-1], D[p], \pi_p) = \langle \delta \rangle\,(\langle \mathsf{st}_{p-1} \rangle, D[p], \pi_p)$. Hence, Execution-Oracle implies that $\mathcal{A}_2$ 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 9 (Here, $\mathbb{L} = \mathbb{L}_r^{\mathcal{P}} \parallel (\mathsf{tx}^*_{\ell}, \ldots, \mathsf{tx}^*_{\ell^*-1}) \preceq \mathbb{L}^{\cup}_{r+\nu}$ and $\langle \mathsf{st} \rangle^*_{\ell^*-1} = \langle \delta^*(\mathsf{st}_0, \mathbb{L}) \rangle = \langle \mathsf{st} \rangle$). However,

$$\begin{gathered}\Pr[\textsc{Consensus-Oracle} \vee \textsc{Execution-Oracle}] \leq \\ \Pr[\mathcal{A}_1 \text{ succeeds}] + \Pr[\mathcal{A}_2 \text{ succeeds}] \leq \text{negl}(\lambda)\,.\end{gathered}$$

Moreover, by Lemma 6, $\mathcal{P}^*$ cannot reveal $\alpha(u+\nu)$ or more entries and win the game except with negligible probability. Hence,

$$\begin{aligned}\Pr[\textsc{Win}] =& \text{negl}(\lambda) + \Pr[\textsc{Win} \wedge \neg\textsc{Consensus-Oracle} \\ & \wedge \neg\textsc{Execution-Oracle} \wedge \ell^* < \ell + \alpha(u+\nu)]\,.\end{aligned}$$

which implies that either $\Pr[\textsc{Win}] = \text{negl}(\lambda)$ 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 6 (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 1 and 2, $\mathcal{P}_{i^*}$ 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 9. In the latter case, $\overline{\mathcal{P}}$ must have played the challenge game with $\mathcal{P}_{i^*}$. Then, by Lemma 7, the state commitment of $\overline{\mathcal{P}}$ satisfies safety and liveness per Definition 9 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. □

B Consensus Oracle Constructions

Consensus oracle constructions for Celestia (LazyLedger) [1], Prism [3,4], and Snap-and-Chat [40,39] follow the same paradigm described in Section 5.2.

B.1 Celestia

Celestia uses Tendermint [9] 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 5.2 can be used to provide a consensus oracle for Celestia.

B.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 5.2 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.

B.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 [22], PHANTOM [43] / GHOSTDAG [44], 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.

B.4 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 [23] 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 8 (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 9 (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 [32]. 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* [32] 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.

C 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 size of the confirmed ledger, light client in this case would have to download and process linear number of transactions in the length of the ledger.

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