Mining in Logarithmic Space

Aggelos Kiayias[1], Nikos Leonardos[2], and Dionysis Zindros[2]

[1] University of Edinburgh, IOHK
[2] University of Athens

Abstract. Blockchains maintain two types of data: Application data and consensus data. Towards long-term blockchain scalability, both of these must be pruned. While a large body of literature has explored the pruning of application data (UTXOs, account balances, and contract state), little has been said about the permanent pruning of consensus data (block headers). We present a protocol which allows *pruning* the blockchain by garbage collecting old blocks as they become unnecessary. These blocks can simply be discarded and are no longer stored by *any* miner. We show that *all* miners can be light miners with no harm to security. Our protocol is based on the notion of *superblocks*, blocks that have achieved an unusually high difficulty. We leverage them to represent underlying proof-of-work without ever illustrating it, storing it, or transmitting it. After our pruning is applied, the storage and communication requirements for consensus data is reduced exponentially.
We develop new probabilistic mathematical methods to analyze our protocol in the random oracle model. We prove our protocol is both secure and succinct under an uninterrupted honest majority assumption for 1/3 adversaries. Our protocol is the first to achieve *always secure*, *always succinct*, and *online* Non-Interactive Proofs of Proof-of-Work, all necessary components for a logarithmic space mining scheme. Our work has applications beyond mining and also constitutes an improvement in state-of-the-art superlight clients and cross-chain bridges.

1 Introduction

Will blockchain [39] systems handle the whole world's economic data for the centuries to come? While such lofty visions are often ubiquitous in the cryptocurrency space, it is a practical reality that today's blockchain technology simply does not scale [2]. One aspect of scalability difficulty stems from the data required to be stored and sent over the network when blockchain nodes synchronize with each other or bootstrap from the network for the first time.

These data contains two pieces of information: First, the *application data*. This includes transactions, account balances, and smart contract [12, 45] state evolution, and everything else that is included in the

block data itself. Secondly, the *consensus data*. This includes consensus-critical information such as proof-of-work [18] (or proof-of-stake) and nonces required to discover the longest chain among a sea of shorter forks — everything that is part of the *block header*. Nodes also need to reach consensus on the application data and ensure it follows the protocol rules for validity, but the application data is not what makes consensus happen. While application data can grow (or shrink) depending on the implementation, consensus data grows unboundedly at a constant linear rate in time. For example, in Bitcoin, while items can be added or removed from the UTXO [8], the number of block headers that need to be stored and communicated to newly bootstrapping nodes grows at a constant rate of 1 block header per 10 minutes in expectation [16]. Similarly, in Ethereum, while smart contracts can be added or destroyed [24], and smart contract state variables added or removed, block headers still grow at a constant rate of 1 block header per 12.5 seconds in expectation.

In the present paper, we focus on *proof-of-work* chains and *consensus data* (i.e., block headers) in particular. We put forth a mechanism to permanently *prune* the consensus data in a way that maintains the blockchain's security, without introducing any additional assumptions beyond honest computational majority. Our protocol compresses the amount of consensus data that needs to be stored and exchanged by nodes from linear to polylogarithmic — an exponential improvement. These reductions affect full nodes and miners alike, and, to our knowledge, are the first of their kind. Our protocol is the first to suggest that nodes need not hold onto chains at all; instead, full nodes and miners collectively only hold a small *sample* of blocks. The rest of the blocks are lost for ever, unless maintained by archival nodes, and are not necessary for achieving consensus or bootstrapping new nodes. We note here that our proposed scheme is not a sharding-based solution. *All* the miners of our protocol will store the *same* data. Sharding solutions can be *composed* with our solution in a per-shard basis to achieve even better scalability.

To achieve these reductions securely, we develop a mathematical framework for the analysis of blockchain systems under *suppression attacks* in which an adversary attempts to silence the generation of selected blocks. For our system to work correctly, it is imperative that the adversary faces difficulty in suppressing our high-value sample blocks, which we call *superblocks*. These represent the compression of proof-of-work. We prove that, in the random oracle model [6], these blocks cannot be silenced by any minority mining adversary. Our framework is an extension built on

top of the blockchain *backbone model* [20, 21] and can be independently useful for analyzing other protocols.

Our contributions. In summary, our contributions in this paper are as follows:

1. We put forth a mechanism which provides exponential improvements in the *consensus* data stored and exchanged between full nodes and miners in proof-of-work settings. Our protocol requires the storage and exchange of only *polylogarithmic* data, even when a new miner is bootstrapping from genesis.
2. We develop a mathematical framework for the analysis of *suppression* attacks, and analyze the security of our protocol therein. Our protocol is secure under honest majority assumptions (a 1/3 adversary) in the random oracle model.

Related work. Our work focuses on compressing *consensus data*, i.e., the proof-of-work headers exchanged and stored. There has been significant work in compressing *application data* in a way that maintains consensus. Such examples include moving transactions and smart contract execution off-chain in Layer 2 constructions such as payment channels [3, 4, 30, 42] and networks, rollups of the optimistic [44] or zero-knowledge [7] kind, and sidechains [28, 33, 41, 41]. Other systems allow (quite successfully) compressing multiple transactions into fewer or smaller, such as in the case of EDRAX [14], bulletproofs [9], or Mimblewimble [40]. These systems do not compress consensus state; all proof-of-work headers must still be sent and stored, even though the actual *application data* is reduced. Any long-term scalability solution must include a compression of *both* application data and consensus data. Our protocol can be *composed* with any of these.

Similar techniques to our consensus compression techniques have been previously used to create superlight clients, wallets that can quickly synchronize with the rest of the network. Such techniques include superblock-based [29, 38] NIPoPoWs [27, 31, 32, 48] and FlyClient NIPoPoWs [10]. However, these still require that miners maintain the whole blockchain so that they can help light clients synchronize. They cannot be readily adapted to logarithmic space *mining* scenarios. Specifically, superblock NIPoPoWs in their previous form cannot be both *always secure* and *always succinct*, while FlyClient NIPoPoWs cannot be built on top of previous NIPoPoWs in an *online* fashion. All of these properties are required for logarithmic space mining. Our protocol is heavily inspired by these protocols and the core idea is based on superblock NIPoPoWs, albeit with

critical changes that ensure *security*, *succinctness*, and the proofs being *online*.

Lastly, CODA [36] has been suggested to compress both *consensus* and *application* data together, but the mechanism requires a trusted setup, has no treatment of security loss due to zero-knowledge recursivity, and may prove impractical in terms of proof sizes or generation times.

Structure. We present our construction in stages. First, we discuss how an existing miner can compress their full state. Next, we discuss how a newly booting miner can bootstrap from genesis using only the compressed state. Subsequently, we show how a miner with only the compressed state can mine new blocks, giving rise to both *light* and *full* miners. Finally, we assemble our complete protocol, in which *all* miners are *light* miners. These constructions are accompanied by high-level security arguments and building an intuitive understanding of why the protocol works. After the full construction has been presented, the formal security analysis in the random oracle and backbone model follows. This analysis part is also where our mathematical framework for the treatment of suppression attacks is put forth. We conclude by discussing the limitations and shortcomings of our protocol.

2 Consensus and Application Data

Blockchain systems maintain certain *application state*. This state can be used to, for example, determine who owns how much money. There are two primary ways of representing ownership in today's blockchains: A *UTXO*-based system, in which the application state is comprised of the *unspent transaction outputs* that remain available for spending; and an *accounts*-based system, in which the application state is comprised of *accounts and their balances*. The first one is used primarily by Bitcoin, while the second one is used by Ethereum.

The application state evolves over time when *transactions* are applied to it. A transaction is a state evolution operator applied on the application state. Given a previous application state and a transaction, a new application state can be computed. Each block in the chain contains multiple transactions in a particular order. As such, a block is itself a state evolution operator which applies multiple transactions in order. By applying a block to a previous application state, a new application state can be computed.

There are two schools of thought regarding what should be stored in a block. In the first school of thought, only transactions (deltas) are stored.

The application state at the end of the blockchain can be computed by starting at the *genesis* application state (an *empty* application state) and *traversing* the blockchain, applying the state evolution described by each block, in order, and arriving at the final application state. This is what Bitcoin does. The other school of thought stores both transactions and the state *after* these transactions have been applied, a so-called *snapshot*. In such systems, if one holds the longest chain, the application state at the end of the chain does not need to be computed by applying any deltas. Instead, a block near the end of the chain can simply be inspected and the application state within it extracted.

It is possible to apply either school of thought to either application state model. Bitcoin only keeps only deltas for a UTXO-based application state. However, nothing prevents Bitcoin from committing to the newly computed UTXO in every block [13, 17, 35], and in fact some Bitcoin forks have already done so. On the other hand, Ethereum keeps both deltas and snapshots in blocks. While the snapshots are not necessary, they are helpful. For the rest of this paper, *we assume a proof-of-work blockchain in which each block commits to an application state snapshot.* The exact application state format (UTXO, accounts, or something else) is irrelevant for our purposes.

In both schools of thought, it is imperative that the validity of the application data (deltas or snapshots) is verified before a block can be accepted as valid. For example, in a snapshotted system, miners must check that the snapshot committed to a block was obtained by applying the transactions to the previous snapshot.

Blocks in chains store the *application data* — transactions and snapshots — in their body. This data is organized into an authenticated data structure, such as a Merkle Tree [37], and placed into a *block header*, which contains the *consensus data*. The consensus data consist of the *commitment* x to the application data; a proof-of-work *nonce* ctr; and a reference s to the previous block. It may also contain additional metadata such as timestamps. These data are hashed together using a hash function H to obtain the blockid $H(ctr \,\|\, x \,\|\, s)$, which is used as the reference s' in the next block.

Let us now discuss how a bootstrapping node can synchronize with the rest of the network. A bootstrapping node is a node holding only the *genesis* block and booting for the first time. A wallet node is interested in the *current* application state that concerns it. For example, it is interested to learn which UTXOs it owns, or how much money is in its own accounts. The *custodial history* of how these assets came to belong to it

is irrelevant [17], beyond archival purposes, as long as it can be sure that the assets it holds correspond to the correct application state based on the history that took place. Inspecting or having access to this history itself is not important for consensus purposes. As such, this node can synchronize with the rest of the network using the *SPV* method [39]: It downloads only the block *headers* to determine which chain is the longest one. It then inspects a block near the end of the chain and extracts the balance from the Merkle tree leaf for its own accounts, or for its UTXOs. This is sufficient to know the assets that it owns. In case some nodes are interested in the history of the blockchain, this history can be maintained by special archival nodes or block explorers, but are not necessary for the maintenance of the security of the network.

A miner bootstrapping their node can function in a similar manner: Download only the block headers to determine the longest chain, then inspect a block near the end of the chain to obtain the application state snapshot. Contrary to a wallet node, the miner must obtain the whole application state so that it can validate new pending transactions as they arrive. As such, the miner downloads the *headers* for the whole chain, and the *full* blocks only for blocks near the end of the chain.

To be more precise, after the longest chain has been determined by comparing block header chain lengths, the k^{th} block from the end is inspected, its application state snapshot is extracted, and the deltas in next k blocks are applied. This is necessary because an adversary can place incorrect snapshots in the most recent k blocks of a blockchain (folklore wisdom suggests $k = 6$ for Bitcoin). While that blockchain will look valid and long to someone verifying only headers, it will have snapshots corresponding to an incorrect application of deltas. However, the adversary cannot modify blocks prior to that, due to the Common Prefix [21] property of blockchains.

Note here that the miner does not need to verify the veracity of all historical transactions: If we assume that the majority of the computational power was honest for the duration of history, this ensures that, at all times during the execution, the longest chain represented the correct history of the world (with the exception of up to k blocks towards the end). Under the honest majority assumption, this scheme is as secure as full mining (but see the Discussion section at the end for a more nuanced take on this argument under temporary dishonest majority). This is contrary to schemes such as SPV mining in which no snapshots are available.

Application data can grow or shrink. UTXOs can be created or deleted, accounts and smart contracts can be created, updated and destroyed. State variables within smart contracts can also be constructed or destructed. *How* the application data grows is application-dependent. Typically, the application data will increase as the execution continues. There are several attempts to optimize the size of these data [3, 4, 7, 14, 30, 42, 44]. In this paper, we do not focus on these.

Instead, we focus on the size of the *consensus data*, that of block headers $H(ctr \,\|\, x \,\|\, s)$. Contrary to the application data, these data increase at a constant linear rate, as block headers are added to the chain. No matter if channels or rollups are used, block headers must keep getting added to the chain. Fortunately, the headers are small. Nevertheless, no matter how much pruning is done on the application layer, the consensus data will keep growing. A system designed to survive for the centuries to come must provision for the scalability of this ever-growing part. Even the solutions above that only download block headers do not tackle that problem. The aim of this paper is to explore whether this part can be pruned. As we will see, it is possible to reduce the consensus data and neither store nor communicate all block headers.

A visualization of the comparison between application and consensus data is shown in Figure 1. The consensus data (horizontal) grows at an expected constant rate in time. The application data (vertical) may grow (or shrink) depending on the application, and optimizations or pruning methods can be applied on top of them.

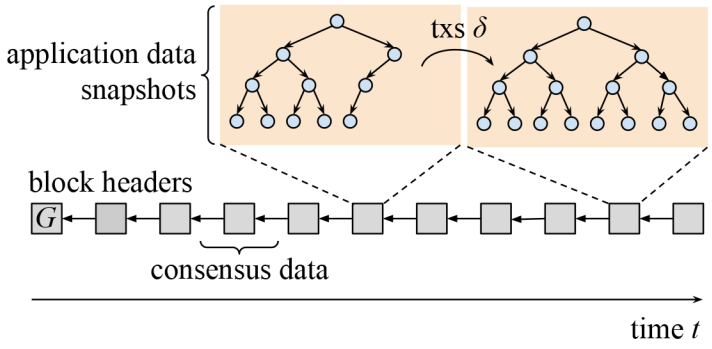


Fig. 1. A comparison of consensus data (growing horizontally with time) and application data (growing or shrinking vertically depending on the application).

A comparison of our paper against previous works is illustrated in Table 1. In all of these protocols, we have a node (the *prover*) that maintains all the necessary state to help a newly booting node (the *verifier* or *client*) synchronize with the rest of the network. We compare the storage requirements for the prover, as well as the communication complexity during bootstrapping. We are also interested in whether, after synchronizing with the rest of the network, the verifier can function as a fully-fledged miner on its own.

In this table, n denotes the number of blocks in the chain, δ is the size of the transactions in a single block (which may vary with time), a is the size of the snapshot or application state (which may also very with time), c is the size of a block header, and k is the common prefix parameter, the number of blocks required for stability (c.f., [26]). *BTC Full* indicates the full bitcoin miner that synchronizes by downloading all block headers and transactions $n(c + \delta)$. *BTC SPV* is a wallet-only client that downloads only block headers and a single transaction, but requires the prover (the node that serves it this data) to store the full history, as there are no snapshots available. *Ethereum* is a blockchain which uses block headers to synchronize, but makes use of snapshots. Here, the prover can prune block contents, but not block headers (the nc term remains). For the last k blocks, the transaction data of total size $k\delta$ are also needed to verify the veracity of the tip of the chain; for the k^{th} block from the end, only a snapshot of size a is needed. The client can start mining on top of these snapshots (after the $k\delta$ transaction data have been applied to the snapshot of size a). Note that $a \leq n\delta$ and $k \leq n$, and so (asymptotically) $n(c + \delta) \geq nc + k\delta + a$. *Superblock* and *FlyClient* NIPoPoWs allow a full node to function as a prover, only sending consensus data polylogarithmic in n, provided snapshots are available, but the receiving verifier cannot function as a miner or a prover for others. In this work, we present a protocol in which the verifier and prover are identical. The prover is only required to store polylogarithmic consensus data, and communication complexity is also polylogarithmic. This is indicated by the term $poly\log(n)c$. The term ka, the application data, remains unaffected and its pruning is orthogonal to this work.

3 State Compression

How can a newly booting miner synchronize with the rest of the network if block headers have been pruned? It seems impossible to do so securely. At first glance, the newly booting miner will be lost in a sea of appli-

Proposal	Storage	Communication	Can mine?
BTC Full	$n(c+\delta)$	$n(c+\delta)$	yes
BTC SPV	nc	nc	no
Ethereum	$nc+k\delta+a$	$nc+k\delta+a$	yes
Superblock NIPoPoWs	$nc+k\delta+a$	$poly\log(n)c+k\delta+a$	no
FlyClient NIPoPoWs	$nc+k\delta+a$	$poly\log(n)c+k\delta+a$	no
This work	$poly\log(n)c+k\delta+a$	$poly\log(n)c+k\delta+a$	yes

Table 1. A comparison of our results and previous work. n: the number of blocks in the chain; δ: size of transactions in a block; c: block header size; a: size of snapshot; k: common prefix parameter

cation snapshots and blockchain tips, without any ability to discern the application snapshot corresponding to the longest chain.

We approach this problem by *compressing* the consensus data. Among all the block headers that would be maintained by a traditional blockchain protocol, we only keep a small sample of block headers that are of interest. *Most* of the block headers headers will be pruned. The small sample of block headers that remains will be polylogarithmic in size and used as evidence that *work* took place throughout history. These sample block headers will be stored by our miners, and will also be sent to new bootstrapping miners when they boot. No other block headers will be stored or communicated beyond these carefully chosen samples. The samples will be chosen to be the same for all miners. As such, some block headers will survive throughout the network, while others will be gone for ever. Once we describe which block headers to keep and which ones to throw away, the construction of our prover will be complete. The rest of the work will be to construct a verifier that can distinguish between honest and adversarial application state claims by examining these samples and, of course, proving that this operation is secure.

Let us begin by discussing which samples among all block headers will be maintained by first presenting our *compression algorithm*: The code that can take in a full chain and perform the sampling. These block header samples will be the only ones that survive in our final protocol design. The compression algorithm takes in a full chain and produces the desired samples, but will not form part of our final protocol. In the final protocol, no full chain is to be found. However, the compression algorithm will prove educational in understanding the final protocol (and can also be used, once, to transition a full miner into a light miner). We will also reuse our compression algorithm in the final light miner construction, despite no full chains ever appearing.

We sample block headers based on their achieved proof-of-work. Recall that a block must satisfy the proof-of-work equation $H(ctr \,\|\, x \,\|\, s) \leq T$ for some constant[3] mining target T. Some blocks satisfy this equation much better than others and in particular may achieve $H(ctr \,\|\, x \,\|\, s) \leq \frac{T}{2^\mu}$ for some $\mu \in \mathbb{N}$. Following previous literature [27, 29, 31], we call these μ-superblocks and μ the *level* of a block. We model our hash function H as a random oracle with κ bits of output, and hence the distribution of μ superblocks, although stochastic, will be quite controlled. In particular, every block is a 0-superblock, about half the blocks in the chain will be 1-superblocks, about a quarter will be 2-superblocks, and in general, the probability that a valid block is a μ-superblock will be $\frac{1}{2^\mu}$. By definition, every block of level $\mu > 0$ is also a block of level $\mu - 1$, and all the levels below down to 0. The *genesis* block is, by convention, of *infinite* level. As the number of blocks per level drops exponentially as the level increases, the number of different levels will be approximately $\log |C|$, where $|C|$ denotes the size of the underlying blockchain [29].

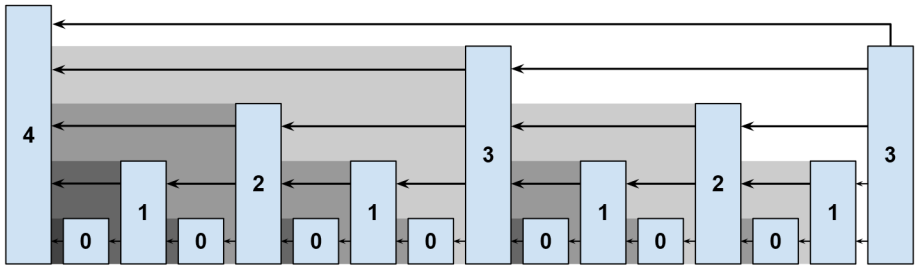
The intuition for our construction is as follows. Superblocks of increasing level become rarer and rarer. As such, superblocks can be used to illustrate that some *work* has occurred in a blockchain without actually delivering every block header. Consider a client that sees 13 superblocks of level 10. That client can readily deduce that approximately 2^{10} blocks must have appeared around each of these 13 superblocks. Otherwise, how was it possible to mine these blocks at this very high difficulty? The client can be sure that a total of about $13 \cdot 2^{10}$ blocks must *have been mined*, even though the client cannot observe these blocks directly (and, to be fair, if these blocks were mined by an adversary, they may have never been broadcast to the network at all).

While the client may see a series of blocks at a certain level, it cannot be sure that these were mined in the order they are presented. An adversary presenting 13 superblocks of level 10 may reorder them arbitrarily. For this reason, the same way we maintain previd pointers between blocks of level 0 in legacy blockchains, we need to maintain pointers between blocks at every level, and these pointers need to be placed within the proof-of-work so as to become immutable (as such, our protocol is not backwards-compatible with existing blockchains, but see the Discussion section for a note on this). As we cannot predict the level of a block before mining it, we will include $\log |C|$ pointers at every block: For each level μ, a pointer to the most for recent preceding block of level at least μ. This

[3] Throughout this paper we will assume the target T is constant; some notes around a variable target T appear in the Discussion section towards the end.

is not very different from a skiplist [43] and is akin to the construction of KLS [29]. There are efficient ways of storing such pointers [27]. This *interlinked* blockchain is illustrated in Figure 2.

Fig. 2. The interlinked blockchain. Each superblock is drawn taller according to its level. A new block links to all previous blocks that have not been overshadowed by higher levels in the meantime.



Our sampling will be performed by *only keeping sufficiently high-level superblocks* and throwing away blocks of low levels. We will keep very high levels (so, very few blocks) near genesis and far back in history. As we get closer to the present, we will start including more and more samples, and so the threshold in our superblock level will decrease. Near the tip (the most recent block) of the blockchain, we will eventually get down to level 0 and keep all blocks.

The samples that we keep will *evolve* as the blockchain grows. A sample that was once selected for inclusion may be thrown away later. However, any sample that is thrown away at some point will never again be needed in the future. This property, of ensuring that the sampling is safe and that no samples discarded will be needed again in the future, is the *online* property of our protocol. It will eventually allow us to build a protocol where no full chain is needed, anywhere.

Our algorithm is parametrized by a security (or, inversely, compression) parameter m and the common prefix parameter k (these can be unified by conservatively setting $m = 3k$). Given a chain $\mathcal{C}$ that we wish to compress, first, we keep the most recent k blocks aside, and let us call them χ. These are *unstable* and will need to always be stored. Besides, any miner that wishes to synchronize with us will need to look at them to arrive at a valid snapshot. For the next part, we only consider the *stable* part of the chain. For our sampling process, we begin by *the highest* level

ℓ that has *at least* $2m$ blocks in it. We will include this level in earnest: All ℓ-superblocks will be included in our sampling. For every level below ℓ, we will include at least the $2m$ most recently generated blocks of that level, but occassionally more. To consider whether to include more blocks than $2m$ blocks in a level μ, we look at the m^{th} most recent block b in the level $\mu + 1$ *immediately above*. We include all μ-superblocks that are *more recent* than block b. Let us make this description more precise by writing it out in pseudocode.

Notation. We will need some notation to describe our *chain compression* algorithm. Let $\mathcal{C}$ denote an interlinked chain of blocks and $\mathcal{C}[i]$ denote its i^{th} (zero-based) element. We denote by $\mathcal{C}[i{:}j]$ the blocks from the i^{th} (inclusive) to the j^{th} block (exclusive). Skipping i means taking the chain from the beginning, and skipping j means taking the chain to the end. If i and j are replaced by *blocks* A and Z instead of block *indices*, we write $\mathcal{C}\{A{:}Z\}$ to designate the blocks of $\mathcal{C}$ from block A (inclusive) to block Z (exclusive), and again any end can be omitted. A negative i or j means to take blocks from the end instead of from the beginning, so $\mathcal{C}[-1]$ is the tip. We write $A \in \mathcal{C}$ to mean that the block A is in the chain $\mathcal{C}$, and $\mathcal{C}_1 \subseteq \mathcal{C}_2$ to mean that all of $\mathcal{C}_1$'s blocks are in $\mathcal{C}_2$. We write $\mathcal{C}{\uparrow}^{\mu}$ to mean the subsequence of $\mathcal{C}$ containing only its μ-superblocks (by the above definition, the $\mathcal{C}{\uparrow}$ operator is absolute: $(\mathcal{C}{\uparrow}^{\mu}){\uparrow}^{\mu+i} = \mathcal{C}{\uparrow}^{\mu+i}$). Because $\mathcal{C}$ is interlinked, $\mathcal{C}{\uparrow}^{\mu}$ will be a chain, too. Given two chains $\mathcal{C}_1$ and $\mathcal{C}_2$ we write $\mathcal{C}_1 \cup \mathcal{C}_2$ to denote the chain consisting of all blocks in either, and $\mathcal{C}_1 \cap \mathcal{C}_2$ to mean the chain consisting of blocks only in both. Similarly we denote $\mathcal{C}_1 \setminus \mathcal{C}_2$ the chain consisting of blocks in $\mathcal{C}_1$ but not in $\mathcal{C}_2$. The blocks must be ordered chronologically and interlink pointers must be checked to ensure that the union, intersection, and subtraction of chains is a chain — they will not always be. The chain filtering operators $\uparrow$, $[\cdot]$, and $\{\cdot\}$ have a higher precedence than $\cup, \cap, \setminus$.

Our chain compression algorithm $\mathsf{Compress}_{m,k}(\mathcal{C})$ is illustrated in Algorithm 1. It uses the helper function $\mathsf{Dissolve}_{m,k}(\mathcal{C})$ to obtain the highest level ℓ, the unstable suffix χ and a set $\mathcal{D}[\mu]$ of blocks sampled from the stable part of the chain at each level $\mu \leq \ell$. All of these levels are combined into a big chain π, which is sparse at the beginning and dense towards the end. The final compressed state consists of π, the stable part, and χ, the unstable part. Together, these form a chain. Let us now examine the inner workings of $\mathsf{Dissolve}_{m,k}(\mathcal{C})$. This function separates the stable part $\mathcal{C}^*$ of the chain and the unstable part χ. In the trivial case that our stable chain has no more than $2m$ blocks, all of them are included. Otherwise, the highest level ℓ with at least $2m$ blocks is extracted and

included in earnest. Then, the levels are traversed downwards. For every level μ, the last $2m$ blocks are always included. This is captured by the term $\mathcal{C}^*{\uparrow}^\mu\ [-2m{:}]$. Additionally, we look at the m^{th} most recent block b from the end at level $\mu+1$, that is $\mathcal{C}^*{\uparrow}^{\mu+1}\ [-m]$. For level μ, we also include all the blocks succeeding b, that is $\mathcal{C}^*{\uparrow}^\mu\ \{b{:}\}$.

Algorithm 1 Chain compression algorithm for transitioning a full miner to a logspace miner. Given a full chain, it compresses it into logspace state.

function $\mathsf{Dissolve}_{m,k}(\mathcal{C})$
  $\mathcal{C}^* \leftarrow \mathcal{C}[:-k]$
  $\mathcal{D} \leftarrow \emptyset$
  if $|\mathcal{C}^*| \geq 2m$ **then**
    $\ell \leftarrow \max\{\mu : |\mathcal{C}^*{\uparrow}^\mu| \geq 2m\}$
    $\mathcal{D}[\ell] \leftarrow \mathcal{C}^*{\uparrow}^\ell$
    for $\mu \leftarrow \ell - 1$ down to 0 **do**
      $b \leftarrow \mathcal{C}^*{\uparrow}^{\mu+1}\ [-m]$
      $\mathcal{D}[\mu] \leftarrow \mathcal{C}^*{\uparrow}^\mu\ [-2m{:}] \cup \mathcal{C}^*{\uparrow}^\mu\ \{b{:}\}$
    end for
  else
    $\mathcal{D}[0] \leftarrow \mathcal{C}^*$
  end if
  $\chi \leftarrow \mathcal{C}[-k{:}]$
  return $(\mathcal{D}, \ell, \chi)$
end function
function $\mathsf{Compress}_{m,k}(\mathcal{C})$
  $(\mathcal{D}, \ell, \chi) \leftarrow \mathsf{Dissolve}_{m,k}(\mathcal{C})$
  $\pi \leftarrow \bigcup_{\mu=0}^{\ell} \mathcal{D}[\mu]$
  return $\pi\chi$
end function

It may not yet be clear why this selection of block headers will lead to a secure protocol, but let us argue that this sampling is polylogarithmic in $|\mathcal{C}|$, considering that m and k are constants that do not grow as the execution progresses.

Theorem 1 (Succinctness). *The construction of Algorithm 1 samples a polylogarithmic number of blocks with respect to the length of the chain $\mathcal{C}$.*

Proof (Sketch). Firstly, the number ℓ of levels of interest is $\Theta(\log|\mathcal{C}|)$. Next, each level μ has either $2m$ blocks or more. $2m$ is a constant, so this is irrelevant. But the *more* blocks cannot be many more either: We are

counting the μ-superblocks following the m^{th} block b at the level $\mu + 1$ above. How many can these be? They are indeed about $2m$. For, suppose for contradiction that they were many more than $2m$. But every block of level μ has a $\frac{1}{2}$ probability of also being a $\mu + 1$ level block. If there were, say, $4m$ instead of $2m$ superblocks of level μ following block b, then b would not be the m^{th} block from the end, but the $2m^{\text{th}}$ one! With high probability (with foresight, utilizing a Chernoff bound), $4m$ can be taken as an upper bound. As such, there will be $2m \log(|\mathcal{C}|) + k$ blocks sampled in expectation, and, with high probability, not many more. □

We make this bound and argument more precise in the Analysis section.

4 Fast Synchronization

We have seen how a full miner can compress their state into a polylogarithmic *sample* $\pi\chi$ of blocks. But what is the use of this? We will now build the other side of the protocol: A node, and future miner, booting to the network for the first time, but holding only genesis $\mathcal{G}$. The node is also parametrized by the security parameters m and k. This node wishes to learn where to mine.

For now, let us assume that the rest of the network consists of full miners, and only one node is a light node. The first step of the neophyte is to determine *what the current tip and snapshot* are. The light miner can then start mining on top of that tip, extending its application data snapshot. It does not need to know the blocks preceding the tip! Of course, this node will not be helpful towards bootstrapping *yet more* nodes, but no matter — it can still mine as if it were a full miner, and just as securely, as long as the tip can be correctly discerned.

The protocol works as follows. Initially, the newly booting node, which we call a *verifier* in this context, connects to multiple full nodes, which we call the *provers*. We assume at least one of the provers is honest (this is a standard assumption in the analysis of all blockchain protocols [20–23,46]). Each of these full nodes *compresses* their state using Algorithm 1 and sends the compressed state, or *proof* $\Pi = \pi\chi$, to the verifier. More concretely, the full node sends the block headers corresponding to the blocks in π (of size $c \cdot \mathit{poly}\log(n)$). For the blocks in χ, the full node sends the whole application snapshot (of size a) stored in $\chi[-k]$ and the transactions (of size $k\delta$) stored in χ. Naturally, the adversary can send any string as a claimed proof. The verifier checks that Π forms a chain, i.e., that all blocks are connected with interlinks and so they have

been produced in the chronological order presented, and also that the first block in Π is the genesis block $\mathcal{G}$ that it knows. It then extracts the last k blocks as χ and the rest as π. It inspects the application data snapshot from $\chi[-k]$ and ensures that the transactions in χ can be cleanly applied. This allows it to obtain the application state at the end of $\pi\chi$, which, in honest cases, is the same as the application snapshot at the end of the underlying blockchain. If any of these checks fail, the particular connection is considered compromised and closed.

The verifier receives and verifies a series of such proofs, each consisting of a stable part π and an unstable part χ, with $|\chi| = k$. Given multiple such proofs $\Pi_1, \Pi_2, \cdots, \Pi_v$, the prover begins inspecting the proofs and comparing one against the other in a pairwise fashion. First, Π_1 is compared against Π_2, and one of them is deemed to be the best (using a mechanism we will soon study). The process continues until only one of them remains. As long as at least one proof was honestly generated, our protocol will arrive at a suffix χ that is *admissible*. This means that our light node will arrive at a snapshot which a *full node miner* booting for the first time from genesis *could* also have arrived at. Upon taking this decision, the light miner stores $\pi\chi$ in its state.

The light miner can then start mining on top of $\chi[-1]$ to produce further blocks and to fully verify the validity of incoming network transactions in its mempool. After all, it is holding onto an application snapshot. These blocks can be broadcast to the network and will be accepted by the rest of the miners, despite our light miner not holding the full chain leading from genesis up to the newly mined block. The light miner can also understand and verify newly mined blocks of others. It can also deal with chain reorganizations: In case a reorganization of up to k blocks occurs, the light miner holds the whole of χ and can verify the state transitions completely. As for reorganizations of more than k blocks long, these will never occur (except with negligible probability) due to the Common Prefix property [21].

As this miner is not interested in helping bootstrap others, it can even throw away π once it has booted up. Furthermore, every time a new block is mined (either by itself or by someone else), it can append it to χ and then truncate χ to only keep the k most recent blocks. However, in the full protocol, described in the next section, the miner will need to hold on to (and update) π to allow others to bootstrap.

Let us now study the security-critical portion of our protocol, namely how the verifier compares two different proofs Π and Π'. Given two proofs Π and Π', the algorithm must decide which one is *best* or cap-

tures the most proof of work. In other words, it must conceptually correspond to the *longest underlying chain*, or the underlying chain with the most work. The comparison algorithm is illustrated in Algorithm 2. The comparison is performed as follows. Initially, the two proofs Π and Π' are verified for syntactic validity: That Π begins with $\mathcal{G}$, it is a chain, and that χ contains valid transactions extending the application data snapshot contained in $\chi[0]$. The comparison continues by invoking the $\mathsf{Dissolve}_{m,k}(\Pi)$ function of Algorithm 1 on each of Π and Π'. As before, this function extracts the maximum level ℓ containing at least $2m$ blocks. Then it picks the required blocks from each level, with at least $2m$ blocks per level, but also a sufficient number of blocks per level to *span* the last m blocks in the level above. Contrary to the invocation in Algorithm 1, we are not passing the full chain to the function; instead, we are passing a chain which has already undergone compression. As such, if the compressed state was honestly generated, the triplet $(\chi, \ell, \mathcal{D})$ on the verifier end will be the same as the triplet on the prover end, because $\mathsf{compress}_{m,k}(\mathcal{C}) = \mathsf{compress}_{m,k}(\mathsf{compress}_{m,k}(\mathcal{C}))$ (but may be something else in case of adversarial proofs).

Algorithm 2 The state comparison algorithm.

function $\mathsf{maxvalid}_{m,k}(\Pi, \Pi')$
  if Π is not valid **then**
    return Π'
  end if
  if Π' is not valid **then**
    return Π
  end if
  $(\chi, \ell, \mathcal{D}) \leftarrow \mathsf{Dissolve}_{m,k}(\Pi)$
  $(\chi', \ell', \mathcal{D}') \leftarrow \mathsf{Dissolve}_{m,k}(\Pi')$
  $M \leftarrow \{\mu \in \mathbb{N} : \mathcal{D}[\mu] \cap \mathcal{D}'[\mu] \neq \emptyset\}$
  if $M = \emptyset$ **then**
    if $\ell' > \ell$ **then**
      return Π'
    end if
    return Π
  end if
  $\mu \leftarrow \min M$
  $b \leftarrow (\mathcal{D}[\mu] \cap \mathcal{D}'[\mu])[-1]$
  if $|\mathcal{D}'[\mu]\{b:\}| > |\mathcal{D}[\mu]\{b:\}|$ **then**
    return Π'
  end if
  return Π
end function

Only once the two proofs are stratified into levels $\mathcal{D}$, the comparison algorithm attempts to choose a level μ at which the comparison will be performed. This level is the *minimum* level μ for which both provers have provided blocks (note that it is not sufficient that both provers have provided the same block at the same level; it must also have been selected in the same index of $\mathcal{D}$). In the edge case that *no* such level can be found, the prover with the higher ℓ wins (if no such level is found *and* they share the same level, it is irrelevant which prover will win). In the normal case that a level *is* found, then the comparison takes place by taking account *only* blocks of that level. The comparison begins by finding the most recent block shared by the two parties at that level, $(\mathcal{D}[\mu] \cap \mathcal{D}'[\mu])[-1]$. We call this the *lowest common ancestor* b. The blocks of the selected level *following* block b (which must necessarily be disjoint by the definition of b) are then counted, and the party with the most blocks wins.

Let us give a high-level intuition of why this protocol chooses the longest chain. The key idea is that, in addition to the Common Prefix property holding for regular blocks, this property also holds for μ-superblocks at any level. More precisely, if there is a forking point b, the adversary could not have produced more than m superblocks of level μ faster than the honest parties can produce m superblocks of level μ. This property stands at the heart of the following theorem.

Theorem 2 (Security). *When the honest verifier of Algorithm 2 receives a proof Π constructed by an honest party using Algorithm 1 and a proof Π' constructed by the adversary, it will decide in favour of the honest proof, unless the adversary is playing honestly and Π' was generated according to protocol.*

Proof (Sketch). First, consider the case that $M \neq \emptyset$. If the comparison is performed at level $\mu = 0$, this is akin to comparing traditional chains and the theorem holds due to the Common Prefix property.

If the comparison is performed at a level $\mu > 0$, then we apply the extended Common Prefix property at level μ. By the minimality of μ, there will be at least m blocks of the appropriate level following b and so the honest parties will win.

Lastly, if $M = \emptyset$, then we can apply the extended Common Prefix property at the highest level ℓ achieved by the honest party. By construction, the honest party holds at least $2m$ blocks at this level. Because the adversary must have achieved a better $\ell' > \ell$ to win, she must also have at least $2m$ blocks of a higher level, but these are also of level ℓ. But this

contradicts the extended Common Prefix property, giving us the desired result. □

While this gives some intuition about why the protocol is designed the way it is, the core security argument pertains to arguing why the extended Common Prefix property holds. We formally prove this statement in the Analysis section for 1/3 adversaries, where we also make the security theorem more precise.

5 Mining New Blocks

So far, we have used full nodes to help bootstrap newly booting miners. Can light miners be used to bootstrap newly booting miners instead? If we can achieve this, then we might as well get rid of full nodes altogether.

Our light miner already holds a valid proof $\Pi = \pi\chi$ corresponding to an underlying honest full node chain $\mathcal{C}$ at the time it is bootstrapped by others. Before further blocks are mined on the network (either by itself, or by others), it can send this Π to newly booting miners, and they, too, will be convinced of the current application data snapshot. The question is how to update this Π when a new block is mined. Suppose a new block b is mined on top of $\mathcal{C}$, either by our light miner or by someone else. The underlying honest chain then becomes $\mathcal{C}' = \mathcal{C}b$. Can we produce a proof Π' corresponding to $\mathcal{C}'$ by only utilizing Π? More specifically, given $\Pi = \mathsf{Compress}_{m,k}(\mathcal{C})$ and b, but not given $\mathcal{C}$, can we produce $\Pi' = \mathsf{Compress}_{m,k}(\mathcal{C}b)$? Indeed we can. In fact, it is as simple as evaluating $\mathcal{C}' = \mathsf{Compress}_{m,k}(\Pi b)$.

Theorem 3 (Online). *Consider $\Pi = \mathsf{Compress}_{m,k}(\mathcal{C})$ generated about an underlying honest chain $\mathcal{C}$, and a block b mined on top of $\mathcal{C}$. Then $\mathsf{Compress}_{m,k}(\mathcal{C}b) = \mathsf{Compress}_{m,k}(\Pi b)$.*

Proof. Consider which blocks are sampled and which blocks are pruned by $\mathsf{Compress}_{m,k}(\mathcal{C}b)$. Clearly the block b will be included in both the results of $\mathsf{Compress}_{m,k}(\mathcal{C}b)$ and $\mathsf{Compress}_{m,k}(\Pi b)$. All the other blocks selected by $\mathsf{Compress}_{m,k}(\mathcal{C}b)$ will already exist in Π, and in the correct positions. For, the blocks selected from a level are the last $2m$ of a level, or the last m spanning the level above, and adding block b at the end can only render a previously sampled block irrelevant, but not add further block requirements from the past. □

Note also that, when mining a new block b, all the data required to compute the interlink pointers of b is readily available in $\pi\chi$, as π contains

the most recent $2m$ blocks of every level, and only the most recent one is needed for interlinking.

Algorithm 3 The final logspace miner.

$\Pi \leftarrow \emptyset$
function $\mathsf{Init}_{m,k}(\overline{\Pi})$
  for $\Pi' \in \overline{\Pi}$ **do**
    $\Pi \leftarrow \mathsf{maxvalid}_{m,k}(\Pi', \Pi)$
  end for
end function
function $\mathsf{Mine}_{m,k}(\mathrm{x})$
  $b \leftarrow \mathsf{pow}(\Pi[-1], x)$
  if $b \neq \epsilon$ **then**
    $\Pi \leftarrow \mathsf{Compress}_{m,k}(\Pi b)$
    BROADCAST(Π)
  end if
end function
upon BOOTSTRAPREQUEST **do**
  return Π
end upon
upon NEWBLOCKRECEIVED(χ') **do**
  $\chi \leftarrow \Pi[-k:]$
  $\pi \leftarrow \Pi[:-k]$
  if χ' is a chain $\wedge\ \chi'[0] \in \chi$ **then**
    $b \leftarrow (\chi \cap \chi')[-1]$
    if $|\chi'\{b:\}| > |\chi\{b:\}|$ **then**
      Validate χ' state transitions starting from b
      $\Pi \leftarrow \mathsf{Compress}_{m,k}(\pi\chi\{:b\}\chi'\{b:\})$
      BROADCAST(Π)
    end if
  end if
end upon

Our final light miner therefore works as follows. It maintains a *current* proof $\Pi = \pi\chi$ and mines using $\chi[-1]$ as the chain tip. If it is successful in mining b on top of χ, it replaces Π by setting it to $\Pi' = \mathsf{Compress}_{m,k}(\Pi b)$ and broadcasts this to the network. As all of the other online miners, light or full, will hold their own χ^* not differing more than k blocks from χ, it is, in fact, sufficient that it broadcasts the new $\chi' = \chi[1:]b$ portion of Π'. Now the newly computed Π corresponds to the chain $\mathcal{C}b$, which the miner never sees, as it has been pruned. Regardless, Π can be used to bootstrap new light miners from genesis.

Consider now the case that our light miner holds a $\Pi = \pi\chi$ and a different miner mines a new block b. By the Common Prefix property,

this block will not deviate more than k blocks from the χ that our light miner already holds. Typically, it will be just a block on top of χ, but occassionally it could correspond to a chain reorganization up to k blocks long. In the case of a reorganization, the light miner requests the last k blocks χ' on top of which b was mined. These can be provided to us if the block b was mined by a light or a full miner, as both hold and can send χ'. The blocks in χ' will intersect the previously known χ at some fork point. The light miner checks that the transactions included in this χ' can be applied to the application data snapshot that the light miner has independently calculated for the fork point. This amounts to full block validation. The light miner also checks that the newly mined block really does correspond to a longer chain and that a reorganization is warranted by ensuring that there are more blocks in χ' after the fork point b than there are in χ after the fork point (i.e., that $|\chi'\{b:\}| > |\chi\{b:\}|$). It then replaces the stored proof by setting $\mathit{\Pi}$ to be the proof corresponding to $\pi\chi$ when the portion of χ after the most recent common block between χ and χ' is replaced by the blocks in χ', i.e., it updates its stored proof to be $\mathit{\Pi}' = \mathsf{Compress}_{m,k}(\pi\chi\{:b\}\chi'\{b:\})$.

The light miner is illustrated in Algorithm 3. At this point, full nodes are no longer necessary. Light miners can bootstrap from genesis. They have all the data needed to mine on their own, and to validate newly mined blocks from the network. If a newly booting light miner wishes to synchronize with the network, they have sufficient data to help them do so. The blockchain protocol remains exactly the same as in traditional blockchains, but all the instances of *chains* are replaced by *proofs* instead. Light miners mine on top of their current proof instead of mining on top of a chain. When they discover a new block, they send the newly computed proof instead of a chain. This concludes our construction.

6 Block Suppression

We begin our analysis by developing a probabilistic framework to study whether the adversary can *suppress* blocks of her choice. The central definition here is the notion of a Q-block, a block that possesses a certain property — such as being a μ-superblock for some $\mu \in \mathbb{N}$. The main theorem we will eventually prove is a generalization of the Common Prefix property: That the Common Prefix property holds for blocks filtered by any attribute Q. This will allow us to prove our protocol is secure by instantiating Q-blocks as μ-superblocks.

For our analysis, we work in the Backbone model [21] and adopt an environment where the network is synchronous and the protocol is executed in distinct *rounds*. We give a short overview of the model. Let κ denote the security parameter, and n denote the total number of parties, t of which are adversarial. Block generation takes place, by honest and adversarial parties alike, by invoking a shared Random Oracle $H(\cdot)$ whose range is $\{0,1\}^\kappa$. We use q to denote the number of random oracle queries available to each party per round. The honest parties search the space by performing q queries for $\mathsf{ctr} \leftarrow 1$ to q. The adversary controlling t parties gets qt total queries per round.

It has been proven [21, 22] that executions follow the properties of *Chain Growth*, *Common Prefix*, and *Chain Quality*.

We define a Q-block as a block satisfying a predicate Q on its hash. Note that this evaluation does not depend on any particular execution.

Definition 1 (Q-block). *A* block property *is a predicate Q defined on a hash output $h \in \{0,1\}^\kappa$. Given a block property Q, a valid block with hash h is called a Q-block if $Q(h)$ is true.*

The block properties we are interested in will be evaluated in actual executions as $Q(H(\langle ctr, s, x\rangle))$ for particular blocks. As such, we will be interested in properties which are polynomially computable given h as the input.

Definitions of random variables. We will call a query of a party *successful* if it submits a triple (ctr, s, x) such that $H(ctr, s, x) \leq T$. Consider a block property Q. Let $\xi_Q = \Pr[Q(h)|h \leq T]$, when h is uniformly distributed over the range of the hash function. For each round i, query $j \in [q]$, and $k \in [t]$ (the k^{th} party controlled by the adversary), we define Boolean random variables $X_Q(i), Y_Q(i)$ and $Z_Q(i,j,k)$ as follows. If at round i an honest party obtains a Q-block, then $X_Q(i) = 1$, otherwise $X_Q(i) = 0$. If at round i exactly one honest party obtains a Q-block, then $Y_Q(i) = 1$, otherwise $Y_Q(i) = 0$. Regarding the adversary, if at round i, the j^{th} query of the k^{th} corrupted party obtains a Q-block, then $Z_Q(i,j,k) = 1$, otherwise $Z_Q(i,j,k) = 0$. Define also $Z_Q(i) = \sum_{k=1}^{t}\sum_{j=1}^{q} Z_Q(i,j,k)$. For a set of rounds S, let $X_Q(S) = \sum_{r\in S} X_Q(r)$ and similarly define $Y_Q(S)$ and $Z_Q(S)$. We drop the subscript from all variables X, Y, Z, when the Q-block is simply the property of being a valid block. Further, if $X(i) = 1$, we call i a *successful round* and if $Y(i) = 1$, a *uniquely successful round*.

As in the backbone model [21], the probability f that at least one honest party computes a solution at given round is an important parameter. Writing $p = T/2^\kappa$ for the probability of success of a single query, we

have

$$(1-f)pq(n-t) \leq f = \mathbb{E}[X(i)] = 1-(1-p)^{q(n-t)} \leq pq(n-t).$$

The following bounds relate the expectations of the random variables defined above to f, for all i and block properties Q.

$$\xi_Q f \leq \mathbb{E}[X_Q(i)] \leq \frac{\xi_Q f}{1-f}, \quad \xi_Q f(1-f) < \mathbb{E}[Y_Q(i)],$$
$$\mathbb{E}[Z_Q(i)] \leq \frac{\xi_Q f}{1-f} \cdot \frac{t}{n-t}.$$

For the derivations of these inequalities see Garay et al. [20].

Typical executions. We now define our typical set of executions. This follows the backbone model, but extended to include block properties. Informally, this set consists of those executions with polynomially many rounds and with the property that all the random variables of interest over sufficiently many (at least $\lambda = \Omega(\kappa)$) consecutive rounds do not deviate too much from their expectation. To this end, recall the following terms [20]. An *insertion* occurs when, given a chain $\mathcal{C}$ with two consecutive blocks B and B', a block B^* created after B' is such that B, B^*, B' form three consecutive blocks of a valid chain. A *copy* occurs if the same block exists in two different positions. A *prediction* occurs when a block extends one which was computed at a later round.

Definition 2 (Typical execution). *For a real $\epsilon \in (f, \frac{1}{4})$, integer λ, and a collection of polynomially many block properties $\mathcal{Q}$, we say an execution is* $\mathcal{Q}$-typical *(or simply typical), if the following hold.*

- *For any $Q \in \mathcal{Q}$ and any set S of at least λ/ξ_Q consecutive rounds we have*

$$(1-\epsilon)\,\mathbb{E}[X_Q(S)] < X_Q(S) < (1+\epsilon)\,\mathbb{E}[X_Q(S)], \tag{1}$$
$$(1-\epsilon)\,\mathbb{E}[Y_Q(S)] < Y_Q(S), \tag{2}$$
$$Z_Q(S) < \mathbb{E}[Z_Q(S)] + \epsilon\,\mathbb{E}[Y_Q(S)]. \tag{3}$$

- *No insertions, no copies, and no predictions occurred.*

Theorem 4. *If $t < (1-\delta)(n-t)$ with $\delta > 3\epsilon + 3f$, an execution is typical with probability $1 - e^{-\Omega(\epsilon^2 f \lambda)}$.*

Proof. The proof uses standard Chernoff bounds, along the lines of [20]. We just note that the variables $X_Q(i)$ (and similarly $Y_Q(i)$ and $Z_Q(i,j,k)$) are independent Bernoulli trials for each Q and successful with probability $\Theta(\xi_Q f)$. In addition, a union bound is applied over all Q. □

Lemma 1. *Assume $t < (1-\delta)(n-t)$ with $\delta > 3\epsilon + 3f$ and a $\mathcal{Q}$-typical execution. Then, the following hold for any $Q \in \mathcal{Q}$ and any set S of at least λ/ξ_Q consecutive rounds.*

(a) $(1-\epsilon)\xi_Q f|S| < X_Q(S) < (1+\epsilon)\cdot\frac{\xi_Q f}{1-f}\cdot|S|$.
(b) $(1-\frac{\delta}{3})\xi_Q f|S| < (1-\epsilon)\xi_Q f(1-f)|S| < Y_Q(S)$.
(c) $Z_Q(S) < (\frac{t}{n-t}\cdot\frac{1}{1-f}+\epsilon)\cdot\xi_Q f|S| \leq (1-\frac{2\delta}{3})\xi_Q f|S|$.
(e) $Z_Q(S) < Y_Q(S)$.

Proof. This follows with straightforward calculations from the properties of a typical execution, the bounds on the expectations of the involved random variables, and the assumed bounds on $t/n, \delta, \epsilon$ and f. □

We now establish an upper bound in the number of Q-blocks an adversary can suppress, regardless of what attack method she follows.

Uniquely successful rounds have the following important property [20].

Lemma 2 (Pairing). *For any i and any pair of distinct blocks $\mathcal{C}[i]$ and $\mathcal{C}'[i]$, if $\mathcal{C}[i]$ was computed by an honest party in a uniquely successful round, then $\mathcal{C}'[i]$ was computed by the adversary.*

Proof. Let r be the uniquely successful round that $\mathcal{C}[i]$ was computed. No honest party would extend $\mathcal{C}'[i-1]$ at a round later than r, since every honest party would have a chain of length at least i. Similarly, if an honest party computed $\mathcal{C}'[i]$ at some round earlier than r, then no honest party would have extended $\mathcal{C}[i-1]$ at round r. Finally, $\mathcal{C}'[i]$ cannot have been computed by an honest party at round r, since r was a uniquely successful round. □

Lemma 3 (Suppression). *If r is a uniquely successful round and the corresponding block does not belong to the chain of an honest party at a later round, then there is a set of consecutive rounds S such that $r \in S$ and $Y(S) \leq Z(S)$.*

Proof. Let $\mathcal{C}$ be the chain of the honest party that was successful at round r and u the depth of the corresponding block. Let r' be the first round after r in which an honest party has a chain $\mathcal{C}'$ which does not contain

block $\mathcal{C}[u]$. Let $\mathcal{C}'[u']$ the last block of $\mathcal{C}'$ at round r'. Let $\mathcal{C}[u^*] = \mathcal{C}'[u^*]$ be the last honest block on the common prefix of $\mathcal{C}$ and $\mathcal{C}'$, and let r^* be its timestamp. We claim that the set $S = \{i : r^* < i < r'\}$ satisfies the requirements of the statement. Clearly, $r \in S$. Let us verify that $Y(S) \leq Z(S)$. Indeed, if $\mathcal{C}^*[v]$ is any block computed during a uniquely successful round $i \in S$, it must hold $u^* < v \leq u'$. The first inequality is because the party computing $\mathcal{C}^*[v]$ knows of $\mathcal{C}[u^*]$ (it was announced at round r^* and received by round $i > r^*$) and would not mine on a shorter chain. The second inequality holds because $v > u'$ contradicts an honest party having a chain of length u' at round $r' > i$ (since $\mathcal{C}^*[v]$ was received by round r'). The inequality then follows by Lemma 2, since it is always possible to find a block distinct from $\mathcal{C}^*[v]$ on $\mathcal{C}$ or $\mathcal{C}'$ (we may use $\mathcal{C}'$, unless $\mathcal{C}^*[v]$ is on $\mathcal{C}'$, in which case—due to the minimality of r'—we have $v < u$ and we can use $\mathcal{C}$). □

An observation that follows from the above lemma is that if the adversary manages to suppress a Q-block from the chain of an honest party and this Q-block was computed in a uniquely successful round, then we can associate with it an adversarial block. In particular, if r is a uniquely successful round and the corresponding block does not belong to the chain of an honest party at a later round, then there is an *associated* adversarial block of the same height that was adopted by an honest party.

We now state and prove our Unsuppressibility Lemma. Informally, the lemma says that if the number of blocks the adversary obtained in a set of consecutive rounds is z and the number of the uniquely successful blocks the honest parties obtained in the same set of rounds is y, then there exist $y - 2z$ blocks that will always belong to the chain of every honest party. It follows that if the power of the adversary is bounded below 1/3 of the total power, then with overwhelming probability there will be a nonzero number of such blocks.

An important note with respect to the Unsuppressibility Lemma is the following. Fix all the randomness the random oracle requires for a given execution. This determines the successful queries of every party and therefore determines the parameters y and z above. The observation is that even if these random coins are revealed to the adversary at the beginning of the execution, one can determine precisely which $y - 2z$ blocks —and no matter the adversary's strategy— will always belong to the chain of every honest party.

Lemma 4 (Unsuppressibility). *In a typical execution, every set of consecutive rounds U has a subset S of uniquely successful rounds, such that*

- $|S| \geq Y(U) - 2Z(U) - 2\lambda f(\frac{t}{n-t} \cdot \frac{1}{1-f} + \epsilon)$ *and*
- *after the last round in S the blocks corresponding to S belong to the chain of every honest party.*

Proof. Let U' be the set of consecutive rounds that contains U and also the λ rounds that come before and after U. By Lemma 3, we may take S to contain all those uniquely successful rounds $r \in U$ such that for any set of consecutive rounds $S' \subseteq U'$ containing r, $Y(S') > Z(S')$. Note that, in a typical execution, no such S' may contain elements outside U'. Letting $y = Y(U)$ and $z = Z(U)$, we need to show $y - |S| \leq 2z + 2(1 - \frac{2\delta}{3})\lambda f$.

Let us focus on the uniquely successful rounds not in S. Consider a collection $\mathcal{T}$ of sets of consecutive rounds with the following properties.

- For all $T \in \mathcal{T}$, $Y(T) \leq Z(T)$.
- For each $r \in U \setminus S$, there is a $T \in \mathcal{T}$ that contains r.
- $|\mathcal{T}|$ is minimum among all collections with the above properties.

We now observe that the minimality condition on $\mathcal{T}$ implies that no round r with $Z_r > 0$ belongs to more than two sets of $\mathcal{T}$. If that was the case, then there would be three sets T_1, T_2, T_3 in $\mathcal{T}$ with $T_1 \cap T_2 \cap T_3 \neq \emptyset$. But then, we could keep the two sets with the leftmost and rightmost endpoints, contradicting the minimality of $\mathcal{T}$. Furthermore, no round in $U' \setminus U$ belongs to more than one set of $\mathcal{T}$. Thus,

$$y - |S| = \sum_{i \in U \setminus S} Y_i \leq \sum_{T \in \mathcal{T}} Y(T) \leq \sum_{T \in \mathcal{T}} Z(T) \leq 2z + Z(U' \setminus U).$$

The third inequality holds because every round in which the adversary was successful is counted at most twice inside U and at most once outside U (by the discussion above the inequalities). Finally, using $|U' \setminus U| \leq 2\lambda$ and Lemma 1(c) we obtain the stated bound. □

The proof of this lemma is quite generous to the adversary on two accounts. First, it reveals to the adversary all coin flips in the beginning of the execution. Second, it gives the adversary two choices for each one of his blocks, and assumes that he will be able to choose among these as he sees fit. Nevertheless, we conjecture that the bound $y - 2z$ cannot be substantially increased in the case the property is rare.

We can now prove that an adversary with less than 1/3 of the total mining power cannot create a chain with more Q-blocks than an honest chain. Such a task would require the adversary to both suppress many Q-blocks from the honest chain *and* to obtain many of them for the adversarial chain.

Lemma 5 (Q-block Common-Prefix). *Assume $t < (\frac{1}{3} - \delta)n$ with $\delta > 3\epsilon + 3f$ and a $\mathcal{Q}$-typical execution. Consider a round at which a chain $\mathcal{C}$ is adopted by an honest party and suppose there exist another chain $\mathcal{C}'$ such that $\mathcal{C}' \setminus (\mathcal{C}' \cap \mathcal{C})$ has at least $22\lambda\xi_Q f$ Q-blocks. Then, with overwhelming probability, $\mathcal{C}$ has more Q-blocks than $\mathcal{C}'$.*

Proof. Assume an execution in which the assumptions of the lemma hold. Let r^* be the round on which the last honest block on $\mathcal{C}^* = \mathcal{C} \cap \mathcal{C}'$ was computed (if no such block exists let $r^* = 0$) and define the set of rounds $S = \{i : r^* < i \leq r\}$. We will study the execution during the rounds in S. To that end, let W' denote the set of adversarial queries on $\mathcal{C}' \setminus \mathcal{C}^*$ at some round at least λ greater from r^*. Denote by W the rest of the adversarial queries in S.

We first observe that no query in W' could have suppressed a Q-block on $\mathcal{C}$. As in the proof of Lemma 3, in such a case there would exist a set of consecutive rounds $|S^*| \geq \lambda$ such that $Y(S^*) \leq Z(S^*)$. This contradicts the last item of Lemma 1.

From this observation and the Unsuppressibility Lemma, there are at least $Y(S) - 2Z(W) - 2\lambda f(\frac{t}{n-t} \cdot \frac{1}{1-f} + \epsilon)$ blocks that the adversary cannot suppress. Each of these is a Q-block independently with probability ξ_Q. Under our assumptions, $2(\frac{t}{n-t} \cdot \frac{1}{1-f} + \epsilon) < \frac{1-\delta}{1-\epsilon}$. We conclude that, with overwhelming probability, there are at least

$$(1-\epsilon)\xi_Q \cdot \big[Y(S) - 2Z(W)\big] - (1-\epsilon)\lambda\xi_Q f$$

Q-blocks on $\mathcal{C} \setminus \mathcal{C}^*$.

On the other hand, the number of Q-blocks on $\mathcal{C}' \setminus \mathcal{C}^*$ is at most the Q-blocks from the W' queries plus the Q-blocks from the initial λ rounds. The latter can be shown to be at most $3\lambda\xi_Q f$. For the former, using Lemma 6 (with $F_j = 1$ when $j \in W'$ and $M_j = 1$ when it resulted in a Q-block) and Lemma 1, in a typical execution, are at most $(1+\epsilon)\xi_Q Z(W')$. Thus, there at most

$$(1+\epsilon)\xi_Q p|W'| + 3\lambda\xi_Q f.$$

Q-blocks on $\mathcal{C}' \setminus \mathcal{C}^*$. Since these are at least $22\lambda\xi_Q f$, it can be shown that the difference between the last two displayed expressions is at least

$(1-\epsilon)\xi_Q \cdot [Y(S) - 2Z(S)]$. This is positive in a typical execution in which the power of the adversary is bounded below $\frac{1}{3} - \delta$ the total power. □

7 Analysis

We are now ready to prove the construction of Algorithms 1 and 2 secure and succinct. For security, we denote Π the proof presented by the honest party and Π' the proof presented by the adversary (but these can be given to the verifier algorithm in any order). We can safely assume that these proofs were both generated at round r (the adversary could have generated the proof earlier, but not later, than the honest party). The honest proof Π was generated based on some honest underlying chain $\mathcal{C}$ using Algorithm 1. On the other hand, we have no guarantees about how the adversarial proof Π' was generated. It may be based on some underlying chain mined according to protocol, or not. In any case Π' *does* form a chain and its blocks must have been mined in order, as the verifier ensures this. However, there may not exist intermediate blocks covering the whole proof-of-work as desired.

Security mandates that the verifier chooses the honest proof, Π. However, it is possible that the verifier also chooses the adversarial proof, Π', without raising any issue, *as long as it extends the honest proof* at a fork point no longer than k from the tip. To see why this is fine, note that an adversary can already do this at the full blockchain: According to the Common Prefix property, she can fork at a block at most k blocks deep from the honest blockchain's end and have up to k blocks following the fork point. In this case, if it happens that the verifier has chosen Π', we require that $(\Pi' \cap \mathcal{C})[-1] \in \mathcal{C}[-k:]$. This means that the adversarial proof extends the honest chain at some fork point in $\mathcal{C}[-k:]$. But let us contemplate what this entails: It means that the portion $\Pi'\{(\Pi' \cap \mathcal{C})[-1]:\}$ is just a valid 0-level extension of the honest chain. As such, requiring $|\Pi'\{(\Pi' \cap \mathcal{C})[-1]:\}| \geq |\mathcal{C}\{(\Pi' \cap \mathcal{C})[-1]:\}|$ would produce a competitive adversarial chain that is *longer* than the honest chain and it would be perfectly acceptable to a full node (and by the common prefix property, this difference cannot be larger than k blocks long). We must clearly allow for this possibility — but it is not a problem, as this situation can occur in full node executions, too. This property also holds trivially in case the honest proof is chosen.

Theorem 5 (Security). *Consider an arbitrary $\frac{1}{3}$-bounded PPT adversary $\mathcal{A}$ in a typical execution. Let Π be a proof generated by an honest party at round r using Algorithm 1 by passing his underlying chain*

$\mathcal{C}$. Let Π' be an arbitrary proof generated by the adversary at round r. Let Π^* be the proof accepted by the verifier using Algorithm 2. Then $|\Pi^*\{(\Pi^* \cap \mathcal{C})[-1]:\}| \geq |\mathcal{C}\{(\Pi^* \cap \mathcal{C})[-1]:\}|$ with overwhelming probability.

Proof. Let $\mathcal{C}' = \Pi'$. We need to show that, either Π will be the proof accepted by the verifier, or Π^* is a proof extending the honest chain that is longer at level 0, as mandated by the theorem statement.

Let us consider first the case that a μ of Algorithm 2 as above exists. When $\mu = 0$, the verifier determines the longer chain and always correctly accepts the corresponding proof. That is, the verifier will either choose $\Pi^* = \Pi$, or, in case $\Pi^* = \Pi'$, the verifier will choose the adversarial proof Π' which contains a χ' that extends the honest chain's χ at level 0 (up to k blocks long) with a longer alternative. This is the only case in which Π' can win. For the other cases, we will now argue that the adversary cannot win.

Let us now focus on the case $0 < \mu \leq \ell$. Note that, since $\mathcal{D}[\mu - 1] \cap \mathcal{D}'[\mu - 1] = \emptyset$ (by the minimality of μ), *both* superchains must have at least m blocks after their common block b. The Q-block Common-Prefix Lemma implies that Π is accepted.

Next, consider the case that no such μ exists. Clearly, $\ell \neq \ell'$ (otherwise $\mathcal{D}[\ell] \cap \mathcal{D}'[\ell']$ would contain the genesis block) and we need to argue that $\ell > \ell'$. Assume —towards a contradiction— that $\ell < \ell'$ and consider the statement of the Q-block Common-Prefix Lemma instantiated with blocks of level $\ell + 1$ as the Q-blocks. Together with $\ell < \ell'$, it implies that $\mathcal{C}'$ has fewer than m Q-blocks after the common block with $\mathcal{C}$ (since $\mathcal{C}$ has fewer Q-blocks than $\mathcal{C}'$ in total, it must also have fewer on its fork; and they must necessary share a common block, since both must begin with genesis). But then, both $\mathcal{C}$ and $\mathcal{C}'$ have fewer than m Q-blocks after their common block. Since $\mathcal{D}[\ell] \cap \mathcal{D}'[\ell] = \emptyset$ by assumption, this cannot be the case. □

Theorem 6 (Succinctness). *In a typical execution with $t < (\frac{1}{3} - \delta)n$ with $3\epsilon + 3f < \delta < \frac{1}{3}$ and letting $m = \lambda$, an honest miner's state is in $O(m^2 \log(r))$ at round r.*

Proof. As $t < (\frac{1}{3} - \delta)n$ with $3\epsilon + 3f < \delta < \frac{1}{3}$, therefore $c = \mathbb{E}[Y] - 2\,\mathbb{E}[Z] - 2\lambda f(\frac{t}{n-t}\frac{1}{1-f} + \epsilon)$ will be a positive constant and for sets of consecutive rounds U with $|U| \geq \lambda$, we will have $Y(U) - 2Z(U) - 2\lambda f(\frac{t}{n-t}\frac{1}{1-f} + \epsilon) > (1 - \epsilon)c|U|$.

Consider a state Π generated by an honest prover and suppose for contradiction that $|\Pi| \in \omega(m \log(r))$, where r indicates the current round

number. From the security of the scheme, this state will correspond to some underlying chain $\mathcal{C}$ such that Π is the compression of $\mathcal{C}$. Consider the variables $(\mathcal{D}, \chi) = \mathsf{Dissolve}_{m,k}(\mathcal{C})$. As $|\chi| = k$ is constant, therefore $|\bigcup_{d\in\mathcal{D}} d| \in \omega(m\log(r))$. Let $\ell = |\mathcal{D}|$. It holds that $\ell \in O(\log(|\mathcal{C}|))$. Consequently, $\sum_{d\in\mathcal{D}} |d| \in \omega(m\log(r))$. Therefore there must exist some μ such that $|\mathcal{D}[\mu]| \in \Omega(\lambda^2)$. Consider the maximum such μ.

We distinguish two cases.

Case 1: $\mu = \ell$. Then consider $\mathcal{D}[\ell]$. Let u_0 denote the round during which $\mathcal{D}[\ell][0]$ was generated and u_1 denote the round during which $\mathcal{D}[\ell][-1]$ was generated and consider the set U of consecutive rounds from u_0 to u_1. As $\mathcal{D}[\ell]$ forms a chain, we have that $|U| \geq |\mathcal{D}[\ell]| > \lambda$. Applying the Unsuppressibility Lemma, we obtain that at least $|S| \geq c|U| = c|\mathcal{D}[\ell]| \in \Omega(\lambda)$ rounds of U must have been uniquely successful and belong to the chain of every honest party. Therefore $|\mathcal{D}[\ell]\uparrow^{\ell+1}| \geq (1-\epsilon)\frac{|S|}{2}$. By the definition of ℓ this is impossible.

Case 2: $0 \leq \mu < \ell$. By maximality of μ, we have $|\mathcal{D}[\mu+1]| \in O(\lambda)$, but $|\mathcal{D}[\mu]| \in \Omega(\lambda^2)$. By the definition of $\mathcal{D}[\mu] = \mathcal{C}[:-k]\uparrow^{\mu}\ [-2m:] \cup \mathcal{C}[:-k]\uparrow^{\mu}\{\mathcal{C}[:-k]\uparrow^{\mu+1}\ [-m]:\}$, clearly $|\mathcal{C}[:-k]\uparrow^{\mu}\ [-2m:]| = 2m$ so necessarily $\mathcal{C}[:-k]\uparrow^{\mu}\{\mathcal{D}[\mu+1][-m]:\} \in \Omega(\lambda^2)$. Therefore there exist blocks A and B in $\mathcal{D}[\mu+1]$ and $\mathcal{D}[\mu]$ such that $|\mathcal{D}[\mu+1]\{A{:}Z\}| = 1$, but $|\mathcal{D}[\mu]\{A{:}Z\}| \in \omega(\lambda)$. Similarly to case 1, consider the rounds u_0 and u_1 during which blocks A and Z were generated respectively and the set of consecutive rounds U from u_0 to u_1 with $|U| \in \omega(\lambda)$. Using the Unsuppressibility Lemma, there must exist a set of uniquely successful rounds $|S| \geq c|U|$ whose blocks have been adopted by all honest parties and of which at least $(1-\epsilon)\frac{|S|}{2} \geq 0$ will be of level $\mu+1$. Therefore there must exist a block between A and Z in $\mathcal{D}[\mu+1]$.

Both cases are contradictions. □

The previous theorem allows us to make miners reject incoming state that is too large (more than polylogarithmic) without processing them fully.

We note here that our analysis critically relies on the honest majority assumption holding *throughout* the execution. The reason why our verifiers can maintain a valid chain is that, once they receive a chain $\mathcal{C}$ which is the longest, they inductively know that $\mathcal{C}[-k]$ must contain valid application data snapshot. Then, since they have all the last k blocks, they can validate the transactions δ on the snapshot obtained before further mining on top of them.

8 Discussion and Future Work

We have presented a scheme in which full miners are replaced with logarithmic-space miners. Our new mining protocol allows miners to only keep storage growing logarithmically in time. Furthermore, the data communicated to newly bootstrapped nodes is also logarithmic. We focused on optimizing the *consensus data* portion of blockchains (i.e., block headers) without concern for the *application data* portion. Our techniques can be composed with application data optimization techniques.

We have proven our scheme succinct and secure against all 1/3 adversaries. Our treatment requires *uninterrupted* honest computational majority throughout the execution, is in the *static* difficulty model, works only for *proof-of-work* blockchains, and requires modifications to the blockchain protocol for deployment. Let us discuss these aspects of our construction.

Temporary dishonest majority. One important difference between our scheme and the existing blockchain protocols is that traditional full nodes are able to verify the *whole* state evolution of the system from genesis. This allows them to recover in case of temporary dishonest majority [1, 5], while our system cannot do so. Let us consider what could happen in case an adversary temporarily has the upper hand in a blockchain where everybody is mining using our protocol. Let $\mathcal{C}$ denote the chain of the honest parties that has converged. The adversary begins mining on top of the honest tip. She eventually produces $k+1$ new blocks on top of $\mathcal{C}[-1]$, generating an adversarial chain $\mathcal{C}^*$, prior to the honest parties advancing by $k+1$ blocks — a Common Prefix violation. In the block $\mathcal{C}^*[-k-1]$, the adversary places an invalid snapshot; say, a snapshot in which she owns a lot of money. The rest of the blocks in $\mathcal{C}^*[-k{:}]$ are filled with valid transactions. This adversary can then compress this consensus state into a convincing proof, as state transitions buried $k+1$ blocks beyond the tip are never checked. As soon as the honest parties transition to this adversarial chain, the attack concludes, and no more adversarial supremacy is required. It is critical to understand what assumptions our protocol mandates: An *uninterrupted* honest majority throughout the execution. It remains an open question whether it is possible to construct logarithmic space mining protocols that can withstand temporary adversarial supremacy. We note that a similar uninterrupted honest majority assumption (for a 1/2 adversary) is taken in the backbone model [21, 22]. Despite the strength of this assumption, we remark that the honest parties that are online during the time a long adversarial fork is first broadcast

to the network can still *detect* it. This long fork, generated during a temporary dishonest majority, will violate the k Common Prefix property: A light miner will see a new *winning* proof arriving on the network with a χ portion that shares no common blocks with its adopted χ, and the second condition in Line 20 of Algorithm 3 will fail. However, miners that are offline when such a proof is first broadcast, and in particular miners who bootstrap after the attack, will never detect it.

The $1/3$ **adversary.** Ideally, our protocol would work for a $1/2$ adversary. However, it is an inherent limitation of our *construction*, and not the techniques of our proofs, that only a $1/3$ adversary can be withstood. This stems from the $y - 2z$ term in Lemma 4. While generous for small μ, we conjecture that this bound is tight for sufficiently high μ, and so an adversary with power between $1/3$ and $1/2$ can choose to operate at such levels. Therefore, a different construction is necessary to achieve a $1/2$ adversarial bound. We leave such improvements for future work.

Variable difficulty. We have built and analyzed our logarithmic mining protocol in the *constant* difficulty setting, i.e., requiring that the target T is a constant. We strongly suspect, but have not provided proof, that similar protocols to ours work in the variable difficulty setting. One important change in the protocol that is required before it can be adapted to variable difficulty settings is that the χ portion of the proofs cannot be a constant number of blocks long. Instead, it must be a suffix which corresponds to *sufficient work having been performed*, the difficulty of which must correspond to the current target. Simply pruning k blocks long is insufficient. As such, the verifier must first gauge the difficulty of the network prior to taking conclusive decisions. An analysis in the variable difficulty model is beyond the scope of this work. The model required here would make use of the martingale arguments in the variable difficulty backbone model [22]. The precise proofs would need to articulate how the security parameter m is related to the epoch length. We leave such analysis for future work.

Deployment. Our scheme requires the introduction of *interlink* pointers to block headers. Some blockchains have already adopted such headers, namely ZCash [34], ERGO [15], Nimiq, and WebDollar. Ethereum has proposals to adopt such interlinking [11]. Notably, Bitcoin, while possible [25], does not plan to include such a scheme. However, it may be possible to introduce these changes using a velvet fork [32, 47]. While velvet forking can enable (superblock and FlyClient) NIPoPoW clients, it remains an open question whether it can also be used to transition to light mining.

Incentive compatibility. There is a known [10] bribing attack against superblocks. This attack takes place in a rational setting and not in the honest majority setting of the backbone model [21] where our Unsuppressibility Lemma was developed. However, these game theoretic attacks are possible only because superblocks give out rewards utilizing the same schedule as regular blocks. A proposal to make unbribable superblocks (using a soft fork) has been recently put forth [48] and can be readily adopted for our purposes.

Comparison to other NIPoPoWs. Our protocol *is* a Non-Interactive Proof of Proof-of-Work, akin to superblock NIPoPoWs [31] and FlyClient [10]. Our difference with FlyClient is the ability to generate *online* proofs, proofs that can be updated as the blockchain grows. Contrary to our construction, FlyClient requires the sampling of past blocks to change as new blocks are added to the tip of the blockchain. This is due to their use of the Fiat–Shamir heuristic [19]. More concretely, a block that was not sampled in the past may need to be sampled in the future. In our protocol, previously pruned blocks never need to be salvaged. As *any* block has a potential for future samplability in FlyClient, no blocks can be discarded, and mining cannot be logarithmic. This is an inherent limitation of their construction. The construction of superblock NIPoPoWs [31] is similar to ours. However, their construction is not both succinct *and* secure against all adversaries. In particular, their *certificates of badness* allow an adversary to pump the storage state required from logarithmic to linear with the appropriate attack, and their construction is only *optimistically succinct* (i.e., succinct in honest settings). While both constructions sample superblocks, our means of sampling and comparing them are different. Critically, our comparison is unweighted and always takes place across the same levels μ, while theirs must be weighted and compared against potentially different levels μ_A and μ_B. The proofs we have developed here are *always succinct* and *always secure* against any 1/3 adversary (with overwhelming probability). Our analysis is based on our framework consisting of the novel Unsuppressibility Lemma and the generalized Common Prefix theorem, a completely new approach to proving security and succinctness. We are thus the first to propose a NIPoPoW which is *online*, *succinct*, and *secure* against all minority (1/3) adversaries. All of these are necessary prerequisites to achieve the desired goal of logarithmic-space mining. Our new probabilistic analysis techniques can be leveraged to significantly simplify the previous analyses of the above protocols.

Proof of Stake. Our protocol only works for Proof-of-Work blockchains. It seems that our techniques cannot be readily adapted to the Proof-of-Stake setting. The probabilistic nature of Q-blocks and predictable stochastic processes are a by-product of the mining process and the nature of the random oracle model. Simple ideas do not work. If we allow the block producers to annotate their blocks with an appropriate level and sign it, the adversary can fake this. If instead we take a stochastic property of blocks, the adversary can perform *grinding* attacks, putting in *work* while honest parties are only putting in *stake*, leading to an adversarial advantage, since this is a setting where only stake majority assumptions are made. It remains an open question whether the consensus data of stake blockchains can be compressed.

References

1. G. Avarikioti, L. Käppeli, Y. Wang, and R. Wattenhofer. Bitcoin security under temporary dishonest majority. In *23rd Financial Cryptography and Data Security (FC)*. Springer, 2019.
2. G. Avarikioti, E. Kokoris-Kogias, and R. Wattenhofer. Divide and scale: Formalization of distributed ledger sharding protocols. *arXiv preprint arXiv:1910.10434*, 2019.
3. Z. Avarikioti, E. Kogias, R. Wattenhofer, and D. Zindros. Brick: Asynchronous incentive-compatible payment channels. In *International Conference on Financial Cryptography and Data Security*. Springer, 2021.
4. Z. Avarikioti, O. S. T. Litos, and R. Wattenhofer. Cerberus Channels: Incentivizing Watchtowers for Bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 346–366. Springer, 2020.
5. C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Consensus redux: distributed ledgers in the face of adversarial supremacy. Technical report, Cryptology ePrint Archive, Report 2020/1021, 2020.
6. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, ACM, 1993.
7. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.
8. J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121. IEEE Computer Society Press, May 2015.
9. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, IEEE Computer Society, 2018.
10. B. Bünz, L. Kiffer, L. Luu, and M. Zamani. Flyclient: Super-light clients for cryptocurrencies. *IACR Cryptology ePrint Archive*, 2019, 2019.
11. V. Buterin. Blockhash refactoring. *URL: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-210.md*, 2017.

12. V. Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.
13. A. Chepurnoy, M. Larangeira, and A. Ojiganov. Rollerchain, a blockchain with safely pruneable full blocks. *arXiv preprint arXiv:1603.07926*, 2016.
14. A. Chepurnoy, C. Papamanthou, and Y. Zhang. Edrax: A Cryptocurrency with Stateless Transaction Validation. *IACR Cryptology ePrint Archive*, 2018:968, 2018.
15. E. Developers. Ergo: A resilient platform for contractual money, 2019.
16. T. B. Developers. Developer guide - bitcoin. Available at: `https://bitcoin.org/en/developer-guide`.
17. T. Dryja. Utreexo: A dynamic hash-based accumulator optimized for the Bitcoin UTXO set. *IACR Cryptol. ePrint Arch.*, 2019:611, 2019.
18. C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, Springer, 1992.
19. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, Aug. 1987.
20. J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications (revised 2019). Cryptology ePrint Archive, Report 2014/765, 2014. `https://eprint.iacr.org/2014/765`.
21. J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In E. Oswald and M. Fischlin, editors, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 9057 of *LNCS*, pages 281–310. Springer, Apr 2015.
22. J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In J. Katz and H. Shacham, editors, *Annual International Cryptology Conference*, volume 10401 of *LNCS*, pages 291–323. Springer, Aug 2017.
23. E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. Cryptology ePrint Archive, Report 2015/263, 2015. `http://eprint.iacr.org/2015/263`.
24. H. Jameson. Renaming suicide opcode. *URL: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-6.md*, 2015.
25. K. Karantias. Enabling NIPoPoW Applications on Bitcoin Cash. Master's thesis, University of Ioannina, Ioannina, Greece, 2019.
26. K. Karantias. Sok: A taxonomy of cryptocurrency wallets. Technical report, IACR Cryptology ePrint Archive, 2020: 868, 2020.
27. K. Karantias, A. Kiayias, and D. Zindros. Compact storage of superblocks for nipopow applications. In *The 1st International Conference on Mathematical Research for Blockchain Economy*. Springer Nature, Springer Nature, 2019.
28. A. Kiayias, P. Gaži, and D. Zindros. Proof-of-stake sidechains. In *IEEE Symposium on Security and Privacy*. IEEE, IEEE, 2019.
29. A. Kiayias, N. Lamprou, and A.-P. Stouka. Proofs of proofs of work with sublinear complexity. In *International Conference on Financial Cryptography and Data Security*, pages 61–78. Springer, Springer, 2016.
30. A. Kiayias and O. S. T. Litos. A composable security treatment of the lightning network. In *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*, pages 334–349. IEEE, IEEE, 2020.
31. A. Kiayias, A. Miller, and D. Zindros. Non-interactive proofs of proof-of-work, 2017.

32. A. Kiayias, A. Polydouri, and D. Zindros. The Velvet Path to Superlight Blockchain Clients. *IACR Cryptology ePrint Archive*, 2020:1122, 2020.
33. A. Kiayias and D. Zindros. Proof-of-work sidechains. In *International Conference on Financial Cryptography and Data Security: Workshop on Trusted Smart Contracts*. Springer, Springer, 2019.
34. Y. T. Lai, J. Prestwich, and G. Konstantopoulos. FlyClient - Consensus-Layer Changes. *URL: `https://zips.z.cash/zip-0221`*, 2019.
35. R. Matzutt, B. Kalde, J. Pennekamp, A. Drichel, M. Henze, and K. Wehrle. How to securely prune bitcoin's blockchain. In *2020 IFIP Networking Conference (Networking)*, pages 298–306. IEEE, 2020.
36. I. Meckler and E. Shapiro. Coda: Decentralized cryptocurrency at scale. https://cdn.codaprotocol.com/v2/static/coda-whitepaper-05-10-2018-0.pdf, May 2018.
37. R. C. Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.
38. A. Miller. The high-value-hash highway. bitcoin forum post, 2012.
39. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
40. A. Poelstra. Mimblewimble. 2016.
41. J. Poon and V. Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, 2017.
42. J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.
43. W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.
44. J. Teutsch and C. Reitwießner. A scalable verification solution for blockchains. *arXiv preprint arXiv:1908.04756*, 2019.
45. G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151:1–32, 2014.
46. K. Wüst and A. Gervais. Ethereum eclipse attacks. Technical report, ETH Zurich, 2016.
47. A. Zamyatin, N. Stifter, A. Judmayer, P. Schindler, E. Weippl, W. Knottenbelt, and A. Zamyatin. A wild velvet fork appears! inclusive blockchain protocol changes in practice. In *International Conference on Financial Cryptography and Data Security*. Springer, 2018.
48. D. Zindros. Soft Power: Upgrading Chain Macroeconomic Policy Through Soft Forks. In *International Conference on Financial Cryptography and Data Security*. Springer, Springer, 2021.

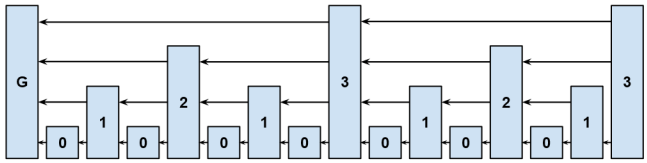
Appendix

A Maintaining Interlinks

We wish to connect the blocks at each level with a *previous block* pointer pointing to the most recent block of the same level. These pointers must be included in the data of the block so that proof-of-work commits to them. As the level of a block cannot be prediced before its proof-of-work is calculated, we extend the *previous block id* structure of classical

blockchains to be a set, the *interlink set*. The interlink set points to the most recent preceding block of every level μ (ignoring duplicates [25]). A pointer to $\mathcal{G}$ is included in every block. The number of pointers that need to be included per block is in expectation $\mathcal{O}(\log(|\mathcal{C}|))$ [27].

Fig. 3. The probabilistic hierarchical blockchain. Higher levels have achieved a higher difficulty during mining. All blocks are connected to the genesis block G.



The algorithm for this construction is shown in Algorithm 4 [27]. The interlink set of the Genesis block is, by definition, empty. The algorithm describes how the interlink can be updated once a block is found. The new interlink is then included in the next block. This construction ensures that every block contains a direct pointer to its most recent μ-superblock ancestor, for every $\mu \in \mathbb{N}$.

Algorithm 4 The updateInterlinkSet algorithm which updates the interlink set

```
1: function updateInterlinkSet(B')
2:     interlinkSet ← {H(B')}
3:     for H(B) ∈ B'.interlink do
4:         if level(B) > level(B') then
5:             interlinkSet ← interlinkSet ∪ {H(B)}
6:         end if
7:     end for
8:     return interlinkSet
9: end function
```

The updateInterlinkSet algorithm accepts a block B', which already has an interlink data structure defined on it. The function evaluates the interlink data structure which needs to be included as part of the next block. It copies the existing interlink data structure from level $\mathsf{level}(B')$ and adds the reference $H(B')$.

B Mathematical Background

Theorem 7 (Chernoff bounds). *Suppose* $\{X_i : i \in [n]\}$ *are mutually independent Boolean random variables, with* $\Pr[X_i = 1] = p$*, for all* $i \in [n]$*. Let* $X = \sum_{i=1}^{n} X_i$ *and* $\mu = pn$*. Then, for any* $\delta \in (0, 1]$*,*

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\delta^2\mu/2} \text{ and } \Pr[X \geq (1+\delta)\mu] \leq e^{-\delta^2\mu/3}.$$

Lemma 6. *For each* $j \in \mathbb{N}$*, let* F_j *and* M_j *be Boolean random variables such that* $\mathbb{E}[M_j] = \zeta$ *and* M_j *is independent of* F_i *for* $i \leq j$ *and independent of* M_i *for* $i \neq j$*. For any* $\epsilon \in (0, 1)$*,*

$$\Pr\left[\sum F_j M_j > (1+\epsilon)\zeta \sum F_j \bigwedge \sum F_j M_j \geq k\right] \leq e^{-\Omega(\epsilon^2 k)}.$$

Proof. Since $\sum_{n \geq k} e^{-\Omega(\epsilon^2 n)} = e^{-\Omega(\epsilon^2 k)}$, by the union bound it suffices to show that

$$\Pr\left[(1+\epsilon)\zeta \sum F_j < k \bigwedge \sum F_j M_j = k\right] \leq e^{-\Omega(\epsilon^2 k)}. \tag{4}$$

In the summations below, let α range over words in $\{0,1\}^*$ and β be any word in $\{0,1\}^\ell$ of weight k. For a fixed α, define $J_\alpha = \{j \in \mathbb{N} : F_j = 1\}$ and $B = (M_j)_{j \in J_\alpha}$. Also, for $j \in \mathbb{N}$, let E_j denote the event $\{(\forall i < j)(F_i = \alpha_i$ and $i \in J \Rightarrow M_i = \beta_i)\}$. Then,

$$\begin{aligned}
\Pr[B = \beta] &= \sum_\alpha \Pr[B = \beta, A = \alpha] \\
&= \sum_\alpha \prod_j \Pr[F_j = \alpha_j | E_j] \prod_{j \in J} \Pr[B_j = \beta_j | E_j, F_j = \alpha_j] \\
&= \sum_\alpha \prod_j \Pr[F_j = \alpha_j | E_j, B = \beta] \prod_{j \in J} \Pr[M_j = \beta_j] \\
&= \sum_\alpha \Pr[A = \alpha | B = \beta] \cdot \zeta^k (1-\zeta)^{\ell - k} \leq \zeta^k (1-\zeta)^{\ell-k}.
\end{aligned}$$

Thus, letting β range over all words in $\{0,1\}^*$ of length less than $\frac{k}{(1+\epsilon)\zeta}$ and weight k ending with 1, the left-hand side of (4) is equal to

$$\sum_\beta \Pr[B = \beta] \leq \sum_{k \leq \ell < \frac{k}{(1+\epsilon)\zeta}} \binom{\ell - 1}{k-1} \zeta^k (1-\zeta)^{\ell-k}.$$

That is, the probability is at most that of a random variable following a negative binomial distribution with parameters k (the number of successes) and ζ (the probability of success) is less than $\frac{k}{(1+\epsilon)\zeta}$. The bound follows from standard Chernoff bounds.

Acknowledgements

The authors would like to thank Peter Gaži for proof reading the construction and pinpointing important errata.