Mining in Logarithmic Space with Variable Difficulty

Loïc Miller
loic.miller@imt-atlantique.fr
IMT Atlantique / IRISA
Rennes, France

Dorian Pacaud
dorian.pacaud@imt-atlantique.fr
IMT Atlantique / IRISA
Rennes, France

Nathanaël Derousseaux–Lebert
IMT Atlantique / IRISA
Rennes, France

Emmanuelle Anceaume
emmanuelle.anceaume@irisa.fr
CNRS / IRISA
Rennes, France

Romaric Ludinard
romaric.ludinard@imt-atlantique.fr
IMT Atlantique / IRISA
Rennes, France

Abstract

This paper presents the first non-interactive, succinct, and secure representation of a PoW-based blockchain that operates under variable mining difficulty while satisfying both completeness and onlineness properties. Completeness ensures that provers can update an existing NIPoPoW by incorporating a newly mined block, whereas onlineness ensures that miners can extend the chain directly from a NIPoPoW. The time complexity for both the prover (to update a NIPoPoW with a new block) and the verifier is logarithmic in the number of blocks of the underlying PoW blockchain. The communication complexity required for synchronization is polylogarithmic in the length of the blockchain. We prove the correctness of our scheme in the presence of a 1/3-bounded PPT adversary.

CCS Concepts

• **Security and privacy** → **Distributed systems security**; • **Computing methodologies** → **Concurrent algorithms**; • **Theory of computation** → *Distributed algorithms.*

ACM Reference Format:
Loïc Miller, Dorian Pacaud, Nathanaël Derousseaux–Lebert, Emmanuelle Anceaume, and Romaric Ludinard. 2025. Mining in Logarithmic Space with Variable Difficulty. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3744874

1 Introduction

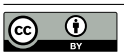
The immutability of a Proof-of-Work (PoW)-based blockchain is the feature that brings an unprecedented level of trust to transactions that have been sequentially stored in each block of the blockchain, for over 16 years in the case of Bitcoin. Concretely, in the absence of trusted third parties, each node interested in any single transaction t must verify the validity of both application and consensus data belonging to every block preceding the block containing t. Application data includes transactions, account data, and smart contract status. Consensus data includes all the information needed for the correct construction of the block sequence. At the time of writing, verifying all the blocks of the Bitcoin blockchain requires downloading 890,500 blocks for a total of 640 GB. This is difficult to reconcile with mobile devices or for nodes that run different blockchain clients. Different techniques have been proposed and are currently deployed to securely optimize application data. These encompass SNAP [29], Layer 2 constructions [16, 20], side chains [3, 13, 27], or compression of multiple transactions into smaller ones [9]. These techniques are by construction not applicable to consensus data, essentially because consensus data guarantees the construction of a unique and immutable sequence of blocks. Consensus data is sealed in the header of each block and includes, in particular, the fingerprint of the previous block in the blockchain, block PoW nonce, and block mining difficulty.

Light clients are protocols that enable faster synchronization of consensus data. The Simplified Payment Verification (SPV) protocol, presented as part of the Bitcoin protocol [26], downloads only the header of each block instead of each full block. This reduces the verification overhead from a few megabytes per block to just 80 B per block. However, the overhead remains substantial as it grows linearly with the number of blocks. The crucial problem to be solved for the full adoption of blockchains is therefore the following:

> **Can we build and non-interactively verify in a sublinear number of operations and in the absence of full nodes a succinct representation of a PoW-based blockchain operating in a permissionless system[1]?**

Sublinear light clients have been proposed to address this problem. In particular, FlyClient [7] allows any user to store only a compact proof of the underlying blockchain. However, *(i)* verification requires a logarithmic number of interactions with full nodes, and *(ii)* the full blockchain is still needed to update the proof. The Non-Interactive Proof of Proof-of-Work (NIPoPoW) proposed by Kiayias *et al.* [19] is a succinct representation of a PoW-based blockchain (i.e., it contains a logarithmic number of blocks). New blocks can be mined directly on top of a NIPoPoW, which does not necessitate the maintenance of full nodes. Any user (miner or client) can be convinced that it is a secure representation of an honest underlying blockchain in a single interaction. The essence of their construction relies on an elegant idea – they take advantage of the probabilistic distribution of block fingerprints to subsample certain blocks called superblocks. A block is an ℓ-superblock if

[1] A system in which nodes can join and leave without the approval or trust of others is said to be permissionless. Bitcoin is a permissionless blockchain where nodes (miners or clients) are free to participate in its construction.



CCS '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3744874

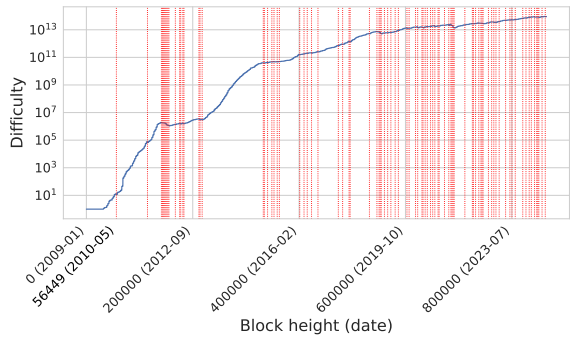


Figure 1: Bitcoin difficulty variation over time (865,042 blocks as of October 11, 2024). Vertical dashed lines indicate a decrease in the mining difficulty compared to the previous epoch.

its fingerprint meets the PoW condition with a $1/2^\ell$ margin, i.e., $H(\text{block}) \leq T/2^\ell$ for some target T. Higher-level superblocks are exponentially rarer – every block is a 0-superblock, half on average are 1-superblocks, a quarter are 2-superblocks, and so on. Kiayias *et al.*'s solution [19] consists in subsampling the last $2K$ ℓ-superblocks, for $\ell = 0, \ldots, \lfloor \log_2(n) \rfloor - 1$ with K a security parameter and n the number of blocks of a PoW-based blockchain. They show that the headers of this logarithmic number of superblocks plus a constant number of full blocks is sufficient to convince a verifier that they are representative of the hashing power devoted to the construction of the entire PoW-based blockchain. However, the construction and verification algorithms of Kiayias *et al.*'s NIPoPoW are correct only if the mining difficulty of each block is constant, i.e., only if the computational power devoted to the creation of each single block is constant since the inception of the blockchain. This is an unrealistic assumption, as illustrated in Figure 1, which shows the evolution of block mining difficulty since Bitcoin's creation. The difficulty continuously increases except for the occasional slight reductions.

1.1 Current Challenges

Envisioning NIPoPoWs compliant with variable mining difficulty raises numerous challenges.

[C1] We need to quantify the extent to which a block deserves to be sampled to be representative of the honest blockchain. In a variable difficulty setting, leveraging only block fingerprints is not enough. Indeed, two blocks showing very close fingerprints might have been mined with different difficulties and thus do not exhibit the same rarity. In contrast, two blocks with fingerprints far from each other may equally outperform their respective targets and thus may both deserve to be sampled.

[C2] Secondly, variable difficulty in any PoW-based blockchain requires verifying the legitimacy of each block's difficulty. In Bitcoin, miners recompute the target every 2016 blocks (an epoch) to maintain a stable block production rate. However, when only a sublinear number of blocks is sampled, this verification becomes impossible, giving rise to adversarial manipulation of difficulty known as *low-difficulty attacks*.

Definition 1 (Low-difficulty attack). *A low-difficulty attack involves an adversary privately mining a chain whose older blocks have strictly lower difficulties than older honest blocks and whose recent blocks have mining difficulties comparable to those of recent honest blocks.*

By mining old blocks with low difficulty, the adversary will be able to mine old blocks very quickly, matching the number of blocks mined by honest parties, although the adversarial computing power is 1/3 less than the total hashing power. Since block sampling retains very few old blocks, a verifier will not be able to detect the adversary's use of illegitimately low difficulty. In contrast, block sampling retains all very recent blocks, allowing a verifier to check the accuracy of the difficulty of the adversary's recent blocks. An adversarial prover will therefore present the verifier with a NIPoPoW that seems to be indistinguishable from a NIPoPoW that represents the honest blockchain. This attack is very subtle but needs to be detected by verifiers to make NIPoPoWs an interesting alternative to full blockchains. Note that a less subtle low-difficulty attack is a brute-force one – the entirety of an adversarial blockchain is mined with illegitimately low mining difficulties, giving rise to a longer blockchain than the honest one. Recall that existing NIPoPoW constructions do not need to defend against low-difficulty attacks since they suppose that all the blocks have been created with the same legitimate difficulty [15, 19, 21].

[C3] Finally, we need to tightly dimension the security sampling parameter K so that if an adversarial prover builds or updates a NIPoPoW with illegitimate block mining difficulties, with overwhelming probability, a verifier will reject the non-legitimate NIPoPoW. This is crucial to defend against low-difficulty attacks.

1.2 Our Contributions

We respond to the above challenges by offering, for the first time, a non-interactive, succinct, and secure representation of a PoW-based blockchain that operates under variable mining difficulty while satisfying both *completeness* and *onlineness* properties. Completeness states that provers can produce a NIPoPoW by updating the current one with a new block, while onlineness says that miners can mine new blocks directly from a NIPoPoW. The time complexity for both the prover (to update a NIPoPoW with a new block) and the verifier is logarithmic in the number of blocks of the underlying PoW blockchain. The communication complexity required for synchronization is polylogarithmic in the number of blocks of the blockchain. The main ideas of our solution are as follows.

First, we subsample blocks so that sampled blocks at level $\ell = 0, \ldots, \lfloor \log_2(|C|) \rfloor$ are equally representative of a blockchain C despite a variable difficulty setting. This is achieved by defining the level of a block as the ratio between its fingerprint and its mining target, projected onto the unit interval $[0, 1]$. A given level may therefore contain blocks whose mining difficulties are different, even more so for high levels. This reflects the fact that those blocks are equally representative of the blockchain they are sampled from but have been created in epochs that involved different computational powers. This responds to Challenge **C1**. Observe that as the blockchain increases, the set of sampled blocks evolves – some previously sampled blocks may subsequently be discarded. However, due to our definition of block level, any discarded block is

guaranteed to be irrelevant for future updates, regardless of its difficulty.

Second, let b be the last common block shared by an honest blockchain C and an adversarial one C'. We show that for any block level ℓ, if the quantity of difficulty accumulated in the ℓ-blocks of C' from block b reaches a certain quantity $D^{\mathcal{A}}_{\min}(\ell)$, then with overwhelming probability the quantity of difficulty accumulated in the ℓ-blocks of C from block b is strictly greater than the quantity accumulated in the ℓ-blocks of C' from the same block b.

Third, we quantify the number of ℓ-blocks that are sufficient to accumulate $D^{\mathcal{A}}_{\min}(\ell = 0)$ difficulty. This is achieved by combining the gambler's ruin problem with the Poisson distribution. Briefly, we model the mining process as a two-phase competition between two teams, the adversarial team and the honest nodes team. Both teams start at their last common block b. We study two scenarios. In the first phase of the competition, both teams create their own blocks until the honest team has created $K_{\mathcal{H}}$ blocks (first scenario) or the adversary has created $K_{\mathcal{A}}$ blocks (second scenario). In the second phase of both scenarios, both teams continue to create their own blocks until either the adversarial team catches up and exceeds the accumulated quantity of difficulty of the honest team, in which case it wins the competition, or the adversarial team is so far behind it is hopeless for it to ever catch up to the quantity of difficulty of the honest blockchain. We set $K = \max(K_{\mathcal{H}}, K_{\mathcal{A}})$ so that the latter case holds with overwhelming probability. The question is whether K blocks are sufficient to accumulate difficulty $D^{\mathcal{A}}_{\min}(\ell)$, at any level $\ell > 0$. A corollary of the ℓ-common prefix theorem (Theorem 6.3) shows that $D^{\mathcal{A}}_{\min}(\ell)$ is exponentially decreasing with level ℓ, which positively answers the question and thus responds to Challenge **C3**. We have compared, for each level $\ell \geq 0$, the accumulated difficulty of the last K blocks of the Bitcoin blockchain up to October 11, 2024, with the theoretical bound $D^{\mathcal{A}}_{\min}(\ell)$ calculated in Lemma 6.3). This comparison clearly illustrates that K blocks are sufficient to accumulate $D^{\mathcal{A}}_{\min}(\ell)$ quantity of difficulty for each level $\ell \geq 0$.

Finally, to respond to Challenge **C2**, we combine the exponential decay property of $D^{\mathcal{A}}_{\min}(\ell)$ together with the security parameter K and the notion of the latest common ancestor (LCA) block to prove that verifiers detect low-difficulty attacks. The notion of LCA is fundamental, as it provides important blockchain structural information when comparing their NIPoPoWs. More precisely, if two NIPoPoWs share an LCA block b at level μ, it means that *(i)* both NIPoPoWs have the same level (see Definition 4), i.e., their underlying blockchains have approximately the same number of blocks; *(ii)* both underlying blockchains share a common prefix and block b is their last common μ-block, and *(iii)* from block b onward, at least one of both underlying blockchains has no less than K μ-blocks. Hence, if the adversarial prover presents a NIPoPoW Π' (representative of a blockchain created during a low-difficulty attack), and that Π' shares an LCA block with an honest NIPoPoW Π, then whatever the level μ of the LCA block, Π' will not be able to accumulate more difficulty in the K sampled μ-blocks than the honest ones of Π. Therefore, the adversarial prover will not be able to convince a verifier that Π' is representative of the honest blockchain. Now, if Π and Π' do not share an LCA block (one is the result of a low-level brute force attack as argued in Section 5.3.3), a verifier will exploit the initial Sync handshake of the Bitcoin protocol to reject the adversarial NIPoPoW. Briefly, when a verifier v wishes to synchronize with the network, prior to executing the verifier algorithm, v annotates a Sync request with a privately generated random number ν and broadcasts the Sync request to the system. Upon receipt of annotated Sync requests, any correct miner u accumulates in a set, say V, all the random numbers $\nu_i, \ldots, \nu_j$ that u has received but which u has not inserted yet into the coinbase of the blocks u has created so far. Adversarial miners are free to not insert V in their block coinbase, but verifier u will only compare NIPoPoWs Π and Π' that do contain blocks b and b' annotated with ν. Verifier u will observe the network until it receives the first K subsequent blocks of b or b'. Once received, u will be able to safely accept the honest NIPoPoW.

To summarize, we present a Non-Interactive Proof of Proof-of-Work (NIPoPoW) that

- operates in a variable mining difficulty setting;
- is succinct: A NIPoPoW is made of a logarithmic number of block headers in the number of blocks of the underlying blockchain. Experiments on Bitcoin blockchain C show that $2K\log(|C|) \leq \text{NIPoPoW}(C) \leq 3K\log(|C|)$, where K is the provably secure sampling parameter of the construction;
- requires a logarithmic number (in the number of blocks of the underlying blockchain) of operations to be updated and verified;
- requires a polylogarithmic number of information and an update time polylogarithmic in the number of blocks of the underlying blockchain for a verifier to synchronize;
- provably achieves security against a Byzantine adversary owning at any time strictly less than 1/3 of the system hashing power.

In the remainder of the paper, Section 2 details the related work with a particular emphasis on Kiayias *et al.*'s NIPoPoW scheme. Section 3 presents the model of the system, and Section 4 formally specifies a Non-Interactive Proof of Proof-of-Work in a variable mining setting. Section 5 presents the prover and verifier algorithms. Section 6 shows the correctness of our solution. Section 7 concludes.

2 Related Work

2.1 Light clients

The problem of blockchain becoming of considerable size was initially predicted by Satoshi Nakamoto in the seminal paper that introduced Bitcoin [26]. He offered a simple solution, *Simplified Payment Verification (SPV)*, that requires a client to only store block headers and leave out transactions. Still, the amount of data that needs to be downloaded from the network grows linearly with the size of the blockchain. FlyClient [7] allows a succinct and secure construction of proofs in a variable difficulty setting. They make use of Merkle mountain ranges to reference the whole previous blockchain from every block. However, if a full node has a proof and mines a new block on top of it, it cannot create a new optimal proof without holding the whole chain. CoinPrune [24] still requires storing the entire chain of block headers prior to the pruning point. Another approach to building succinct proofs is to rely on SNARKs

Table 1: Compression schemes. $|C|$ is the number of blocks in C; c is the block header size; t_x is the block transaction size; k is the common prefix parameter; a is the snapshot size and χ is the number of blocks of the uncompressed subchain.

	Storage	Communication	Online	Variable Difficulty	Adv.
BTC SPV	$\|C\|c + \log(t_x)$	$\|C\|c + \log(t_x)$		✓	1/2
Kiayias [21]	$\|C\|c + kt_x + a$	$\text{poly}\log(\|C\|)c + kt_x + a$			1/3
FlyClient [7]	$\|C\|c + kt_x + a$	$\text{poly}\log(\|C\|)c + kt_x + a$		✓	1/2
Kiayias [19]	$\text{poly}\log(\|C\|)c + kt_x + a$	$\text{poly}\log(\|C\|)c + kt_x + a$	✓		1/3
Jain [15]	$\text{poly}\log(\|C\|)c + kt_x + a$	$\text{poly}\log(\|C\|)c + kt_x + a$	✓		1/2
This work	$\text{poly}\log(\|C\|)c + (\chi + k)t_x + a$	$\text{poly}\log(\|C\|)c + (\chi + k)t_x + a$	✓	✓	1/3

(Succinct Non-Interactive Argument of Knowledge). Mina [6], formerly known as Coda [5][2], is such a construction. Mina compresses a chain to polylogarithmic size and updates the proof with new blocks. However, utilizing SNARKs requires a trusted setup for the common reference string.

2.2 Non-Interactive Proofs of Proof-of-Work

A Non-Interactive Proof of Proof-of-Work (NIPoPoW) primitive aims at constructing a proof that is representative of the size of the original blockchain. Such a primitive has been proposed and instantiated by Kiayias *et al.* [19], and elegantly relies on the idea that sampling a small set of well-chosen blocks is sufficient to estimate the size of the original blockchain [17, 19, 21]. To be kept as a sample, a block needs to satisfy a specific property on its cryptographic hash. The distribution of hash values is stochastic, and thus some blocks end up with hash values significantly below the mining target T. Blocks that hash to a value less than $T/(2^\ell)$ are called ℓ-superblocks [21], where T is the mining target and $\ell \geq 0$ is called the level of the block. The notion of ℓ-superblock reflects *block rarity*. In expectation, for a blockchain C of n blocks, only one block of C is a $(\lfloor\log(n)\rfloor)$-superblock, two blocks of C are $(\lfloor\log(n)\rfloor - 1)$-superblocks, ..., and all the blocks of C are 0-superblocks. A NIPoPoW requires every block header to store pointers to the last ℓ-superblock that precedes it at every level $\ell \geq 0$ in order to ensure that the subsampled blocks also form a valid chain, i.e., a totally ordered sequence of valid blocks. The construction proposed by Kiayias *et al.* [19] consists in keeping the $2K$ most recent blocks of each level, where K is a security parameter whose value, by rule of thumb, is set proportionally to the common prefix parameter k[3]. Note that Kiayias *et al.* [19] use m in place of K, but this can cause confusion with the length of an epoch m used to recalculate block mining difficulties. Their proof is resilient to a 1/3-adversary. The authors in [15] extend Kiayias *et al.* [19]'s scheme to obtain a proof that is correct in the presence of a 1/2-bounded PPT adversary. Their main idea is to attach increasing weights $W_\beta(\ell)$ to ℓ-superblocks, making them "diamond-blocks" so that the selected proof is the heaviest. Because the adversary has minority mining power, they cannot create a heavier sequence of diamond blocks faster than the honest parties, for the same reason that an adversary cannot create a longer regular blockchain faster than the honest parties create one.

However, Jain et al. [15] prove the correctness of their construction in a static setting, that is, assuming that the mining difficulty is constant. Our preliminary results on NIPoPoWs in a variable difficulty setting appear in [23]. In this previous work, parameter K evolves as a function of the variation of block mining difficulties. Specifically, when a sampled ℓ-block b shows a decreasing mining difficulty with respect to the oldest ℓ-block b' of the proof, block b' is not pruned from the proof. Instead, it remains in the proof as long as the accumulated difficulty of new sampled ℓ-blocks is less than the one obtained before updating the proof with b. Beyond its complexity, the latency of the update depends on the variation of mining difficulties, which is undesirable and an opportunity for the adversarial prover to devise strategic attacks.

Table 1 summarizes the storage and communication complexity of these constructions, indicates whether or not the construction is sufficient to create new blocks and to synchronize any newcomer, whether it fits a dynamic environment, and finally whether it is robust against a 1/2 or a 1/3-bounded PPT adversary. This work improves upon existing solutions by providing a scheme that is both online, i.e., one can mine blocks directly from the proof, and works in a variable difficulty setting against a 1/3-bounded PPT adversary. The storage and communication costs of our scheme are comparable to Kiayias *et al.* [19] and Jain *et al.* [15]. We add one constant, keeping our scheme with polylogarithmic complexity.

3 Model of the system

We consider Proof-of-Work blockchains, that is, blockchains whose block construction requires solving a resource-consuming calculation [2]. We specifically focus on consensus data, and we assume that each block contains a snapshot of the application state. In Bitcoin, for instance, a snapshot of the application data is composed of the current set of UTXOs. Thus the state committed by a block is represented by the snapshot of this block.

Proofs of correctness of our construction rely on the model adopted by Garay *et al.* [11]. Specifically, we consider a *synchronous* setting where time is quantized into discrete rounds r during which each party can send a message to other parties, receive the messages sent to it during the round, and execute computational steps based on the received messages. We assume the presence of an adversary that models the behaviors of adversarial parties. The series $n = \{n_r\}_{r\in\mathbb{N}}$ represents the total number of participants in

[2] https://minaprotocol.com/blog/coda-protocol-relaunches-as-mina-the-worlds-lightest-blockchain.

[3] For any two blockchains C_1 and C_2, with $|C_1| \leq |C_2|$, we say that k is the common prefix of both blockchains if C_1, from which the k most recent blocks have been removed, is a prefix of C_2.

the system over time, $t = \{t_r\}_{r\in\mathbb{N}}$ of which are adversarial. The relative proportion of adversarial parties is bounded by a variable δ, typically called the advantage of honest parties, and we have $\forall r : t_r \leq (1-\delta)(n_r - t_r)$. Each party is allowed to make ρ queries to the cryptographic hash function in every round to create a block (specifically to find an adequate nonce). We say that a block b is valid if its header contains a nonce, along with non-double-spending transactions, that hashes to a value h below a given mining target. The cryptographic hash function $\mathsf{h}(.)$ behaves as an ideal random function and is modeled as a random oracle [4]. It produces a constant length κ output, where κ is a security parameter typically equal to 256. The adversary can query the cryptographic hash function up to $t_r \times \rho$ times per round, where t_r represents the number of adversarial parties at round r [12]. We denote by f the probability that at least one honest query is successful during one round, i.e., the probability that at least one honest party successfully mines a block during one round. Table 2 lists all the variables used.

We suppose that the adversary is a *rushing adversary* in the sense that they can observe what the honest parties have done during the round before using their computational power at the end of the round. The adversary is also a *Sybil adversary* as they can inject as many additional messages as they wish by faking multiple identities. We limit the adversary to a probabilistic polynomial-time Turing machine that behaves arbitrarily. The adversary may thus not follow the prescribed algorithms, but it remains computationally bounded. Hence, it cannot, in a polynomial number of steps of time or space, forge honest party signatures or break the hash function and signature scheme with all but negligible probability. Therefore, we name our adversary the *1/3-bounded PPT adversary*. Any party following the prescribed protocol is called an *honest* party.

4 Non-Interactive Proof of Proof-of-Work in a variable difficulty setting

A Non-Interactive Proof of Proof-of-Work system in a variable difficulty setting consists of two algorithms: a prover algorithm and a verifier algorithm. The prover algorithm, called `Compress()`, takes as input a chain of blocks with variable mining difficulties and returns a NIPoPoW Π. The node that executes the prover algorithm is called the prover. In contrast to an honest prover, an adversarial prover accepts to compress chains whose certain blocks were created with illegitimate mining difficulties. The verifier algorithm, called `Compare()`, takes as input two NIPoPoWs (or, more generally, two chains of blocks) and returns the accepted NIPoPoW. A verifier is typically a newcomer (miner or simple user) but more generally any party that may have lost connection for some time.

Definition 2 ((Compress, Compare)). *We call (`Compress()`, `Compare()`) a Non-Interactive Proof of Proof-of-Work system if, for all 1/3-bounded PPT adversaries, it has the security, succinctness, completeness, and onlineness properties described below.*

Security: *For any two proofs Π and Π' such that at least one of them is presented by an honest prover, `Compare`(Π, Π') returns the proof that is representative of the blockchain that has accumulated the largest quantity of difficulty;*

Succinctness: *For any blockchain C maintained by an honest party, $|\mathtt{Compress}(C)| = O(\mathrm{poly}\log|C|)$;*

Completeness: *For any blockchain C maintained by an honest party and any block b valid for C, let $\Pi = \mathtt{Compress}(C)$. Then $\mathtt{Compress}(C\,b) = \mathtt{Compress}(\Pi\,b)$;*

Onlineness: *For any blockchain C maintained by an honest party, the state committed by $\mathtt{Compress}(\Pi)$ is equal to the one committed by C,*

where $|C|$ represents the number of block headers of C, and $C\,b$ (resp. $\Pi\,b$) indicates that block b is appended to C (resp. proof Π).

Security states that if two honest provers present their proofs, the verifier will be convinced by the one that has accumulated the largest quantity of difficulty (this is generally called "perfect completeness"). Furthermore, it states that if among both provers one is adversarial, then it will not be able to convince the verifier (this is generally called "perfect soundness") unless it behaves honestly. Completeness states that a prover can produce a proof Π by updating the current one with a new block, while onlineness says that miners can mine new blocks directly from a proof. Onlineness is crucial for scalability, as the underlying blockchain does not need to be maintained; the proof is self-contained.

5 The Prover and Verifier Algorithms

5.1 Variable mining difficulty

Let C be some PoW-based blockchain. To guarantee that on average the time interval between any two successive block creations is constant, despite unpredictable variations of the global hashing power, miners in PoW-based blockchains periodically recalculate the block mining difficulty. The period of time between any two successive recalculations is called an epoch. In Bitcoin, an epoch is made of $m = 2016$ blocks, i.e., two weeks. The mining difficulty at epoch i represents how much more difficult it is to mine a block at epoch i than it was when the genesis block of the blockchain was created. The mining difficulty is inversely proportional to the mining target [26]. This last notion refers to the PoW inequation satisfied by any block b of a PoW-based blockchain, i.e., $\mathsf{h}(b) \leq T_i$, where T_i represents the mining target at epoch i. In the following, we will denote by $\|b\|$ the mining difficulty of block b, and by extension by $\|C\|$ the mining difficulty of blockchain C. This is equal to the sum of the difficulties of all the blocks that comprise C. If $\|b\| = 1/T_i$, block b is valid for epoch i.

5.2 Evaluating the significance of a block

We revisit the definition of block level, introduced in Kiayias *et. al* [19] to take into account variable mining difficulties. The level of a block is an indicator of the blockchain's size: the higher the level, the more representative the block is of the total number of blocks in the chain. The level of a block is a projection of its fingerprint over its mining target onto the unit interval $[0, 1]$. A given level may contain blocks whose mining targets are different, and even more so for high levels. This reflects the fact that those blocks are equally representative of the original blockchain but have been created in epochs that involved different computational powers.

Definition 3 (Level of a block). *Let C be a blockchain, and let block $b \in C$ be mined during epoch $i \geq 0$. We say that the level of*

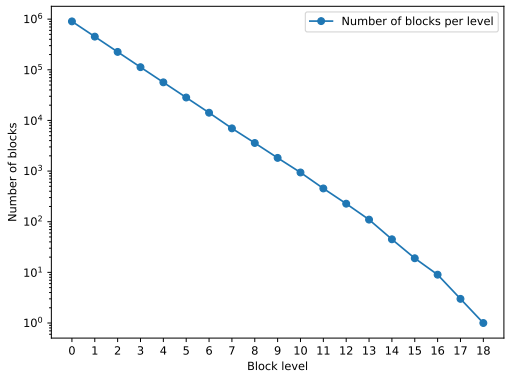


Figure 2: Number of blocks in the Bitcoin blockchain as of October 11, 2024, as a function of their level (see Definition 3).

block b is equal to ℓ if it verifies the following condition:

$$\frac{\mathsf{h}(b)}{T_i} \leq \frac{1}{2^\ell}.$$

A block of level ℓ is called an ℓ-block. By construction, a block of level ℓ is also a block of level ℓ', for all $0 \leq \ell' < \ell$. By convention, the genesis block has an infinite level. On average, one needs to create 2^ℓ blocks to obtain one block of level ℓ. There is therefore on average one block of level $\lfloor \log_2(|C|) \rfloor$, two blocks of level $\lfloor \log_2(|C|) \rfloor - 1$, $\ldots$, $|C|$ blocks of level 0. By sampling only a few blocks at each level, one will be able to exponentially compress the blockchain. Figure 2 has been plotted from the Bitcoin blockchain on October 11, 2024, and depicts the number of blocks that match a given block level ℓ with a logarithmic scale on the vertical axis. As expected, decay is exponential.

5.3 Construction of a NIPoPoW

A NIPoPoW consists of two algorithms. The prover algorithm, called `Compress()`, aims to build a NIPoPoW from a complete blockchain or to update a NIPoPoW when a new valid block is received. The verifier algorithm, called `Compare()`, aims to compare NIPoPoWs in order to select from them the one that has accumulated the largest quantity of difficulty. Our algorithms are relatively close to the ones proposed in [19]. This is very valuable because it keeps the elegance of the original construction, and it makes ours compliant with both static and variable difficulties.

5.3.1 Notations used in the algorithms. We often use C to denote a blockchain, with $C[i]$ representing its i^{th} block. $C[:j]$ represents the sequence of blocks of C from the genesis block to its j^{th} block exclusive, and $C[i:]$ denotes the sequence of blocks from the i^{th} one inclusive to the last blocks of C. $C[i:j]$ denotes the sequence of blocks from the i^{th} one inclusive to the j^{th} block exclusive. The block indices i and j can be replaced by blocks A and Z. We then write $C\{A:Z\}$ to designate the chain from block A inclusive to Z exclusive. Again, any end can be omitted. A negative index means to take blocks from the end, $C[-1]$ denotes the tip of C. We write $C \uparrow^\ell$ to mean the subsequence of C containing only its ℓ-blocks. The $C \uparrow$ operator is absolute: $(C \uparrow^\ell) \uparrow^{\ell+i} = C \uparrow^{\ell+i}$. Since every block header keeps pointers to the last preceding block of every level (see Section 5.3.2), $C \uparrow^\ell$ forms a chain of blocks. $C_1 \cap C_2$ denotes the chain consisting of blocks only in both chains. We note $C_1 \setminus C_2$ the chain consisting of blocks in C_1 but not C_2. The chain filtering operators $[\cdot]$, $\{\cdot\}$, $\uparrow$ have precedence over the $\cup$, $\cap$ and $\setminus$ operators.

5.3.2 The prover algorithm. The `Compress()` algorithm, whose pseudo-code is given in Algorithm 1, aims to sample a polylogarithmic number (in the size of the underlying blockchain C) of well-chosen blocks from C or from any chain of blocks anchored at the genesis block. The latter case enables the `Compress()` algorithm to be used to update a NIPoPoW with a new valid block. Only sampled blocks will appear in the NIPoPoW; all other blocks will be pruned. The sample still forms a chain of blocks by having, for each block header, the fingerprint of the last preceding block at every level $\ell = 0, \ldots, \lfloor \log_2(|C|) \rfloor$.

Execution of the $\texttt{Compress}_{K,\chi,k}(C)$ algorithm builds three subchains, the *unstable*, *uncompressed*, and *compressible* subchains from blockchain C. The *unstable* subchain Ω is made of the k most recent full blocks of the blockchain, i.e., $\Omega = C[-k:]$. The term unstable refers to the fact that the k most recent blocks have a non-negligible probability of being pruned during a fork resolution. The analytical dimensioning of the common prefix parameter k by Garay *et al.* [11] in a variable setting shows that k is proportional to the ratio m/τ, where m is the length of an epoch and τ is the dampening factor in Bitcoin's mining target recalculation [26] (see Section 5.1). A conservative evaluation gives $k = 323$ [11][4]. The *uncompressed* subchain X is made of the χ most recent full blocks of C after having removed the unstable subchain, i.e., $X = C[-\chi - k : -k]$. Parameter χ is set to $2m$ to guarantee that the uncompressed subchain contains at least the m blocks created during the second most recent blockchain epoch so that the difficulties of the uncompressed subchain blocks can be checked. Remark that by construction, $\Omega[0]$, i.e., the first block of Ω, points to $X[-1]$, i.e., the last block of X. Note that, in contrast to our construction, there is no need for an uncompressed part when dealing with static difficulties (e.g., [15, 19]). Finally, the *compressible* subchain $C^\star$ is made of all the headers of the blocks of C after having removed the unstable and uncompressed subchains, i.e., $C^\star = C[: -\chi - k]$. For each $\ell = 0, \ldots, \lfloor \log_2(|C^\star|) \rfloor$, the last $2K$ ℓ-blocks of subchain $C^\star$ are sampled. The highest level that contains $2K$ ℓ-blocks is called the level of the NIPoPoW.

Definition 4 (level of a NIPoPoW). *Let $C^\star$ be the compressible subchain of a PoW-based blockchain C. The level of the NIPoPoW of the underlying blockchain C is given by*

$$\texttt{Level(NIPoPoW)} \triangleq \max\left\{\ell \in \mathbb{N} \;\middle|\; |C^\star \uparrow^\ell| \geq 2K\right\}.$$

The secure dimensioning of parameter K appears in Section 6.4. Note that there are some ℓ'-blocks whose cardinal number is less than $2K$. Then all these ℓ'-blocks are also sampled and appear at level $\ell = \texttt{Level(NIPoPoW)}$. The genesis block, among others, appears at level ℓ. This guarantees that updating a proof with new blocks progressively leads to a proof of a higher level. Then for each lower level $\mu = \ell - 1, \ldots, 0$, the $2K$ most recent μ-blocks are sampled (i.e., the $2K$ blocks of $C^\star \uparrow^\mu [-2K:]$) as well as the μ-blocks coming after the K-th block of level $\mu + 1$ (i.e., the K blocks $C^\star \uparrow^\mu \{b^\star:\}$,

[4] It is interesting to see the difference between the conservative analysis of Garay *et al.* [11] which gives $k = 323$ and the value $k = 6$ commonly mentioned in informal discussions.

Input : C, which is either a regular or a compressed chain of blocks
Output: tuple $(\mathcal{D}, X, \Omega, \ell)$ where
Ω is the unstable subchain of C of size k,
X is the uncompressed subchain of C of size χ,
$\mathcal{D}$ is the compressed chain,
and ℓ is the highest level of C

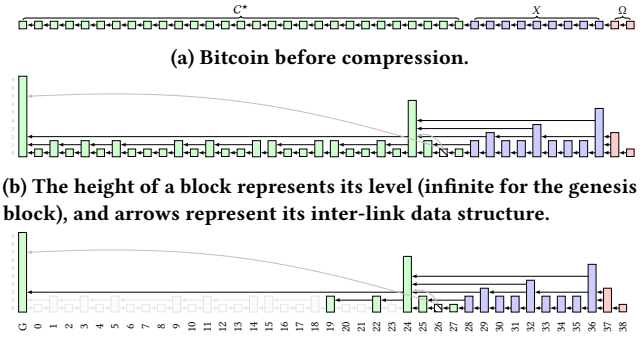
1 **function** $\texttt{Compress}_{K,\chi,k}(C)$:
2 $\quad\mathcal{D} \leftarrow \emptyset$
3 $\quad\Omega \leftarrow C[-k:]$ // Unstable subchain
4 $\quad X \leftarrow C[-\chi-k:-k]$ // Uncompressed subchain
5 $\quad C^\star \leftarrow C[:-\chi-k]$ // Compressible subchain
6 $\quad$**if** $|C^\star| \geq 2K$: // The chain is long enough
7 $\quad\quad\ell \leftarrow \max\{\mu : |C^\star \uparrow^\mu| \geq 2K\}$ // Level of the NIPoPoW
8 $\quad\quad\mathcal{D}[\ell] \leftarrow C^\star \uparrow^\ell$ // Keep all the ℓ-blocks
9 $\quad\quad$**for** $\mu \leftarrow \ell - 1$ **down to** 0: // For lower levels
10 $\quad\quad\quad b^\star \leftarrow C^\star \uparrow^{\mu+1} [-K]$ // Determine pivot block
11 $\quad\quad\quad\mathcal{D}[\mu] \leftarrow C^\star \uparrow^\mu [-2K:] \cup C^\star \uparrow^\mu \{b^\star:\}$ // Keep the $2K$ most recent μ-blocks plus all the most recent μ-blocks starting from $b^\star$
12 $\quad$**else:** // The chain to be compressed is too short
13 $\quad\quad\ell \leftarrow 0$
14 $\quad\quad\mathcal{D}[0] \leftarrow C^\star$
15 $\quad$**return** $(\mathcal{D}, X, \Omega, \ell)$

Algorithm 1: Prover algorithm.

where $b^\star = C^\star \uparrow^{\mu+1} [-K]$). We call $b^\star$ the *pivot* block as it ensures that any two consecutive levels of the construction intersect in at least K blocks. Furthermore, and as just described above, the header of any block b points back to the last block of level μ, $0 \leq \mu \leq \ell$, preceding b. This ensures that the $O(\log |C^\star|)$ sampled blocks form a subchain $\mathcal{D}$ of $C^\star$. Remark again that $X[0]$ points to $\mathcal{D}[0][-1]$. The $\texttt{Compress}_{K,\chi,k}(C)$ algorithm returns the headers of the blocks of $\mathcal{D}$, and the full blocks of X, Ω, and level ℓ. To check the validity of blocks in the unstable subchain Ω, the snapshot of block $X[-1]$ is attached to its header. The NIPoPoW Π is the subchain $\mathcal{D}\,X\,\Omega$. Figure 3 reflects the outcome of the compression algorithm run on the first 40 blocks of the Bitcoin blockchain (see Figure 3a). Security parameters are set to $K = 2$, $\chi = 9$, and $k = 2$. The unstable subchain Ω gathers the last $k = 2$ full blocks, the uncompressed subchain X gathers the next $\chi = 9$ full blocks, and the compressible subchain $C^\star$ contains all the other 29 block headers. Note that block header 36 comes with the snapshot of its block. Figure 3b depicts the very same subchain as Figure 3a by representing blocks as a function of their level and by showing their pointers to the last previous block at each level. Note that for legibility reasons, links from each block to the genesis block have been omitted. Finally, Figure 3c shows the result of the Compress() algorithm. The NIPoPoW built from the first 40 blocks of the Bitcoin blockchain is the subchain of blocks G 19 22 24 25 26 27 28 ... 36, where G, 19, 22, 24 and 25 are 1-blocks and 24, 25, 26 and 27 are 0-blocks. The level of the NIPoPoW is equal to 1 (see Definition 4).

5.3.3 Verifier algorithm. As briefly introduced in Section 1, before executing the Compare() algorithm, the verifier v broadcasts a Sync request[5] along with its private random number ν to synchronize with the system. Upon receipt of annotated Sync requests, a miner accumulates in the set V all the random numbers $\nu_i, \ldots, \nu_j$ it has received but which it has not yet inserted into the coinbase of the blocks it has created so far. Insertion into the coinbase can be achieved thanks to OP_RETURN scripts, for instance. When a prover u receives a new valid block b (i.e., b can be appended to u's locally maintained NIPoPoW Π), u executes the Compress() algorithm with the chain of blocks $\Pi\,b$ as input and broadcasts the updated NIPoPoW Π' to the system. Once v receives two annotated NIPoPoWs Π_1 and Π_2 (i.e., both Π_1 and Π_2 contain, respectively, a block b_1 and a block b_2 such that b_1's and b_2's coinbases contain ν), v executes the Compare() algorithm with Π_1 and Π_2 as inputs. Pseudo-code of the Compare() algorithm appears in Algorithm 2. The Compare() algorithm also accepts whole annotated blockchains, as its first step is to Compress() each of its inputs to be able to compare them on an equal basis. The $\texttt{Compare}_{K,\chi,k}()$ algorithm is parameterized by the same parameters as Compress().

Given two annotated NIPoPoWs, the algorithm looks for the smallest level μ for which both NIPoPoWs share a common block b. Block b is called the *latest common ancestor* (LCA) of both NIPoPoWs (see Lines 13–16). The existence or nonexistence of an LCA between any two NIPoPoWs provides crucial information on their respective underlying blockchains C_1 and C_2. Specifically, the presence of an LCA block allows the verifier to detect and therefore reject NIPoPoWs that result from low-difficulty attacks, and the absence of an LCA together with annotated coinbase allows a verifier to reject both obsolete NIPoPoWs and NIPoPoWs that result from brute-force low-difficulty attacks.



(a) Bitcoin before compression.

(b) The height of a block represents its level (infinite for the genesis block), and arrows represent its inter-link data structure.

(c) NIPoPoW of the first 40 blocks of Bitcoin.

Figure 3: Compression scheme on the first 40 blocks of Bitcoin ($K = 2$, $\chi = 9$, and $k = 2$).

Property 1 (Latest common ancestor (LCA)). *Let b be the LCA block of Π_1 and Π_2, where $\Pi_1 = \texttt{Compress}_{K,\chi,k}(C_1)$ and $\Pi_2 = \texttt{Compress}_{K,\chi,k}(C_2)$. Let μ be the level of block b. Then,*

- $\texttt{Level}(\Pi_1) = \texttt{Level}(\Pi_2)$, *i.e., both C_1 and C_2 have approximately the same number of blocks;*

[5] From an implementation point of view, this just requires leveraging the initial VERSION message [10] of the Bitcoin protocol and setting the VERSION's nonce field with a random number (in Bitcoin, the VERSION message implements the generic and classic Sync request).

Input : C_1, C_2, two blockchains or NIPoPoWs annotated with the verifier's random number ν
Output: tuple $(\mathcal{D}, X, \Omega, \ell)$ where
Ω is the unstable subchain of the accepted chain,
X is the uncompressed subchain of the accepted chain,
$\mathcal{D}$ is the compressed chain of the accepted chain,
and ℓ is the level of the accepted chain

1 **function** $\texttt{Compare}_{K,\chi,k}(C_1, C_2)$:
2 $(\mathcal{D}_1, X_1, \Omega_1, \ell_1) \leftarrow \texttt{Compress}_{K,\chi,k}(C_1)$
3 $(\mathcal{D}_2, X_2, \Omega_2, \ell_2) \leftarrow \texttt{Compress}_{K,\chi,k}(C_2)$
4 $M \leftarrow \{\mu \in \mathbb{N} \mid \mathcal{D}_1[\mu] \cap \mathcal{D}_2[\mu] \neq \emptyset\}$
5 **if** $M = \emptyset$:
6 // no block in common at the same level, i.e., no LCA block
7 $b_1 \leftarrow \texttt{findblock}(\mathcal{D}_1, X_1, \Omega_1, \nu)$
8 $b_2 \leftarrow \texttt{findblock}(\mathcal{D}_2, X_2, \Omega_2, \nu)$
9 **Wait** $|(\mathcal{D}_1 X_1 \Omega_1)\{b_1 :\}| = K \vee |(\mathcal{D}_2 X_2 \Omega_2)\{b_2 :\}| = K$
10 $(\mathcal{D}, X, \Omega, \ell) \leftarrow$ $\texttt{highdiff}((\mathcal{D}_1 X_1 \Omega_1)\{b_1 :\}, (\mathcal{D}_2 X_2 \Omega_2)\{b_2 :\})$
11 **else**:
12 $\mu \leftarrow \min M$
13 $b \leftarrow (\mathcal{D}_1[\mu] \cap \mathcal{D}_2[\mu])[-1]$ // b is the LCA block
14 $(\mathcal{D}, X, \Omega, \ell) \leftarrow$
15 $\texttt{highdiff}(\ (\mathcal{D}_1\ \{b :\}\ X_1\ \Omega_1)\ \uparrow^{\mu}, (\mathcal{D}_2\ \{b :\}\ X_2\ \Omega_2)\ \uparrow^{\mu})$
16 **return** $(\mathcal{D}, X, \Omega, \ell)$

Algorithm 2: Verifier algorithm.

- *If $\mu = 0$, both C_1 and C_2 are almost identical, with the exception of their at most $2K + \chi + k$ most recent blocks.*

We have the following lemma. Intuitively, the lemma says that if two NIPoPoWs Π_1 and Π_2 share an LCA block b at level $\mu > 0$, then b is followed by at least K blocks in either Π_1 or Π_2.

Lemma 5.1. *Let b be the LCA block of Π_1 and Π_2, where $\Pi_1 = \texttt{Compress}_{K,\chi,k}(C_1)$ and $\Pi_2 = \texttt{Compress}_{K,\chi,k}(C_2)$. Let μ be the level of block b. If $\mu > 0$, then $|C_1\{b :\}\uparrow^{\mu}| > K \vee |C_2\{b :\}\uparrow^{\mu}| > K$.*

Proof. By minimality of μ, $(\Pi_1 \cap \Pi_2)\uparrow^{\mu-1} = \emptyset$ (Line 12 of Algorithm 2). Suppose by contradiction that neither Π_1 nor Π_2 have K blocks after their LCA block b. Block b therefore belongs to the last K blocks of level μ for both proofs. By construction (Line 11 of Algorithm 1), the last K blocks of a proof are present at level $\mu - 1$. Block b is present at level $\mu - 1$ of both Π_1 and Π_2, which is impossible by the minimality of μ. This completes the proof. □

If both NIPoPoWs Π_1 and Π_2 share an LCA block b (Lines 13–16), then function `highdiff` is called with the subchains of Π (resp. Π_2) restricted to the μ-blocks that follow the μ-block b. Function `highdiff` returns the subchain that has accumulated the largest quantity of difficulty within its at least K μ-blocks after the μ-block b. Theorem 6.9 shows that even if the adversarial blockchain contains as many μ-blocks as the honest one, their accumulated mining difficulty cannot exceed the honest one. Thus, the adversarial prover cannot convince the verifier by presenting a NIPoPoW built from a blockchain that has been created during a low-difficulty attack.

On the other hand, if Π_1 and Π_2 do not share an LCA block (Lines 6–10), then verifier v searches for the annotated blocks b_1 and b_2 in Π_1 and Π_2, respectively (Lines 7 and 8). By construction, blocks b_1 and b_2 are recent blocks (they belong to subchains $X_1\ \Omega_1$ and $X_2\ \Omega_2$, respectively) and have been mined at approximately the same time. Verifier v can therefore use them as a temporal anchor from which the accumulated difficulty of subsequent blocks can be checked exactly as in the Bitcoin protocol. Specifically, if both b_1 and b_2 are less than K deep from the tips of Π_1 and Π_2, respectively, then v observes the system and waits until b_1 or b_2 are followed by K blocks (i.e., $|(\mathcal{D}_1 X_1 \Omega_1)\{b_1 :\}| = K \vee |(\mathcal{D}_2 X_2 \Omega_2)\{b_2 :\}| = K$, Line 9). Once this condition is verified, v accepts the NIPoPoW returned by function `highdiff` provided with the chains of blocks that respectively follow b_1 and b_2. By construction, this is the subchain that has accumulated the largest quantity of difficulty. Theorem 6.9 shows that this enables the verifier to reject NIPoPoWs that have been generated from a blockchain created during a brute-force low-difficulty attack.

6 Analysis

6.1 Preliminary definitions

Our analysis relies on Garay *et al.*'s Bitcoin Backbone with chains of variable difficulty model [11], as well as concepts introduced in Kiayias *et al.*'s protocol [19]. In particular, it strongly relies on Garay *et al.*'s notion of typical executions [11]. An execution E of the protocol generates a sequence S of rounds. An execution E is typical if the considered random variables do not deviate too much from their expected values, i.e., at a distance ε of their expected value. From Garay *et al.* [11] we have $|S| > m/(16\tau f)$, where m represents the number of blocks defining an epoch ($m = 2016$ for Bitcoin). Honest parties have a probability f to create at least one block in a round, and τ is the dampening filter in Bitcoin target's recalculation function. A round r is successful if the honest parties succeed in creating at least one block during r, and is uniquely successful if the honest parties succeed in creating exactly one block during r. Consider some block b that extends a chain C. We say that block b contains the point of difficulty d, $d \in \mathbb{R}^+$, if $\|C\| \leq d < \|C\ b\|$ [11]. Recall that the notation $\|C\|$ represents C's difficulty. We denote by Q_r the random variable that represents the difficulty of the block created during a uniquely successful round r. If r is not uniquely successful, then $Q_r = 0$. By extension, $\sum_{r \in S} Q_r$ represents the accumulated difficulty obtained during an execution of S rounds. The quantity $\sum_{j \in J} A_j$ represents the accumulated difficulty obtained by the adversarial parties over a set of queries J. Considering a set of queries rather than a set of rounds allows the adversary to target specific rounds during which it queries the hash function. Finally, we denote by ϕ the quantity $\rho/2^{\kappa}$. The full version of this paper provides the formalization of all the notions we use in our proofs [25]. More details can be directly found in Garay *et al.* [11]. Table 2 contains a summary of the variables used in the analysis.

6.2 Suppression of honest difficulty

The following technical lemmas will be deeply used to prove the correctness of our solution. The first two show the conditions under which the adversary may impose its chain on honest parties. This can be achieved in one of two ways. Either the adversarial

Table 2: Summary of used variables, their domain of definition, and their meaning

	Domain of definition	Description
n_r	$\forall r : n_r \in \mathbb{N}$	Number of total parties in round r.
t_r	$\forall r : t_r \in \mathbb{N}$	Number of adversarial parties in round r.
ρ	$\rho \in \mathbb{N}$	Number of queries of each party.
ϕ	$\phi \in \mathbb{Q}_{\geq 0}$	Convenience notation. $\phi = \rho/2^\kappa$.
δ	$\delta \in (0,1)$	Advantage of honest parties. For any round r, $t_r \leq (1-\delta)(n_r - t_r)$.
m	$m \in \mathbb{N}$	Length of an epoch in blocks, $m = 2016$ for Bitcoin.
f	$f \in (0,1)$	Target probability that at least one honest party mines a block in any round r.
τ	$\tau \in \mathbb{R}$	Dampening filter, $\tau = 4$ for Bitcoin.
(γ, s)	$\gamma \in \mathbb{R}, s \in \mathbb{N}$	Bound on variation of the number of parties.
(η, θ)	$\eta \in (0,1], \theta \in [1, \infty)$	Lower and upper bound on f.
ε	$\varepsilon \in (0,1)$	Quality of concentration of random variables in typical executions.
Q_r	$Q_r \in \mathbb{R}$	Difficulty of an honest block mined in a uniquely successful round r.
D_r	$D_r \in \mathbb{R}$	Largest block difficulty among honest blocks mined in round r.
A_j	$A_j \in \mathbb{R}$	Difficulty of an adversarial block mined in query j.
κ	$\kappa \in \mathbb{N}$	Security parameter that represents the number of bits of the output of the hash function. $\kappa = 256$ for Bitcoin.
k	$k \in \mathbb{N}$	Common prefix parameter. It represents the number of blocks of the unstable subchain of a NIPoPoW.
χ	$\chi \in \mathbb{N}$	Number of blocks of the uncompressed subchain of a NIPoPoW.
K	$K \in \mathbb{N}$	Security sampling parameter.
ℓ	$\ell \in \mathbb{N}$	Typically represents the level of a block or the level of a proof.
α	$\alpha \in \mathbb{R}$	Ratio between the block difficulty h of the honest parties and the one a of the adversarial parties.
p	$p \in [0,1]$	Fraction of honest computing power. We have $p = \frac{2}{3}$.
q	$q \in [0,1]$	Fraction of adversarial computing power. We have $q = \frac{1}{3}$.

miner aims at creating more difficulty in its own chain by concentrating uniquely on its own chain or tries to bias the distribution of block levels in the honest chain by "suppressing" (i.e., forking) well-chosen blocks to replace them with its own blocks, whose levels are different.

On the other hand, Theorem 6.2 states that in executions long enough, the honest difficulty of the adopted blockchain dominates the adversarial one. Proofs of these lemmas extend the ones presented in Kiayias *et al.* [19]. Observe that in the constant mining difficulty setting [19], the height of a blockchain (i.e., its number of blocks) is equivalent to its accumulated difficulty, as all the blocks are created with the same difficulty. In the variable mining difficulty setting, there is a difference between the height of a blockchain and its accumulated difficulty, and so to compare C and C', we need to consider the different possible relationships between the height of a chain and its associated difficulty.

Observation 1 (Pairing [11]). *Consider an execution where a valid block b has been created by an honest party at a uniquely successful round, such that b is appended to chain C. Let $d \in \mathbb{R}^+$ such that $\|C\| \leq d < \|Cb\|$. Then if there exists a chain $C'b'$ with $b \neq b'$ such that $\|C'\| \leq d < \|C'b'\|$ then b' has been created by the adversary.*

Indeed, let r be the uniquely successful round at which b was created. We show that no honest party would have created b' at round r'. Due to the synchronous setting, every new block created and sent at round r is received by any honest party during round r. Consider $r' > r$. Since honest parties are aware of Cb at round r and, by definition of d, $\|C\| \leq d < \|Cb\|$, then no honest party would have created b' such that b' extends C' at any later round r'. Consider $r' < r$. If an honest party had created b' at round $r' < r$, then any honest party would not have created b at round r as $\|C\| < \|C'b'\|$. Finally, $r \neq r'$ because r is uniquely successful.

Lemma 6.1 (Suppression). *If r is a uniquely successful round and the corresponding block b does not belong to the chain of an honest party at a later round, then there is a set of consecutive rounds S and a set J of adversarial queries in S such that $r \in S$ and $\sum_{r \in S} Q_r < \sum_{j \in J} A_j$.*

Proof. Consider an execution in which an honest party creates b such that b is valid for C. By assumption of the lemma, this occurs during a uniquely successful round, say r, so b is appended to C. Let C' be the chain adopted by an honest party at a later round r_1 such that b does not belong to C'. Let b_0 be the last honest block that belongs to the common prefix of C and C', and r_0 the round at which b_0 was created. If b_0 is the genesis block, then $r_0 = 0$. Let S be a sequence of consecutive rounds $S = \{r' : r_0 < r' \leq r_1\}$. We show that the set of adversarial queries J exhibits more difficulty than the ones of the honest parties during S. We consider two cases.

- Case (1): We suppose that r_1 is the first round in sequence S at which an honest party adopts C' such that $b \notin C'$. This means that all the difficulty accumulated on C' during S has been contributed by the adversarial queries (by Observation 1). At round r_1, an honest party adopts C'. This means that the difficulty accumulated in the blocks of C during S is

strictly smaller than the one accumulated in C''s blocks, i.e., $\sum_{r\in S} Q_r < \sum_{j\in J} A_j$. This completes the case.

- Case (2): We now suppose that an honest party has contributed at least once to C' before round r. We show that even in these scenarios, the adversarial contribution is larger than the sum of the honest contributions (in uniquely successful rounds). Let $S' = \{u_1, v_1, u_2, \ldots, u_n, v_n\}$ be a sequence of rounds such that $u_1 = r_0+1$, $v_n = r_1$ and for all $i = 1, \ldots, n-1$, $u_{i+1} = v_i + 1$. Each subsequence of rounds $\{u_i, v_i\}$ represents the rounds during which an honest party mines on chain $C^\star$ with either $C^\star = C$ or $C^\star = C'$. Each subsequence $\{u_i, v_i\}$ is maximal. Observe that $S = S'$. In the following, we call the subsequence of consecutive rounds $\{u_i, v_i\}$, with $i \in 1, \ldots, n$ a *hook*. Observe that each hook represents exactly the situation described in case (1). Specifically, if during the i-th hook $\{u_i, v_i\}$ the honest party mines on $C^\star = C$, then all the difficulty accumulated on C' during the i-th hook $\{u_i, v_i\}$ has been contributed by the adversarial queries. Case (1) applies, i.e., $\sum_{r\in\{u_i,v_i\}} Q_r < \sum_{j\in J} A_j$, with $J \in \{u_i, v_i\}$. At round $u_{i+1} = v_i + 1$, the $(i+1)$-th hook takes place, the honest party now mines on $C^\star = C'$, and all the difficulty accumulated on C during the $(i+1)$-th hook is contributed by the adversarial queries. Case (1) applies, i.e., $\sum_{r\in\{u_{i+1},v_{i+1}\}} Q_r < \sum_{j\in J} A_j$, with $J \in \{u_{i+1}, v_{i+1}\}$. The same argument applies for all the hooks of S', which covers all the rounds of S. By an accumulation argument, this case completes the proof.

□

Lemma 6.2 (Unsuppressibility). *Given a typical execution with an honest party and an adversary, every set of consecutive rounds U has a subset S of uniquely successful rounds such that the following conditions hold:*

(1) $\sum_{r\in S} Q_r \geq \sum_{r\in U} Q_r - 2\sum_{j\in U} A_j - 2(1-\varepsilon)(1-\theta f)\phi \sum_{r\in J}(n_r - t_r)$, where J indexes the queries of the adversary in a set of $m/(16\tau f)$ rounds;

(2) 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 exactly U and the $m/16\tau f$ rounds that come before and after U. Taking the contrapositive of Theorem 6.1, we build S such that S contains all those uniquely successful rounds $r \in U$ during which blocks were not suppressed by the adversary, i.e., such that for any set of consecutive rounds $S' \subseteq U'$ containing r, $\sum_{r\in S} Q_r > \sum_{j\in J'} A_j$, where J' indexes the queries of the adversary in S'. Note that in a (η, θ)-good typical execution and (γ, s)-respecting environment, such S' may not contain elements outside U'.

We need to prove the following statement: $\sum_{r\in U} Q_r - \sum_{r\in S} Q_r \leq 2\sum_{j\in U} A_j + 2(1-\varepsilon)(1-\theta f)\phi \sum_{r\in J}(n_r - t_r)$. We focus on the uniquely successful rounds in U during which the blocks were suppressed by the adversary, i.e., the rounds not in S. Consider a collection $\mathcal{T}$ of sets T of consecutive rounds with the following properties.

- $\forall T \in \mathcal{T}, \sum_{r\in T} Q_r \leq \sum_{j\in J} A_j$, where J indexes the queries of the adversary in T;
- $\forall r \in U \setminus S$, there is a set of rounds $T \in \mathcal{T}$ that contains r;
- $|\mathcal{T}|$ is minimum among all collections with the above properties.

Observe that by the minimality condition on $\mathcal{T}$, no round r with $A_r > 0$ 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, by construction of U', no round in $U' \setminus U$ belongs to more than one set of rounds T. Thus,

$$\begin{aligned}\sum_{r\in U} Q_r - \sum_{r\in S} Q_r = \sum_{r\in U\setminus S} Q_r &\leq \sum_{T\in\mathcal{T}}\sum_{r\in T} Q_r \\ &\leq \sum_{T\in\mathcal{T}}\sum_{j\in J_1} A_j \leq 2\sum_{j\in J_2} A_j + \sum_{j\in J_3} A_j,\end{aligned}$$

where J_1 (resp. J_2) indexes the adversarial requests in set T (resp. U), and J_3 those in set $U' \setminus U$. The third inequality holds because every round in which the adversary was successful is counted at most twice inside U and at most once in $U' \setminus U$. Finally, using $|U' \setminus U| = 2m/16\tau f$ and $\sum_{j\in J} A_j < (1-\varepsilon)(1-\theta f)\phi \sum_{r\in J_3}(n_r - t_r)$, we obtain the stated bound.

By assumption, the adversary is 1/3-bounded, which guarantees that in any typical execution, in every set of consecutive rounds, there exists a positive quantity of honest difficulty that can never be suppressed by the adversary. This completes the proof. □

6.3 Proofs of the succinctness, completeness, and onlineness properties

Intuitively, the ℓ-block Common-Prefix lemma shows that while the difficulty accumulated by the ℓ-blocks of the adversarial blockchain for a given period of time can be greater than the one accumulated by the honest ℓ-blocks, at some point the honest parties will accumulate more difficulty than what the adversary can ever achieve.

Lemma 6.3 (ℓ-block Common-Prefix). *Given a typical execution with a 1/3-bounded PPT adversary, such that blockchain C is maintained by the honest parties at round r, and there exists another blockchain C' such that at round r the ℓ-blocks of $C' \setminus (C \cap C')$ have accumulated at least $D^{\mathcal{A}}_{\min}(\ell) = (1+\varepsilon)\xi_\ell\phi\gamma t K_{\mathcal{A}}/f$ quantity difficulty, then with overwhelming probability, we have $\|C \uparrow^\ell\| > \|C' \uparrow^\ell\|$, where $\max_{r\in S} t_r \leq \gamma t$.*

Proof. Consider a typical execution E that satisfies the assumptions of the lemma. We observe the execution at round r. Let r^* be the round during which the last honest block b^* on $C^\star = C \cap C'$ was computed. If no such block exists, we set $r^* = 0$. Consider the sequence of rounds $S = \{i : r^* < i \leq r\}$ of the typical execution E. Because E is typical, we can apply Lemma 6.2. By this lemma, the quantity of honest difficulty accumulated on $C \setminus C^\star$ is strictly greater than 0. Furthermore, Lemma 6.2 imposes no constraints on the positions of the unsuppressed honest blocks within the execution. So we can split the sequence S into two subsequences $S_1 = \{i : r^* < i \leq r^{*\prime}\}$ and $S_2 = \{i : r^{*\prime} < i \leq r\}$, such that S_1 is the sequence of rounds during which all the honest blocks have been suppressed by adversarial requests. I.e., round $r^{*\prime}$ is the last round beyond which adversarial queries cannot suppress honest blocks. Now, by Lemma 6.2, S_2 is non-empty. The cumulated difficulty on $C \setminus C^\star$ is minimal if, during S_2, the adversary does not contribute to adversarial difficulty on it.

In Section 6.4.1, through a competition between an honest team and an adversarial one, we analyze the maximal number of blocks $K_{\mathcal{A}}$ the adversarial team can produce such that from this $K_{\mathcal{A}}$-th block onward the adversarial team will never be able to catch up to the sequence of blocks concurrently produced by the honest team. The number of rounds needed to produce those $K_{\mathcal{A}}$ blocks is $K_{\mathcal{A}}/f$, where f is the probability to mine a block during one round. Thus, round $r^{*\prime}$ in sequence S_1 represents exactly the $K_{\mathcal{A}}/f$-th from which the adversarial chain cannot accumulate more difficulty than what can be achieved by the honest one. Specifically, the quantity of difficulty accumulated on $C' \setminus C^{\star}$ during S_1 is equal to $\sum_{j \in J_1} A_j$, where J_1 indexes all the adversarial queries in the sequence S_1. Each of the rounds that contribute to that quantity of difficulty produces an ℓ-block with probability $\xi_\ell = 1/2^\ell$. Thus, with overwhelming probability, the quantity of difficulty of ℓ-blocks on $C' \setminus C^*$ during S_1 is equal to $\xi_\ell \sum_{j \in J_1} A_j < (1+\varepsilon)\xi_\ell \phi \sum_{r \in S_1} t_r \leq (1+\varepsilon)\xi_\ell \phi \gamma t K_{\mathcal{A}}/f$, where $\max_{r \in S} t_r \leq \gamma t$ in a (γ, s)-respecting execution. The quantity of difficulty accumulated on C minus the one accumulated on C' during S is equal to the one accumulated on C minus the one accumulated on C' during S_2. This quantity of difficulty is strictly greater than 0 by Lemma 6.2 and by the above argumentation. This completes the proof. □

Corollary 6.4 (Exponential decay of ℓ-block difficulty threshold). *Quantity $D^{\mathcal{A}}_{\min}(\ell)$ decays exponentially with level ℓ.*

Proof. From our model, the variables $\varepsilon, \phi, \gamma$, and f are constants. The term $K^{\mathcal{A}}_{\min}$ is a bound that uniquely depends on the relative computational power of the parties and ε. The term t is bounded within an execution. Thus the main variable affecting the term is ξ_ℓ. By definition of a block level (see Definition 3), ξ_ℓ decreases exponentially with ℓ. This completes the proof of the corollary. □

By Corollary 6.4, $D^{\mathcal{A}}_{\min}(0)$ is the highest quantity of difficulty required to determine the honest blockchain.

Theorem 6.5 (Succinctness). *For any infinite sequence S of rounds and for any subsequence $S_j \subseteq S$ of consecutive rounds, any honest miner stores $O(K^2 \log(r))$ blocks at round $r \in S_j$.*

Proof. Consider a proof Π generated by an honest prover from an underlying chain C, and suppose, in contradiction, that $|\Pi| \in \omega(K \log(r))$. Consider $(\mathcal{D}, X, \Omega, \ell) = \texttt{Compress}(C)$. By assumption on Π, we have $\sum_{\mu \in \mathbb{N}} |\mathcal{D}[\mu]| \in \omega(K \log(r))$, since χ, k are constant natural integers. Let $\ell = \max\{\mu \in \mathbb{N} \mid \mathcal{D}[\mu] \neq \emptyset\}$, and let us consider some $\mu \leq \ell$ such that $\mathcal{D}[\mu] \in \omega(K \log(r))$ and thus $\mathcal{D}[\mu] \in \Omega(K^2 \log(r))$. Note r_0, r_1 the rounds in which $\mathcal{D}[\mu][0], \mathcal{D}[\mu][-1]$ were created, respectively. Consider U the set of consecutive rounds between r_0 and r_1. We have $|U| \geq |\mathcal{D}[\mu]| \geq K$. By definition of μ, at least K^2 blocks must have been created during U, among which $(1-\varepsilon)K^2/2 \geq 2K$ are $(\mu+1)$-blocks, which belong to $\mathcal{D}[\mu+1]$ as well. The case $\mu = \ell$ contradicts the maximality of ℓ, and thus the assumption $\Pi \in \omega(K \log(r))$. If $\mu < \ell$, then we have $\mathcal{D}[\mu] \in \Omega(K^2 \log(r))$, and $\mathcal{D}[\mu+1] \in O(K)$. Since $\mathcal{D}[\mu] = C[:-\chi-k] \uparrow^{\mu} [-2K:] \cup C[:-\chi-k] \uparrow^{\mu} \{\mathcal{D}[\mu+1][-K:]\}$ and $|C[:-\chi-k] \uparrow^{\mu} [-2K:]| = 2K$, we have $C[:-\chi-k] \uparrow^{\mu} \{\mathcal{D}[\mu+1][-K:]\} \in \Omega(K^2 \log(r))$. Since $\mathcal{D}[\mu+1] \in O(K)$, $|\mathcal{D}[\mu]| \leq 2K + O(K)$, which contradicts $\mathcal{D}[\mu] \in \Omega(K^2 \log(r))$, completing the proof. □

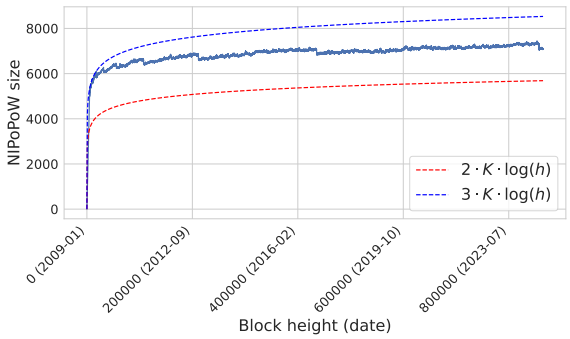


Figure 4: Evolution of the number of blocks kept in the Bitcoin NIPoPoW as a function of the number of blocks in the Bitcoin blockchain. Setting: $K = 208$, $\chi = 4032$, and $k = 323$.

On October 11, 2024, the Bitcoin Mainnet blockchain consisted of 865,042 blocks. Figure 4 shows the size of Bitcoin NIPoPoW upon receipt of a new block. Its size varies polylogarithmically with the size of Bitcoin's blockchain, which illustrates the succinctness property. Analysis of parameter K is detailed in Section 6.4.

Theorem 6.6 (Completeness:). *For any blockchain C maintained by an honest party and any block b valid for C, let $\Pi = \texttt{Compress}(C)$. Then $\texttt{Compress}(C\ b) = \texttt{Compress}(\Pi\ b)$.*

Proof. First block b will be included in both proofs at the end of the unstable subchain. Now, for all the other ℓ-blocks that are sampled from $C\ b$, they already appear in Π at the right level ℓ of $\mathcal{D}$ and will be shifted by one position on the left if the first block of the uncompressed subchain is an ℓ-block, making the former ℓ-block irrelevant for the new proof. In any case, the new block b will not require sampling blocks different from b that were not previously in Π. This completes the proof of the theorem. □

Theorem 6.7 (Onlineness). *For any blockchain C maintained by an honest party, any block b valid for C, let $\Pi = \texttt{Compress}_{K,\chi,k}(C)$. We have $\texttt{Compress}_{K,\chi,k}(C\ b) = \texttt{Compress}_{K,\chi,k}(\Pi\ b)$.*

Proof. By Algorithm 5.3.2, $\Pi \subseteq C$, and the snapshot of $C[-k-1]$ is equal to the snapshot of $\Pi[-k-1]$. By construction of C, once all the blocks of $C[-k:]$ have been validated based on the snapshot of $C[-k-1]$, the snapshot of $C[-1]$ represents the state of C. By Algorithm 5.3.2, $C[-k:] = \Pi[-k:]$. Thus the snapshot of $\Pi[-1]$ represents the state of Π and the state of C. This completes the proof of the theorem. □

Figure 5 shows the latency of an update on the Bitcoin NIPoPoW, that is, the time that elapses, at the prover, between the reception of a new block and the time the NIPoPoW is updated with this new block. This plot illustrates how much a prover needs to "pay" to build a NIPoPoW. The median time to update the proof is 2.73 ms. The time to update the proof varies logarithmically with the number of blocks. Note that the latency of an update decreases each time the level of the NIPoPoW increases. This can be observed on the lower part of the plot. This comes from the fact that when the level of the NIPoPoW increases, say from ℓ to $\ell+1$, all the $\ell+1$-blocks that were at level ℓ of the NIPoPoW are now sufficiently numerous

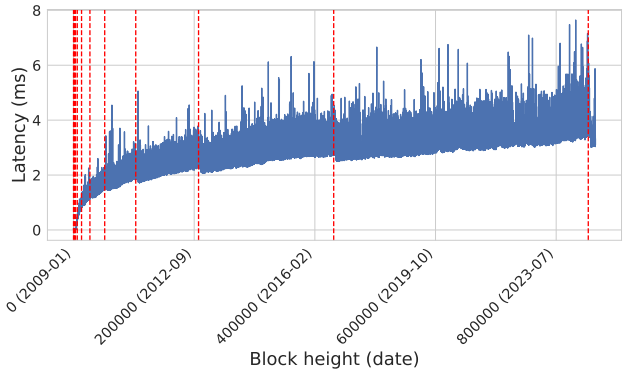


Figure 5: Latency of the NIPoPoW generation as a function of the number of added blocks. Vertical bars represent the time instants at which the NIPoPoW level changes. Setting: $K = 208$, $\chi = 4032$**, and** $k = 323$**.**

(i.e., their cardinal number reaches $2K$) to belong to the new level of the NIPoPoW (i.e., $\ell + 1$). Thus it takes less time for the prover to update the NIPoPoW when it receives an ℓ or $\ell + 1$ block.

6.4 Proof of the security property

The properties of NIPoPoW in a variable setting rely on the security parameters K, χ, and k. A conservative evaluation of the common prefix parameter k gives $k = 323$ (see Section 5.3.2), and the incompressibility parameter χ is set to $2m$ so that verifiers can verify block mining difficulties of the most recent blocks of a NIPoPoW, in particular to detect brute-force low-difficulty attacks (see Section 5.3.3). The security parameter K must be set so that if the verifier receives two NIPoPoWs, Π and Π', where $\Pi = \texttt{Compress}(C)$, with C the honest blockchain, and Π' is presented by an adversarial prover, then with overwhelming probability, starting from the last common ℓ-block, for $\ell \geq 0$, the last $2K$ sampled ℓ-blocks are sufficient to convince the verifier to accept Π.

Let C be the honest blockchain such that at some point in the execution the adversarial miner forks C at ℓ-block $b^\star$ to mine its own blockchain C'. The remainder of the section aims at finding $K_{\mathcal{H}}$ and $K_{\mathcal{A}}$. $K_{\mathcal{H}}$ represents the number of blocks in $C \setminus (C \cap C')$ such that $\|C \setminus (C \cap C')\| > \|C' \setminus (C \cap C')\|$ with overwhelming probability. Similarly, $K_{\mathcal{A}}$ represents the number of blocks in $C' \setminus (C \cap C')$ such that $\|C \setminus (C \cap C')\| > \|C' \setminus (C \cap C')\|$ with overwhelming probability. The security parameter K will therefore be set to the maximum of both $K_{\mathcal{A}}$ and $K_{\mathcal{H}}$. We detail the computation of $K_{\mathcal{H}}$. The argument for the dimensioning of $K_{\mathcal{A}}$ is similar.

6.4.1 Dimensioning of parameter K. We model the mining process as a two-phase competition between two teams, the adversarial team and the honest nodes team. Both teams start at their last common block, $b^\star$. In the first phase, both the honest team and the adversary create their own blocks until the honest team has created n blocks. In the second phase, both teams continue to create their own blocks until either the adversarial team catches up and exceeds the accumulated quantity of difficulty of the honest team, in which case it wins the competition, or else the adversarial team is so far behind that it is hopeless for it to ever catch up to the quantity of difficulty of the honest blockchain. We assume that the competition takes place within an epoch, which means that the honest target T is constant during the competition. We assume that the adversary can mine its own blocks with a mining difficulty different from that of the honest team. Let $\alpha = h/a$, where h (resp. a) is the difficulty of one honest (resp. adversarial) block. We denote by C_n^α the event that the adversarial subchain $C' \setminus (C \cap C')$ catches up with the honest subchain $C \setminus (C \cap C')$ after n blocks have been appended by the honest team to block $b^\star$.

We determine the probability $\mathbb{P}\{C_n^\alpha\}$ by combining the gambler's ruin problem [18] and the Poisson distribution. The gambler's ruin problem computes $P_{ruin}(M)$, the probability that the adversary will catch up to the honest team, given an initial gap M, where M is the difference between the accumulated difficulty on the honest subchain and the one on the adversarial subchain i.e., $M = \|C \setminus (C \cap C')\| - \|C' \setminus (C \cap C')\|$. Assuming that block creations happen with a known average rate and independently of the time since the last block creation, we can use the Poisson distribution to compute the probability that the adversary will be able to create a given number of blocks during a fixed interval of time.

We model the evolution of the gambler's payoff along the game by a homogeneous discrete-time Markov chain $X = \{X_k, k \geq 0\}$, where $M \in \mathbb{N}$ is the initial wealth of the gambler ($X_0 = M$) with

$$\mathbb{P}\{X_k = j\} = p_j, \quad -v \leq j < \infty,$$

where v is the maximum possible loss with the assumption that $p_{-v} \neq 0$. The gambler must stop playing when his wealth is less than v, in which case the gambler is ruined. The probability of ruin $P_{ruin}(M)$ depends on the payoff distribution $\{p_j\}_{-v \leq 0 < \infty}$ and on the initial wealth M. We know that $P_{ruin}(M) = 1$ if the expected value of X is non-positive, so we assume that

$$\mathbb{E}(X) = \sum_{k=-v}^{\infty} k p_k > 0.$$

We assume that $M \geq v$, otherwise the gambler is already ruined and $\forall M < v, P_{ruin}(M) = 1$ (see [18]). We define the generating function $p(z)$ as

$$p(z) = \sum_{k=-v}^{\infty} p_k z^k. \tag{1}$$

The analogy between the mining process competition and the gambler's ruin problem is as follows. At each step k of the competition, $X_k = -a$ with $\mathbb{P}\{X_k = -a\} = p_{-a}$ and the gap is reduced by a, or $X_k = h$ with $\mathbb{P}\{X_k = h\} = p_h$ and the gap is increased by h. We have $v = a$, and Equation (1) becomes

$$p(z) = p_{-a} z^{-a} + p_h z^h. \tag{2}$$

Let p (resp. q) be the mining power of the honest team (resp. adversary) such that $p + q = 1$. When the adversary mines a block with target αT, p_{-a} represents the probability that the next block is mined by the adversary. Similarly, p_h represents the probability that the next block is mined by the honest team. We consider a time interval during which z blocks are mined with target T (i.e., when $\alpha = 1$) are created. Among the z blocks, qz of them have been created by the adversary and pz ones by the honest team. Now if the adversary mines with target αT, the adversary will mine αqz blocks during the very same interval so that the total number of

blocks during the time interval will no longer be z but $\alpha qz + pz$, where αqz is the number of blocks belonging to the adversary and pz to the honest team. Finally, determining p_a and p_{-b} comes down to calculating the proportion of blocks belonging to, respectively, the adversarial and honest subchains among the $\alpha qz + pz$ blocks mined in total during the considered time interval. We have

$$p_{-a} = \frac{\alpha qz}{pz + \alpha qz} = \frac{q\alpha}{p + q\alpha} \quad \text{and} \quad p_h = \frac{pz}{pz + \alpha qz} = \frac{p}{p + q\alpha}.$$

For integers $n > 0$ and $r \geq 0$, the complete symmetric polynomial of order r in the variables $z_1, \ldots, z_n$ is defined as the sum of all products of the variables $z_1, \ldots, z_n$ of degree r, that is:

$$\Phi_{n,r}(z_1, \ldots, z_n) = \sum_{\substack{i_j \geq 0, \\ i_1 + \cdots + i_n = r}} \prod_{j=1}^{n} z_j^{i_j}.$$

Theorem 6.8 gives the probability $P_{ruin}(M)$ that the adversary catches up with the honest subchain given the initial gap $M \geq \nu$.

Theorem 6.8 ([18]). *Equation $p(z) = 1$ has ν solutions (counting multiplicities) in the unit disk $|z| < 1$ of the complex plane, which we denote as η_j (for $1 \leq j \leq \nu$). The probability of ruin is given by*

$$P_{ruin}(M) = \sum_{n=1}^{\nu} \Phi_{n,M-n+1}(\eta_1, \ldots, \eta_n) \prod_{j=1}^{n-1} (1 - \eta_j).$$

When the roots $\eta_1, \ldots, \eta_\nu$ are distinct, we can use the following alternative expression:

$$P_{ruin}(M) = \sum_{j=1}^{\nu} \eta_j^M \prod_{\substack{i=1 \\ i \neq j}}^{\nu} \frac{1 - \eta_i}{\eta_j - \eta_i}.$$

We combine the gambler's ruin problem with the Poisson distribution inspired by [28] to derive $\mathbb{P}\{C_n^\alpha\}$. We consider all possible initial gaps between the adversary subchain and the honest team when the honest team has produced n blocks in $C \setminus (C \cap C')$. Specifically, for $i \geq 0$, we set $M_i = nh - ia$, which represents the gap when the adversary has produced i blocks during the time the honest team has produced n blocks. Let λ represent the average number of blocks produced by the adversary during the time interval I_n, where I_n is the period of time during which the honest team produces n blocks. When the adversary mines blocks with target T, $z = qz + pz$ represents the total number of blocks mined during a time interval J_z, where J_z represents the time interval during which z blocks have been mined by both parties when the adversary mines with honest target T (i.e., $\alpha = 1$). Recall that qz represents the number of adversarial blocks and pz the number of blocks of the honest team. When the adversary mines with target αT ($\alpha \neq 1$), the total number of blocks mined during J_z becomes $\alpha qz + pz$. Consequently, to mine n blocks, the corresponding time interval I_n must satisfy

$$I_n = \frac{n}{p\,z} J_z.$$

The adversary produces αqz blocks during J_z, therefore

$$\lambda = \frac{\alpha qz\, I_n}{J_z} = \frac{\alpha q\, n}{p}.$$

The number of blocks X produced by the adversary during I_n follows a Poisson distribution with parameter λ. The set $\{\{X = i; \lambda\}; i \geq 0\}$ is a complete system of events with $\mathbb{P}\{X = i; \lambda\} \neq 0$ and $P_{ruin}(M_i) = \mathbb{P}\{C_n^\alpha \mid \{X = i; \lambda\}\}$. The total probability $\mathbb{P}\{C_n^\alpha\}$ for the adversary to catch up is given by

$$\begin{aligned}
\mathbb{P}\{C_n^\alpha\} &= \sum_{i=0}^{\infty} \mathbb{P}\{X = i; \lambda\} \mathbb{P}\{C_n^\alpha \mid \{X = i; \lambda\}\} \\
&= \sum_{i=0}^{\infty} \mathbb{P}\{X = i; \lambda\} P_{ruin}(M_i) \\
&= \sum_{i=0}^{\lfloor n\alpha - 1 \rfloor} \frac{\lambda^i e^{-\lambda}}{i!} P_{ruin}(M_i) + \sum_{i=\lfloor n\alpha - 1 \rfloor + 1}^{\infty} \frac{\lambda^i e^{-\lambda}}{i!} \\
&= 1 - \Bigg(1 - \sum_{i=0}^{\lfloor n\alpha - 1 \rfloor} \frac{\lambda^i e^{-\lambda}}{i!} P_{ruin}(M_i) \\
&\qquad - \Big(e^{-\lambda} \sum_{i=0}^{\infty} \frac{\lambda^i}{i!} - \sum_{i=0}^{\lfloor n\alpha - 1 \rfloor} \frac{\lambda^i e^{-\lambda}}{i!} \Big) \Bigg) \\
&= 1 - \sum_{i=0}^{\lfloor n\alpha - 1 \rfloor} \frac{\lambda^i e^{-\lambda}}{i!} \big(1 - P_{ruin}(M_i)\big).
\end{aligned}$$

Let $\varepsilon \in (0, 1)$. We set $K_{\mathcal{H}} = \max_\alpha \inf_{n \geq 0} \{\mathbb{P}\{C_n^\alpha\} < \varepsilon\}$ and we denote by E the event that $\|C \setminus (C \cap C')\| > \|C' \setminus (C \cap C')\|$ given that $K_{\mathcal{H}}$ blocks have been appended to $C \setminus (C \cap C')$. Let $\overline{E}$ be the complementary event of E. Since $\overline{E} \subseteq C_{K_{\mathcal{H}}}^\alpha$, we have $\mathbb{P}\{\overline{E}\} \leq \mathbb{P}\{C_{K_{\mathcal{H}}}^\alpha\} < \varepsilon$. Thus, we have $\mathbb{P}\{E\} > 1 - \varepsilon$. The argument for parameter $K_{\mathcal{A}}$ is similar except that during the first phase of the two-phase competition, both the honest team and the adversary create their own blocks until the adversary has created n_a blocks. We determine the probability of $C_{n_a}^\alpha$, i.e., the probability that the adversarial subchain $C' \setminus (C \cap C')$ catches up with the honest subchain $C \setminus (C \cap C')$ after n_a blocks have been appended by the adversarial team to block $b^\star$. Let $M_i = ih - n_a a$. We introduce the random variable $X_h \sim \text{Poisson}(\lambda_h)$, which counts the number of blocks the honest team produces in the time it takes for the adversary to mine n_a blocks. Quantity $\lambda_h = pn_a/\alpha q$ is the expected number of honest blocks generated over that same interval.

$$\mathbb{P}\{C_{n_a}^\alpha\} = \sum_{i=0}^{\left\lfloor \frac{1+n_a}{\alpha} \right\rfloor} \frac{\lambda_h^i e^{-\lambda_h}}{i!} + \sum_{i=\left\lfloor \frac{1+n_a}{\alpha} \right\rfloor + 1}^{\infty} \frac{\lambda_h^i e^{-\lambda_h}}{i!} P_{ruin}(M_i).$$

We set $K_{\mathcal{A}} = \max_\alpha \inf_{n_a \geq 0} \{\mathbb{P}\{C_{n_a}^\alpha\} < \varepsilon\}$, and finally set K as $K = max(K_{\mathcal{A}}, K_{\mathcal{H}})$.

For conservative reasons, we take $K_{\mathcal{H}} = 208$ and $K_{\mathcal{A}} = 32$, which are obtained when $\varepsilon = 10^{-6}$, which means that when $K = 208$ blocks have been mined on either $C \setminus (C \cap C')$ or $C' \setminus (C \cap C')$, then with probability $1 - \varepsilon$, $\|C \setminus (C \cap C')\| > \|C' \setminus (C \cap C')\|$. More details on the impact of α are discussed in the full version of this paper [25].

Figure 6 compares, for each level $\ell \geq 0$, the accumulated difficulty of the last K blocks of the Bitcoin blockchain up to October 11, 2024, with the theoretical bound $D_{\min}^{\mathcal{A}}(\ell)$, when K is set to 208. It clearly shows that K blocks are sufficient to accumulate $D_{\min}^{\mathcal{A}}(\ell)$ quantity of difficulty for each level $\ell \geq 0$.

Based on the previous results, we can now show that our solution is secure in the presence of a 1/3-adversary.

Theorem 6.9 (Security). *Given a typical execution with a 1/3-bounded PPT adversary. Let r be a round of the execution. Let $\Pi =$*

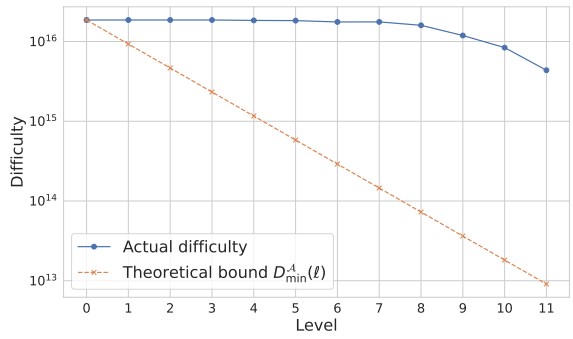


Figure 6: Comparison between the theoretical accumulated quantity of difficulty $D^{\mathcal{A}}_{\min}(\ell)$ and the one accumulated in the last K blocks of level $\ell \geq 0$ of the Bitcoin blockchain on October 11, 2024. Setting: $K = 208$, $\chi = 4032$, and $k = 323$.

$\texttt{Compress}_{K,\chi,k}(\mathcal{C})$ be the NIPoPoW generated from blockchain $\mathcal{C}$ by an honest prover at round r. Let Π' be a proof generated by the adversarial prover at round r. Let $\Pi^\star = \texttt{Compare}_{K,\chi,k}(\Pi, \Pi')$ be the proof accepted by the verifier algorithm. Then, with overwhelming probability, the verifier accepts the honest NIPoPoW, i.e., $\|\Pi^\star\{(\Pi^\star \cap \mathcal{C})[-1] :\}\| \geq \|\mathcal{C}\{(\Pi^\star \cap \mathcal{C})[-1] :\}\|$.

Proof. Let M be the set of levels where both chains share at least one block, i.e., $M = \{\mu \in \mathbb{N} \mid (\Pi \cap \Pi') \uparrow^\mu \neq \emptyset\}$. If M is not empty, we set $\mu = \min(M)$; otherwise, we set $\mu = \perp$. Three cases must be considered.

$\mu = 0$: By definition, $(\Pi^\star \cap \mathcal{C})[-1]$ represents the last block b shared by both $\Pi^\star$ and $\mathcal{C}$, and $\Pi^\star\{(\Pi^\star \cap \mathcal{C})[-1] :\}$ represents the subchain of $\Pi^\star$ starting from block b to the last block of $\Pi^\star$. By assumption of the case, $\mu = 0$, block b therefore belongs to the last $2K + \chi + k$ blocks of both $\mathcal{C}$ and $\Pi^\star$. At round r, the honest prover maintains $\mathcal{C}$. Thus, for any other chain $\mathcal{C}'$, we have $\|\mathcal{C}\| \geq \|\mathcal{C}'\|$. Hence, the only possibility for a verifier to accept $\Pi^\star = \Pi'$ is that the adversarial prover correctly executes $\texttt{Compress}\ (\Pi'\ b')$ with b' a valid block mined with legitimate difficulty. Indeed, by assumption of the case, the legitimacy of b''s difficulty can be checked with the last $2K + \chi + k$ blocks. This completes the case proof.

$\mu > 0$: Let $\mathcal{C}'$ be any chain different from the honest chain $\mathcal{C}$. By Lemma 6.3, there exists a quantity $D^{\mathcal{A}}_{\min}(\mu)$ such that if $\|\mathcal{C}' \setminus (\mathcal{C} \cap \mathcal{C}') \uparrow^\mu\| \geq D^{\mathcal{A}}_{\min}(\mu)$, then, with overwhelming probability, $\|\mathcal{C} \setminus (\mathcal{C} \cap \mathcal{C}') \uparrow^\mu\| > \|\mathcal{C}' \setminus (\mathcal{C} \cap \mathcal{C}') \uparrow^\mu\|$. The combination of Corollary 6.4 and the Bitcoin trace (Figure 6) shows that the K blocks at level μ are sufficient to accumulate the quantity of difficulty $D^{\mathcal{A}}_{\min}(\mu)$, this quantity of difficulty being exponentially smaller than the one required at level 0. By Algorithm 2, if $\Pi = \texttt{Compress}(\mathcal{C})$, then $\Pi\{b :\} \uparrow^\mu = \mathcal{C}\{b :\} \uparrow^\mu$. Furthermore, by Lemma 5.1, either Π or Π' have at least K μ-blocks after the LCA block b. Function `highdiff()` in Algorithm 2 returns the subchain restricted to the last K μ-blocks that has accumulated the largest quantity of difficulty. Thus, even if the adversarial blockchain contains as many μ-blocks as the honest one, their accumulated mining difficulty cannot exceed the honest one, preventing accordingly low-difficulty attacks. By Lemma 6.3, Π is accepted, which completes the case of the proof.

$\mu = \perp$: In this case, the proofs Π and Π' do not share an LCA block. By Algorithm 2, the verifier searches for the annotated blocks b and b' in Π and Π', respectively. By construction, blocks b and b' are recent blocks (they belong to subchains $X\ \Omega$ of Π and $X'\ \Omega'$ of Π', respectively) and have been mined at approximately the same time. They can therefore be used as a temporal anchor from which the accumulated difficulty of subsequent blocks can be checked exactly as in the Bitcoin protocol (i.e., by checking the legitimacy of each block after both b and b'). If both b and b' are less than K deep from the tips of Π and Π', respectively, then the verifier observes the system and waits until b or b' are followed by K blocks (i.e., $|(\mathcal{D}\ X\ \Omega)\{b :\}| = K \vee |(\mathcal{D}'\ X'\ \Omega')\{b' :\}| = K$). Once this condition is verified, the verifier accepts the NIPoPoW returned by function `highdiff` fed with the chains of blocks that respectively follow b and b'. By construction, `highdiff` returns the K long subchain that has accumulated the largest quantity of difficulty. As for the case $\mu = 0$, the legitimacy of all these blocks can be checked. Hence, if the adversarial prover presents an illegitimate proof (in particular one that results from a brute-force low-difficulty attack), then the verifier will reject it. This completes the proof of this last case. □

Our code, data, and figures are all publicly available for the implementation of our scheme[6] and for the calculation of K[7]. The implementation of our scheme uses Bitcoin's historical data to generate the proofs.

7 Conclusion

We have presented a non-interactive, succinct, and secure representation of a PoW-based blockchain that operates in a variable difficulty setting. This proof, called NIPoPoW, satisfies both the completeness and onlineness properties. Communication, storage, and latency costs are polylogarithmic in the size of the full blockchain. From an operational point of view, building a NIPoPoW requires augmenting each block header with a logarithmic number of pointers. As proposed by Kiayias *et al.* [22, 30] this can be deployed via velvet forks.

As future work, we intend to extend the solution proposed in [15]. We are pretty confident that handling block difficulties and weight should improve the resiliency of our solution to a 1/2-adversary. The other challenging objective is to find succinct and secure representations of other types of permissionless blockchains (e.g., Proof-of-Stake [8, 14], Proof-of-Interaction [1]). Indeed, our scheme deeply relies on the stochastic and cryptographic properties of the hash function of the PoW. The question is whether these alternative permissionless blockchains present similar stochastic properties.

Acknowledgments

We would like to thank the reviewers for their thoughtful comments and efforts towards improving our manuscript.

Nathanaël Derousseaux-Lebert, Loïc Miller and Dorian Pacaud have been funded by the ANR (french National Research Agency) project BC4SSI ANR-22-CE25-0001.

[6] https://github.com/loicmiller/variable-mls
[7] https://github.com/nderousseaux/imt-internship

References

[1] Jean-Philippe Abegg, Quentin Bramas, and Thomas Noël. 2021. Blockchain Using Proof-of-Interaction. In *Proceedings of the International Conference on Networked Systems (NETYS)*, Karima Echihabi and Roland Meyer (Eds.). Springer International Publishing, Cham, 129–143.

[2] Adam Back. 2002. Hashcash - A Denial of Service Counter-Measure. (2002).

[3] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew K. Miller, Andrew Poelstra, and Jorge Timón. 2014. Enabling Blockchain Innovations with Pegged. https://api.semanticscholar.org/CorpusID:18659636

[4] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS)*. ACM, 62–73.

[5] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. 2020. Coda: Decentralized Cryptocurrency at Scale. *IACR Cryptol. ePrint Arch.* (2020), 352. https://eprint.iacr.org/2020/352

[6] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. 2020. Mina: Decentralized Cryptocurrency at Scale. https://minaprotocol.com/wp-content/uploads/technicalWhitepaper.pdf

[7] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. 2020. FlyClient: Super-Light Clients for Cryptocurrencies. In *Proceedings of the Symposium on Security and Privacy (S&P)*. IEEE, 928–946.

[8] Vitalik Buterin and Virgil Griffith. 2017. Casper the Friendly Finality Gadget. *ArXiv* abs/1710.09437 (2017). https://api.semanticscholar.org/CorpusID:11301538

[9] Alexander Chepurnoy, Charalampos Papamanthou, and Yupeng Zhang. 2018. Edrax: A Cryptocurrency with Stateless Transaction Validation. *IACR Cryptol. ePrint Arch.* (2018), 968. https://eprint.iacr.org/2018/968

[10] Bitcoin Protocol documentation. 2021. https://en.bitcoin.it/wiki/Protocol_documentation

[11] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2017. The Bitcoin Backbone Protocol with Chains of Variable Difficulty. In *Proceedings of the Annual International Cryptology Conference (CRYPTO)*. Springer, 291–323.

[12] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9057. Springer, 281–310.

[13] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. 2019. Proof-of-Stake Sidechains. In *Proceedings of the Symposium on Security and Privacy (S&P)*. IEEE, 139–156.

[14] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on the ACM Operating Systems Principles (SOSP) (SOSP)*. 51–68.

[15] Anurag Jain, Emmanuelle Anceaume, and Sujit Gujar. 2023. Extending The Boundaries and Exploring The Limits Of Blockchain Compression. In *Proceedings of the 43th Symposium on Reliable and Distributed Systems (SRDS)*. IEEE, 187–197.

[16] Thaddeus Dryja Joseph Poon. 2016. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf

[17] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. 2019. Compact Storage of Superblocks for NIPoPoW Applications. In *Proceedings of the 1st International Conference on Mathematical Research for Blockchain Economy (MARBLE)*. Springer, 77–91.

[18] Guy Katriel. 2013. Gambler's ruin probability—A general formula. *Statistics & Probability Letters* 83, 10 (2013), 2205–2210.

[19] Aggelos Kiayias, Nikos Leonardos, and Dionysis Zindros. 2021. Mining in Logarithmic Space. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[20] Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. 2020. A Composable Security Treatment of the Lightning Network. In *Proceedings of the 33rd Computer Security Foundations Symposium (CSF)*. IEEE, 334–349.

[21] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. 2020. Non-interactive Proofs of Proof-of-Work. In *Proceedings of the Financial Cryptography and Data Security (FC)*. Springer.

[22] Aggelos Kiayias, Andrianna Polydouri, and Dionysis Zindros. 2021. The velvet path to superlight blockchain clients. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies (AFT)*. Association for Computing Machinery, 205–218.

[23] Benjamin Loison. 2023. Mining in Logarithmic Space with Variable Difficulty. In *Proceedings of the 5th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*.

[24] Roman Matzutt, Benedikt Kalde, Jan Pennekamp, Arthur Drichel, Martin Henze, and Klaus Wehrle. 2021. CoinPrune: Shrinking Bitcoin's Blockchain Retrospectively. *Transactions on Network and Service Management* 18, 3 (2021), 3064–3078.

[25] Loïc Miller, Dorian Pacaud, Nathanael Derousseaux, Emmanuelle Anceaume, and Romaric Ludinard. 2025. Technical Report: Mining in Logarithmic Space with Variable Difficulty. (2025). https://cnrs.hal.science/view/index/docid/5138795 working paper or preprint.

[26] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf

[27] Jonas Nick, Andrew Poelstra, and Gregory Sanders. 2020. Liquid: A Bitcoin Sidechain. https://blockstream.com/assets/downloads/pdf/liquid-whitepaper.pdf

[28] A. Pinar Ozisik and Brian Neil Levine. 2017. An Explanation of Nakamoto's Analysis of Double-spend Attacks. *CoRR* abs/1701.03977 (2017). http://arxiv.org/abs/1701.03977

[29] SNAP. 2020. Ethereum Snapshot Protocol. https://github.com/ethereum/devp2p/blob/master/caps/snap.md

[30] Alexei Zamyatin, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Edgar R. Weippl, and William J. Knottenbelt. 2018. A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice - (Short Paper). In *Proceedings of the Financial Cryptography and Data Security Conference (FC)*. Springer.