Non-Interactive Proofs of Proof-of-Work

Aggelos Kiayias[1], Andrew Miller[2], and Dionysis Zindros[3,⋆]

[1] University of Edinburgh, IOHK
[2] University of Illinois at Urbana-Champaign, Initiative for Cryptocurrencies and Contracts
[3] National and Kapodistrian University of Athens

December 4, 2017

Abstract. Blockchain protocols such as Bitcoin provide decentralized consensus mechanisms based on proof-of-work (PoW). In this work we introduce and instantiate a new primitive for blockchain protocols called Non-Interactive-Proofs-of-Proof-of-Work (NIPoPoWs) which can be adapted into existing PoW-based cryptocurrencies. Unlike a traditional blockchain client which must verify the entire linearly-growing chain of PoWs, clients based on NIPoPoWs can verify a certain blockchain property requiring resources only logarithmic in the length of the blockchain. NIPoPoWs solve two important open questions for PoW based consensus protocols: The problem of constructing efficient transaction verification (SPV) clients and the problem of constructing efficient sidechain proofs. We provide a formal model for NIPoPoWs and two constructions for blockchain properties that we prove secure and are of interest with respect to the above applications. We also provide simulations and experimental data to measure the concrete communication efficiency and security of our construction. We also present an attack against the only previously known (interactive) PoPoW protocol that showcases the difficulty of designing such protocols. Finally, we provide two ways that our NIPoPoWs can be adopted by existing blockchain protocols, first via a soft fork, and second via a new update mechanism that we term a "velvet fork" that enables harnessing some of the performance benefits of NIPoPoWs even with a minority upgrade.

1 Introduction

Cryptocurrencies such as Bitcoin [1] [2] and Ethereum [3] are peer-to-peer networks that maintain a globally consistent transaction ledger, using a consensus protocol based on proof-of-work (PoW) puzzles [4, 5]. Worker nodes called "miners" expend computational work in order to reach agreement on the state of the network. Clients on the network, such as mobile phone apps, must verify these PoWs in order to determine the correct view of the network's state, something necessary to transmit and receive payments correctly.

In this work we introduce, analyze and instantiate a new primitive, Non-Interactive Proofs of Proof-of-Work (NIPoPoWs), which can be adapted into existing cryptocurrencies to support more efficient clients. A traditional blockchain client in order to check a certain blockchain property must verify the entire chain of proofs-of-work, which grows linearly over time. On the contrary, clients based on NIPoPoWs require resources only logarithmic in the length of the blockchain.

1.1 Motivation

There are three trends that motivate the need for more efficient clients, specifically clients that can verify transactions without having to process the whole blockchain. First, while Bitcoin remains by far the largest cryptocurrency, the ecosystem has become significantly more diverse. There are hundreds of competitors, called "altcoins", and Bitcoin's dominance in the market is at an all time low [6].[4] We envision our protocol will form the basis

⋆ Research supported by ERC project CODAMODA, project #259152

[4] Using market cap as a measure of economic size is debatable but still indicates the direction of the evolving landscape.

of an efficient multi-blockchain client, which could efficiently support payments using hundreds of separate cryptocurrencies. This requires optimizing the "SPV client" described in the original Bitcoin paper [1] which, while quite efficient, requires processing an amount of data growing linearly with the size of the blockchain.

Second, cryptocurrencies and blockchain protocols in general are increasingly used as components of larger systems. As one example, a recent system called Catena [7] uses the blockchain as an authenticated log; the browser embeds a Bitcoin client to validate HTTPS certificates. The bandwidth costs that are tolerable for a dedicated Bitcoin wallet are likely unacceptable in such embedded contexts. Our techniques can directly reduce the cost of Catena and similar systems.

Finally, there has been significant interest in the development of "cross-chain" applications, i.e. logical transactions that span multiple separate blockchains. Simple cross-chain transactions are feasible today: the most well-known is the atomic exchange [8], e.g., a trade of bitcoin for ether. However, more sophisticated applications could be enabled by our protocol, which would allow the blockchain of one cryptocurrency to embed a client of a separate cryptocurrency. This concept, initially popularized by a proposal by Back et al. [9], can be used to avoid a difficult upgrade process: a new blockchain with additional features, such as experimental opcodes, can be backed by deposits in the original bitcoin currency, obviating the need to bootstrap new currencies from scratch. The ability to use sidechains as an upgrade and experimentation mechanism has been heralded as a key enabler of scalability in the Bitcoin ecosystem; however, the sidechains proposal is incomplete, as it does not provide a protocol that satisfies a natural security definition, i.e., where attacks succeed with only negligible probability, even though several attempts were made [10, 11]. Our work is the first to answer the open problem posed by Back et al. Sidechains can also be used as a mechanism to off-load some of the workload incurred by the Bitcoin network by moving coins to dedicated sidechains, which can later be moved back.

These examples illustrate that our solution is a key component for two important pillars needed for next-generation blockchains: *interoperability* and *scalability*. While we use bitcoin concretely as an example, any proof-of-work cryptocurrency can adopt our techniques.

1.2 Our contributions

In summary, we make the following contributions. Our main technical contribution is the introduction and instantiation of a new cryptographic primitive called Non-Interactive Proofs of Proof-of-Work (NIPoPoW). We present a formal model and a provably secure instantiation of NIPoPoWs in the backbone model [12]. Our contribution builds on previous work of [13] who posed *interactive* proofs of proof-of-work, which, in turn, are based on previous discussion of such concepts in the bitcoin forums [14]. We in fact show an explicit attack against the construction of [13] that showcases the difficulty of designing such protocols. It follows that our construction is the first secure Proof of Proof-of-Work. Furthermore, our solution has the additional property of being non-interactive, as the previous construction could revert into interaction by an adversarial prover. Similar to previous work, we prove that the proofs are optimistically succinct meaning that, in honest conditions, the proofs are logarithmic in size. Improving previous work, we show that optimistic succinctness can be achieved for adversarially-generated proofs for a number of blockchain predicates that are high value use cases. Our definition fills the gap in terms of security modelling and design that existed in previous proposals, e.g., the notion of cumulative "Dynamic Member Multisignature" [9].

We provide concrete parameterization and empirical analysis focusing on showing the potential savings of our approach versus existing clients. Using real data from the Bitcoin network and other blockchains, we quantify the actual savings of NIPoPoWs over the previous known techniques of constructing efficient SPV verifiers.

We describe two practical deployment paths for our NIPoPoWs that existing cryptocurrencies can adopt: First using either a "soft fork" or a "hard fork" upgrade procedure, both of which have been successfully used by existing cryptocurrencies [15]. Second, using a less disruptive update mechanism that we term a "velvet fork" and which may be of independent interest. In a velvet fork update, the clients that have "forked" from the original implementation continue to be fully compatible with un-updated clients. It follows that, in a velvet fork, the blockchain system can remain supported by a diverse software codebase indefinitely, while it can still enjoy, at least in proportion, some of the (efficiency in our case) benefits of the update without any of the security downsides.

2 Model and Definitions

We define our problem in the setting of a blockchain client which is completely stateless beyond the knowledge of a common reference string, the genesis block. Without loss of generality, this single reference block could be any stable checkpoint block, either hard-coded in the client or obtained through a previous interaction with the network. However, importantly, the client does not maintain a blockchain. The challenge is, then, to build a client that is able to communicate with the network and, without downloading the whole chain headers, is convinced in a secure manner about a certain predicate on the "main chain", e.g., whether a specific transaction has been confirmed. We term such a client a *verifier* and we term *provers* the full nodes it connects to. The provers maintain a complete blockchain and can be thought of as full nodes. The verifier connects to multiple provers, some of which may be malicious, but at least one of which is honest. All of the provers provide a proof and the verifier has to decide based on the feedback received what is the correct value of the predicate in question.

2.1 The prover and verifier model

The prover-verifier interaction is as follows: for a given predicate on the chain (e.g. "the transaction took place"), the prover computes the proof string and transmits it to the verifier. The verifier accepts or rejects the string. Note that in the interactive case the prover and verifier may engage in more than one round of message passing.

The entities on the blockchain network are of 3 kinds: (1) miners, who try to mine new blocks on top of the longest known blockchain and broadcast them as soon as they are discovered (for simplicity we assume that difficulty is constant and thus the "longest chain rule" sufficiently describes honest miner behavior); (2) full nodes, who maintain the longest blockchain without mining and also act as the provers in the network; (3) verifiers or stateless clients, who connect to provers and ask for proofs in regards to which blockchain is the largest. The verifiers attempt to determine the value of a predicate on these chains.

We model proof-of-work discovery attempts by using a random oracle [16] as in [12]. The random oracle produces κ-bit strings, where κ is the system's security parameter. The network is synchronized into numbered rounds, which correspond to moments in time. n denotes the total number of miners in the game, while t denotes the total number of adversarial miners. Each miner is assumed to have equal mining power captured by the number of queries q available per player to the random oracle, each query of which

succeeds independently with probability p (a successful query produces a block with valid proof-of-work). Mining pools and miners of different computing power can be captured by assuming multiple players combine their computing power. This is made explicit for the adversary, as they do not incur any network overhead to achieve communication between adversarial miners. On the contrary, honest players discovering a block must *diffuse* it (broadcast it) to the network at a given round and wait for it to be received by the rest of the honest players at the beginning of the next round. A round during which an honest block is diffused is called a *successful round*; if the number of honest blocks diffused is one, it is called *uniquely successful round*. We assume there is an honest majority, i.e., that $t/n < 0.5$ with a significant gap [12]. We further assume that the network is adversarial, but that there is no eclipsing attacks [17]. More specifically, we allow the adversary to reorder messages transmitted at a particular round, to inject new messages thereby capturing Sybil attacks [18], but not to drop messages. Each honest miner maintains a local chain $\mathcal{C}$ which they consider the current active blockchain. Upon receiving a different blockchain from the network, the current active blockchain is changed if the received blockchain is longer than the currently adopted one. Receiving a different blockchain of the same length as the currently adopted one does not change the adopted blockchain.

Blockchain blocks are generated by including the following data in them: *ctr*, the nonce used to achieve the proof-of-work; x the Merkle tree [19] root of the transactions confirmed in this block; and *interlink*, a vector containing pointers to previous blocks, including the id of the previous block. The *interlink* data structure contains pointers to more blocks than just the previous block. We will explain this further in Section 3. Given two hash functions H and G modelled as random oracles, the id of a block is defined as $\mathsf{id} = H(ctr, G(x, \mathsf{interlink}))$. In bitcoin's case, both H and G would be SHA256.

The predicates of interest in our context are predicates on the active blockchain. Some of the predicates are more suitable for succinct proofs than others. We focus our attention in *stable* predicates having the property that all honest miners share their view of them in a way that is updated in a predictable manner, with a truth-value that persists as the blockchain grows (an example of an unstable predicate is e.g., the least significant bit of the hash of last block). Following the work of [12], we wait for k blocks to bury a block before we consider it *confirmed* and thereby the predicates depending on it stable. k is the *common prefix* security parameter, which in bitcoin folklore is often taken to be $k = 6$.

In our setting, for a given predicate Q, several provers (including adversarial ones) will generate proofs claiming potentially different truth values for Q based on their claimed local longest chains. The verifier receives these proofs and accepts one of the proofs, determining the truth value of the predicate. We denote a *blockchain proof protocol* for a predicate Q as a pair (P, V) where P is the *prover* and V is the *verifier*. P is a PPT algorithm that is spawned by a full node when they wish to produce a proof, accepts as input a full chain $\mathcal{C}$ and produces a proof π as its output. V is a PPT algorithm which is spawned at some round, receives a pair of proofs (π_A, π_B) from both an honest party and the adversary and returns its decision $d \in \{T, F, \perp\}$ before the next round and terminates. The honest miners produce proofs for V using P, while the adversary produces proofs following some arbitrary strategy. Before we introduce the security properties for blockchain proof protocols we introduce some necessary notation for blockchains.

2.2 Blockchain addressing

Blockchains are finite block sequences obeying the *blockchain property* that in every block in the chain there exists a pointer to its previous block. A chain is *anchored* if its first block is *genesis*, denoted *Gen*.

For chain addressing we use Python brackets $\mathcal{C}[\cdot]$ as in [20]. A zero-based positive number in a bracket indicates the indexed block in the chain. A negative index indicates a block from the end, e.g., $\mathcal{C}[-1]$ is the tip of the blockchain. A range $\mathcal{C}[i:j]$ is a subarray starting from i (inclusive) to j (exclusive).

Given chains $\mathcal{C}_1, \mathcal{C}_2$ and blocks A, Z we concatenate them as $\mathcal{C}_1\mathcal{C}_2$ or $\mathcal{C}_1 A$. $\mathcal{C}_2[0]$ must point to $\mathcal{C}_1[-1]$ and A must point to $\mathcal{C}_1[-1]$. We denote $\mathcal{C}\{A:Z\}$ the subarray of the chain from A (inclusive) to Z (exclusive). We can omit blocks or indices from either side of the range to take the chain to the beginning or end respectively.

Helper functions for blockchains. The *id* function returns the id of a block given its data, i.e., $\mathsf{id} = H(ctr, G(x, \mathsf{interlink}))$. *depth* is a function which, given a block, returns its distance from the genesis block.

Valid blocks satisfy the proof-of-work condition: $id \leq T$, where T is the mining target. Throughout this work, we make the simplifying assumption that T is constant. Some blocks will achieve a lower id. If $id \leq \frac{T}{2^\mu}$ we say that the block is of level μ. All blocks are level 0. Blocks with level μ are called *μ-superblocks*. μ-superblocks for $\mu > 0$ are also $(\mu-1)$-superblocks. By convention, for *Gen* we set $id = 0$ and $\mu = \infty$.

In this paper, we will extend blocks to contain multiple pointers to previous blocks. Certain blocks can be omitted from a chain, obtaining a subchain, as long as the blockchain property that each block must contain a pointer to its previous block in the sequence is maintained.

Blockchains are sequences, but it is more convenient to use set notation for some operations. Specifically, $B \in \mathcal{C}$; $\mathcal{C}_1 \subseteq \mathcal{C}_2$ and $\emptyset$ have the obvious meaning. $\mathcal{C}_1 \cup \mathcal{C}_2$ is the chain obtained by sorting the blocks contained in both $\mathcal{C}_1$ and $\mathcal{C}_2$ into a sequence (this may be not always defined). We will freely use set builder notation $\{B \in \mathcal{C} : p(B)\}$. $\mathcal{C}_1 \cap \mathcal{C}_2$ is the chain $\{B : B \in \mathcal{C}_1 \wedge B \in \mathcal{C}_2\}$. In all cases, the blockchain property must be maintained. The lowest common ancestor is $\mathsf{LCA}(\mathcal{C}_1, \mathcal{C}_2) = (\mathcal{C}_1 \cap \mathcal{C}_2)[-1]$. If $\mathcal{C}_1[0] = \mathcal{C}_2[0]$ and $\mathcal{C}_1[-1] = \mathcal{C}_2[-1]$, we say the chains $\mathcal{C}_1, \mathcal{C}_2$ *span* the same block range.

It will soon become clear that it is useful to construct a chain containing only the superblocks of another chain. Given $\mathcal{C}$ and level μ, the *upchain* $\mathcal{C}\uparrow^\mu$ is defined as $\{B \in \mathcal{C} : level(B) \geq \mu\}$. A chain containing only μ-superblocks is called a *μ-superchain*. It is also useful, given a μ-superchain $\mathcal{C}'$ to go back to the regular chain $\mathcal{C}$. Given chains $\mathcal{C}' \subseteq \mathcal{C}$, the *downchain* $\mathcal{C}'\downarrow_{\mathcal{C}}$ is defined as $\mathcal{C}[\mathcal{C}'[0] : \mathcal{C}'[-1]]$. $\mathcal{C}$ is the *underlying chain* of $\mathcal{C}'$. The underlying chain is often implied by context, so we will simply write $\mathcal{C}'\downarrow$. By the above definition, the $\mathcal{C}\uparrow$ operator is absolute: $(\mathcal{C}\uparrow^\mu)^{\mu+i} = \mathcal{C}\uparrow^{\mu+i}$. Given a set of consecutive rounds $S = \{r, r+1, \cdots, r+j\} \subseteq \mathbb{N}$, we define $\mathcal{C}^S = \{B \in \mathcal{C} : B \text{ was generated during } S\}$.

2.3 Desired properties

We now define two desired properties of a non-interactive blockchain proof protocol, *succinctness* and *security*. For clarity, the properties are described in the execution model of [12] and can be easily ported to a more refined model e.g., [21].

Definition 1. *(Security) A blockchain proof protocol is secure if for all environments and for all PPT adversaries $\mathcal{A}$ the output of V on round r is the same as the evaluation of $Q(\mathcal{C})$ on some honest party's chain $\mathcal{C}$.*

Definition 2. *(Succinctness) A blockchain proof protocol is succinct if the maximum proof size $|\pi|$ across all provers for a given round r is $O(polylog(r))$.*

Based on these definitions, it is trivial to construct a secure but not succinct protocol: The prover provides the chain $\mathcal{C}$ itself as a proof and the verifier simply compares the proofs

received by length: Since the longest proof is literally the longest chain, the protocol is trivially secure but not succinct. This is what traditional SPV clients do. It is also easy to build succinct insecure clients: The prover simply sends the value of the predicate directly. This is what web-based wallets do. The challenge we will solve is to provide a non-interactive protocol which at the same time achieves security and (optimistic) succinctness over a large class of predicates.

2.4 Provable chain predicates

Not all predicates are suitable for our purposes. In this section we characterize the class of blockchain predicates that will be of interest to us for constructing proofs. Given that we are in a decentralized setting it can be the case that for certain sensible blockchain predicate, there will be honest nodes that have not reached a conclusion about its value. For this reason we allow our predicates to have an undefined value $\perp$ before they take on a truth value of T or F. We now define some useful predicate properties.

Definition 3. *(Monotonicity) A chain predicate $Q(\mathcal{C})$ is monotonic if for all chains $\mathcal{C}$ and for all blocks B we have that: $Q(\mathcal{C}) \neq \perp \Rightarrow Q(\mathcal{C}) = Q(\mathcal{C}B)$.*

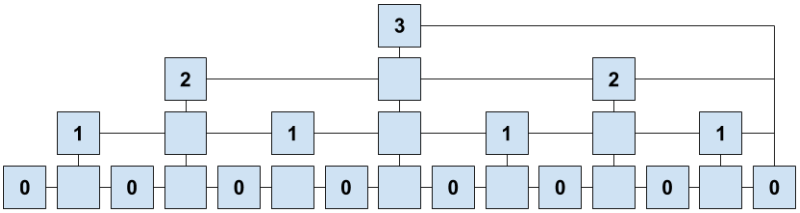
Definition 4. *(Stability) Parameterized by $k \in \mathbb{N}$, a chain predicate Q is k-stable if its value only depends on $\mathcal{C}[:-k]$.*

3 Consensus layer support

3.1 The interlink pointers data structure

In order to construct our protocols, as in [13], we take advantage of a data structure that will be stored in the header of a block. Observe that in a blockchain protocol execution it is expected half of the blocks will be of level 1, 1/4 of the blocks will be of level 2, 1/8 will be of level 3 and $1/2^{\mu}$ blocks will be of level μ. In expectation, the number of superblock levels of a chain $\mathcal{C}$ will be $\Theta(\log(\mathcal{C}))$ [13]. Figure 1 illustrates the blockchain superblocks starting from level 1 and going up to level 4 in case these blocks are distributed exactly according to expectation. Here, each level contains half the blocks of the level below.

Fig. 1. The hierarchical blockchain. Higher levels have achieved a lower target (higher difficulty) during mining.



The *interlink* data structure is proposed to be included in each block, replacing the existing pointer to the previous block with a list of pointers to a small number of previous blocks. For each block, this data structure contains a pointer to the most recent preceding block of every level μ. The algorithm for this construction is shown in Algorithm 1 and is borrowed from [13]. Genesis is of infinite level and hence a pointer to it is included in every block at the first available index within the interlink data structure. The interlink

data structure turns the blockchain into a skiplist-like [22] data structure. The number of pointers that need to be included per block is in expectation $\log(|\mathcal{C}|)$.

The updateInterlink algorithm accepts a block B', which already has an interlink data structure defined on it. The function evaluates the interlink data structure which needs to be included as part of the next block. It copies the existing interlink data structure and then modifies its entries from level 0 to $\mathsf{level}(B')$ to point to the block B'.

Algorithm 1 updateInterlink

```
1: function updateInterlink(B')
2:     interlink ← B'.interlink
3:     for μ = 0 to level(B') do
4:         interlink[μ] ← blockid(B')
5:     end for
6:     return interlink
7: end function
```

3.2 Superchain quality

In order to argue formally about the security properties of blockchains that are equipped with the interlink data structure we will introduce a new concept of chain quality, inspired by the definition of this property in [12], called *superchain quality.* We first define a notion of "goodness" that bounds the deviation of superblocks of a certain level from their expected mean. Using this we then define superchain quality.

Definition 5 (Locally good superchain). *A superchain $\mathcal{C}'$ of level μ with underlying chain $\mathcal{C}$ is said to be* μ-locally-good *with respect to security parameter δ, written $\mathsf{local\text{-}good}_\delta(\mathcal{C}', \mathcal{C}, \mu)$, if $|\mathcal{C}'| > (1-\delta)2^{-\mu}|\mathcal{C}|$.*

Definition 6 (Superchain quality). *The (δ, m) superquality property Q^{μ}_{scq} of a chain $\mathcal{C}$ pertaining to level μ with security parameters $\delta \in \mathbb{R}$ and $m \in \mathbb{N}$ states that for all $m' \geq m$, it holds that $\mathsf{local\text{-}good}_\delta(C{\uparrow^\mu}[-m':], C{\uparrow^\mu}[-m':]{\downarrow}, \mu)$. That is, all sufficiently large suffixes are locally good.*

While superchain quality guarantees that a superchain has a sufficient number of blocks w.r.t. the level 0 chain, we will also need to be able to compare it with other superchains. For this reason we introduce multilevel quality.

Definition 7 (Multilevel quality). *A μ-superchain $\mathcal{C}'$ is said to have multilevel quality, written* multi-good$_{\delta,k_1}(\mathcal{C}, \mathcal{C}', \mu)$ *with respect to an underlying chain $\mathcal{C} = \mathcal{C}'{\downarrow}$ with security parameters k_1, δ if for all $\mu' < \mu$ it holds that for any $\mathcal{C}^* \subseteq \mathcal{C}$, if $|\mathcal{C}^*{\uparrow^{\mu'}}| \geq k_1$, then $|\mathcal{C}^*{\uparrow^{\mu}}| \geq (1-\delta)2^{\mu-\mu'}|\mathcal{C}^*{\uparrow^{\mu'}}|$.*

Putting the above together we conclude with the notion of a *good* superchain.

Definition 8 (Good superchain). *A μ-superchain $\mathcal{C}'$ is said to be good, written* good$_{\delta,k_1}(\mathcal{C}, \mathcal{C}', \mu)$, *with respect to an underlying chain $\mathcal{C} = \mathcal{C}'{\downarrow}$ if it has both superquality and multilevel quality with parameters (δ, m).*

It is not hard to see that the above good statistical properties are attained with overwhelming probability by all chains that are generated in optimistic environments, i.e. if no adversary tries to violate them. This is established in the following lemmas.

Lemma 1 (Local goodness). *Assume $\mathcal{C}$ contains only honestly-generated blocks in an optimistic execution. For all levels μ, for all constant $\delta > 0$, all continuous subchains $\mathcal{C}' = \mathcal{C}[i:j]$ with $|\mathcal{C}'| \geq m$ are locally good, $\mathsf{local\text{-}good}_\delta(\mathcal{C}', \mathcal{C}, \mu)$, with overwhelming probability in m.*

Proof. Observing that for each honestly generated block the probability of being a μ-superblock for any level μ follows an independent Bernoulli distribution, we can apply a Chernoff bound to show that the number of superblocks within a chain will be close to its expectation, which is what is required for local goodness. □

Lemma 2 (Multilevel quality). *For all $\mu, 0 < \delta \leq 0.5$, chain $\mathcal{C}$ containing only honestly-generated blocks in an optimistic execution has (δ, k_1) multilevel quality at level μ with overwhelming probability in k_1.*

Proof. Identical. □

Lemma 3 (Superquality). *For all $\mu, \delta > 0$, a chain $\mathcal{C}$ adopted in an optimistic execution has (δ, m)-superquality at level μ with overwhelming probability in m.*

Proof. Let $\mathcal{C}' = \mathcal{C}{\uparrow^\mu}$ and let $\mathcal{C}^* = \mathcal{C}'[-m':]$ for some $m' \geq m$. Then let $B \in \mathcal{C}^*{\downarrow}$ and let X_B be the random variable equal to 1 if $level(B) \geq \mu$ and 0 otherwise. $\{X_B : B \in \mathcal{C}^*\}$ are mutually independent Bernoulli random variables with expectation $E(X_B) = 2^{-\mu}|\mathcal{C}^*{\downarrow}\ |$. Let $X = \sum_{B \in \mathcal{C}^*{\downarrow}} X_B$. Then X follows a Binomial distribution with parameters $(m', 2^{-\mu})$ and note that $|\mathcal{C}^*| = X$. Then $\mathbb{E}(|\mathcal{C}^*{\downarrow}\ |) = 2^{-\mu}|\mathcal{C}^*|$. Applying a Chernoff bound on $|\mathcal{C}^*{\downarrow}\ |$ we obtain $\Pr[|\mathcal{C}^*{\downarrow}\ | \leq (1-\delta)2^{-\mu}|\mathcal{C}^*{\downarrow}] \leq \exp(-\delta^2 2^{-\mu-1}|\mathcal{C}^*|)$. □

Lemma 4 (Optimistic superchain distribution). *For a level μ, and $0 < \delta < 0.5$, a chain $\mathcal{C}$ containing only honestly-generated blocks adopted by an honest party in an execution with random scheduling is (δ, m)-good at level μ with overwhelming probability in m.*

Proof. This is a direct consequence of Lemma 3 and Lemma 2. □

4 An attack against the PoPoW of [13]

In this section, we revisit the construction for interactive proofs of proof-of-work from [13] and its security. Their construction is also the starting point for our non-interactive PoPoW. We show that their construction is susceptible to a double-spending attack even in the case the adversary controls a minority of the hashing power.

We proceed in two steps. We first show that a powerful attacker can break chain superquality with non-negligible probability. Then we construct a concrete double spending attack based on this observation. Note that maintaining chain superquality was not in the original security model; however, we show how the property affects the security of the underlying blockchain proofs. Our concrete attack works under the assumption that the adversary has sufficient hashing power, but still below 50%.

4.1 The interactive proofs of proof-of-work protocol

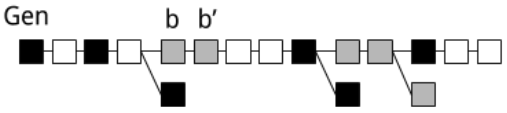
The interactive PoPoW of [13] works as follows. The main algorithm of the verifier aims at distinguishing between two candidate proofs (π_A, χ_A) and (π_B, χ_B). The honest prover, having adopted $\mathcal{C}_B$ during mining, initially produces the proof (π_B, χ_B) as follows. First, the last k blocks are sent literally as $\chi_B = \mathcal{C}_B[-k:]$. Then for the first part of the chain,

$\mathcal{C}_B[:-k]$, the prover sets π_B to be the μ-superchain spanning $\mathcal{C}_B$ for the largest μ such that $|\pi_B| = m$, where m is the protocol's security parameter. The verifier ensures that $|\pi_A| \geq m, |\pi_B| \geq m$ so that the proofs are not shorter than m and then checks whether $\pi_A = \pi_B$; if so, the decision is drawn immediately based on χ_A, χ_B without interaction. Otherwise, the verifier queries the provers for their claimed anchored superchains $\mathcal{C}_A{\uparrow^\mu}$, $\mathcal{C}_B{\uparrow^\mu}$ at some level μ. The verifier starts querying at the highest possible level μ and descends until level μ is sufficiently low such that $b = LCA(\pi_A{\uparrow^\mu}, \pi_B{\uparrow^\mu})$ is m blocks from the tip of the chain for one of the proofs. That is, the querying stops at such μ when $max(|\pi_A{\uparrow^\mu}\ \{b:\}|, |\pi_B{\uparrow^\mu}\ \{b:\}|) \geq m$. The winner is designated as the prover with the most blocks after b at that level; i.e., A, if $|\pi_A{\uparrow^\mu}\ \{b:\}| \geq |\pi_B{\uparrow^\mu}\ \{b:\}|$, and B otherwise. The communication overhead is reduced by only requesting blocks after the purported LCA. The security parameter m ensures, through the application of a binomial Chernoff bound, that the probability of the attacker producing a long superchain is negligible.

4.2 Attacking chain superquality

We construct an adversary $\mathcal{A}$ that breaks this property at level μ. $\mathcal{A}$ works as follows. Assume she wants to attack the honest party B in order to have him adopt chain $\mathcal{C}_B$ which has a bad distribution of superblocks, i.e. such that local goodness is violated in some sufficiently long subchain. She continuously determines the current chain $\mathcal{C}_B$ adopted by B. The adversary starts playing after $|\mathcal{C}_B| \geq 2$. If $level(\mathcal{C}_B[-1]) < \mu$, then $\mathcal{A}$ remains idle. However, if $level(\mathcal{C}_B[-1]) \geq \mu$, then $\mathcal{A}$ attempts to mine an adversarial block b on top of $\mathcal{C}_B[-2]$. If successful, she attempts to mine another block b' on top of b. If successful again, she broadcasts b and b'. The adversarial mining continues until B adopts a new chain, which can be due to two reasons: Either the adversary successfully mined b, b' on top of $\mathcal{C}_B[-2]$ and B adopts them; or one of the honest parties mined a block which was adopted by B. In either case, the adversary restarts the strategy by inspecting $\mathcal{C}[-1]$ and acting accordingly. An execution of this attack is illustrated in Figure 2.

Fig. 2. The superquality attack. Gray blocks are adversarially mined 0-blocks. Black blocks are μ-superblocks.



Assume now that an honestly-generated μ-superblock was adopted by B at position $\mathcal{C}_B[i]$ at round r. We now examine the probability that $\mathcal{C}_B[i]$ will remain a μ-superblock in the long run. Suppose $r' > r$ is the first round after r during which a block is generated. $\mathcal{A}$ will succeed in this attack with non-negligible probability and cause B to abandon the μ-superblock from their adopted chain. Therefore, there exists δ such that the adversary will be able to cause δ-variance with non-negligible probability in m. This suffices to show that superquality is violated.

As seen in the illustration, while the honest parties have generated several μ-superblocks, some of them are in blockchain forks which have been abandoned, causing a superquality harm.

4.3 A double-spending attack

Extending the above attack, we modify the superquality attacker into an attacker that causes a double spending attack in the PoPoW construction. We first give a sketch of the attack.

As before, $\mathcal{A}$ targets the proofs generated by honest party B by violating μ-superquality in B's adopted chain. $\mathcal{A}$ begins by remaining idle until a certain chosen block b. After block b is produced, $\mathcal{A}$ starts mining a secret chain which forks off from b akin to a selfish mining attacker [23]. The adversary performs a normal spending transaction on the honestly adopted blockchain and has it confirmed in the block immediately following block b. She also produces a double spending transaction which she secretly confirms in her secret chain in the block immediately following b.

$\mathcal{A}$ keeps extending her own secret chain as usual. However, whenever a μ-superblock is adopted by B, she temporarily pauses mining in her secret chain and devotes her mining power to harm the μ-superquality of B's adopted chain. Intuitively, for large enough μ, the time spent trying to harm superquality will be limited, because the probability of a μ-superblock occurring will be small. Therefore, the adversary's superchain quality will be larger than the honest parties' superchain quality (which will be harmed by the adversary) and therefore, even though the adversary's 0-chain will be shorter than the honest parties' 0-chain, the adversary's μ-superchain will be longer than the honest parties' μ-superchain and thus will be favored by the verifier! The formal calculation of the probability of this attack succeeding is in the next section. We note that actually, for appropriate choice of system parameters, the attack can be made to succeed with overwhelming probability.

Remark. It is worth isolating the mistake in the security proof from the interactive construction paper [13]. Suppose player B is honest and player $\mathcal{A}$ is adversarial and suppose b, the LCA block, was honestly generated and suppose that the superchain comparison happens at level μ. Their security proof then argues that there will have been more honestly- than adversarially-generated μ-superblocks after block b. Nevertheless, we observe that the mere fact that there have been more honestly- than adversarially-generated μ-superblocks after b does not imply that $|\overline{\pi}_{\mathcal{A}}\uparrow^{\mu} \{b :\}| \leq |\overline{\pi}_{B}\uparrow^{\mu} \{b :\}|$. The reason is that some of these superblocks could belong to blocktree forks that have been abandoned by B. Thus, the security conclusion does not follow. Regardless, their basic argument and construction is what we will use as a basis for constructing a system that is both provably secure and succinct under the same assumptions, albeit requiring a more complicated construction structure to obtain security.

We now calculate the exact attack probabilities [5]. We simplify the attack to ease the probabilistic analysis. The attack is parameterized by parameters r, μ which are picked by the adversary. μ is the superblock level at which the adversary will produce a proof longer than the honest proof. The modified attack works as follows: Without loss of generality, fix block b to be Genesis. The adversary always mines on the secret chain which forks off from genesis, unless a *superblock generation event* occurs. If a superblock generation event occurs, then the adversary pauses mining on the secret chain and attempts a *block suppression attack* on the honest chain. The adversary devotes exactly r rounds to this suppression attack; then resumes mining on the secret chain. We show that, despite this simplification (of fixing r) which is harmful to the adversary, the probability of a successful attack is non-negligible for certain values of the protocol parameters.

The adversary monitors the network for superblock generation events. Whenever an honest party diffuses an honestly-generated μ-superblock at the end of a given round r_1,

[5] We wish to thank Giorgos Panagiotakos, Peter Gaži, and Nikos Leonardos for their insights.

the superblock generation event will have occurred and the adversary will starting devoting their mining power to block suppression starting from the next round.

A block suppression attack works as follows. Let b be the honestly generated μ-superblock which was diffused at the end of the previous round. If the round was not uniquely successful, let b be any of the diffused honestly-generated μ-superblocks. Let b be the tip of an honest chain $\mathcal{C}_B$. The adversary first mines on top of $\mathcal{C}_B[-2]$. If she is successful in mining a block b', she continues extending the chain ending at b' (to mine b'' and so on). The value r is fixed, so the adversary devotes exactly r rounds to this whole process; the adversary will keep mining on top of $\mathcal{C}_B[-2]$ (or one of the adversarially-generated extensions of it) for exactly r rounds, regardless of whether b' or b'' have been found. At the same time, the honest parties will be mining on top of b (or a competing block in the case of a non-uniquely successful round). Again, further successful block diffusion by the honest parties shall not affect that the adversary is going to spend exactly r rounds for suppression. This attack will succeed with overwhelming probability for the right choice of protocol values.

Theorem 1 (Double-spending attack). *There exist parameters p, n, t, q, μ, δ, with $t \leq (1-\delta)(n-t)$, and a double spending attack against KLS PoPoW that succeeds with overwhelming probability.*

Proof. Recall that in the backbone notation n denotes the total number of parties, t denotes the number of adversarial parties, q denotes the number of the random oracle queries allowed per party per round and p is the probability that one random oracle query will be successful and remember that $p = T/2^{\kappa}$ where T is the mining target and κ is the security parameter (or hash function bit count). Then f denotes the probability that a given round is successful and we have that $f = 1 - (1-p)^{q(n-t)}$. Recall further that a requirement of the backbone protocol is that the honest majority assumption is satisfied, that is that $t \leq (1-\delta)(n-t)$ were $\delta \geq 2f + 3\epsilon$, where $\epsilon \in (0, 1)$ is an arbitrary small constant describing the quality of the concentration of the random variables.

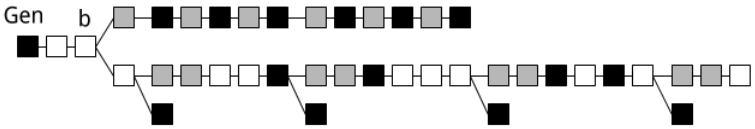
Denote $\alpha_{\mathcal{A}}$ the secret chain generated by the adversary and α_B the honest chain belonging to any honest party. We will show that for certain protocol values we have that $\Pr[|\alpha_{\mathcal{A}}\uparrow^{\mu}| \geq |\alpha_B\uparrow^{\mu}|]$ is overwhelming.

Assume that, to the adversary's harm and to simplify the analysis, the adversary plays at beginning of every round and does not perform adversarial scheduling. At the beginning of the round when it is the adversary's turn to play, she has access to the blocks diffused during the previous round by the honest parties.

First, observe that at the beginning of each round, the adversary finds herself in one of two different situations: Either she has been forced into an r-round-long period of suppression, or she is not in that period. If she is within that period, she blindly performs the suppression attack without regard for the state of the world. If she is not within that period, then she must initially observe the blocks diffused at the end of the previous round by the honest parties. Call these rounds during which the diffused data must be examined by the adversary *decision rounds*. Let there be ω decision rounds in total. In each such decision round, it is possible that the adversary discovers a diffused μ-superblock and therefore decides that a suppression attack must be performed starting with the current round. Call these rounds during which this discovery is made by the adversary *migration rounds*. Let there be y migration rounds in total. The adversary devotes the migration round to performing the suppression attack as well as $r-1$ non-migration rounds after the migration round. Call these rounds, including the migration round, *suppression rounds*. In the rest of the decision rounds, the adversary will not find any μ-superblocks diffused.

Call these *secret chain rounds*; these are rounds where the adversary devotes her queries to mining on the secret chain. Let there be x secret chain rounds. If the adversary devotes ω decision rounds to the attack in total, then clearly we have that $\omega = x + y$. If the total number of rounds during which the attack is running is s then we also have that $s = x+ry$, because for each migration round there are $r - 1$ non-decision rounds that follow.

Fig. 3. The double spending attack. The top chain fork is wholly adversarially mined, while the bottom is honestly adopted. Gray blocks are adversarially mined 0-blocks. Black blocks are μ-superblocks.



We will analyze the honest and adversarial superchain lengths with respect to ω, which roughly corresponds to time (because note that $\omega \geq s/r$, and so ω is proportional to the number of rounds). Let us calculate the probability p_{SB} ("superblock probability") that a decision round ends up being a migration round. Ignoring the negligible event that there will be random oracle collisions, we have that $p_{SB} = (n - t)qp2^{-\mu}$.

Given this, note that the decision taken at the beginning of each decision round follows independent Bernoulli distributions with probability p_{SB}. Denote z_i the indicator random variable indicating whether the decision round was a migration round. Therefore we can readily calculate the expectations for the random variables x and y, as $x = \omega - y$, $y = \sum_{i=1}^{\omega} z_i$. We have $E[x] = (1 - p_{SB})\omega$ and $E[y] = p_{SB}\omega$. Applying a Chernoff bound to the random variables x and y, we observe that they will attain values close to their mean for large ω and in particular $\Pr[y \geq (1 + \delta)E[y]] \leq \exp(-\frac{\delta^2}{3}E[y])$ and similarly $\Pr[x \leq (1 - \delta)E[x]] \leq \exp(-\frac{\delta^2}{2}E[x])$, which are negligible in ω.

Given that there will be x secret chain rounds, we observe that the random variable indicating the length of the secret adversarial superchain follows the binomial distribution with xtq repetitions and probability $p2^{-\mu}$. We can now calculate the expected secret chain length as $E[|\alpha_{\mathcal{A}}\uparrow^{\mu}|] = xtqp2^{-\mu}$. Observe that in this probability we have given the adversary the intelligence to continue using her random oracle queries during a round even after a block has been found during a round and not to wait for the next round. Applying a Chernoff bound, we obtain that $\Pr[|\alpha_{\mathcal{A}}\uparrow^{\mu}| \leq (1 - \delta)E[|\alpha_{\mathcal{A}}\uparrow^{\mu}|]] \leq exp(-\frac{\delta^2}{2}E[|\alpha_{\mathcal{A}}\uparrow^{\mu}|])$, which is negligible in ω (because we know that with overwhelming probability $x > (1 - \delta)(1 - p_{SB})\omega$).

It remains to calculate the behavior of the honest superchain. Suppose that a migration round occurs during which at least one superblock B is diffused. We will now calculate the probability p_{sup} that the adversary is able to suppress that block after r rounds by performing the suppression attack and cause all honest parties to adopt a chain not containing B.

One way for this to occur is if the adversary has generated exactly 2 shallow blocks (blocks which are not μ-superblocks) after exactly r rounds and the honest parties having generated exactly 0 blocks after exactly r rounds. This provides a lower bound for the probability, which is sufficient for our purposes. Call ADV-WIN the event where the adversary has generated exactly 2 shallow blocks after exactly r rounds since the diffusion of B and call HON-LOSE the event where the honest parties have generated exactly 0 blocks after exactly r rounds since the diffusion of B.

The number of blocks generated by the adversary after the diffusion of B follows the binomial distribution with r repetitions and probability p_{LB}, where p_{LB} denotes the probability that the adversary is able to produce a shallow block ("low block probability") during a single round. We have that $p_{LB} = tqp(1-2^{-\mu})$. To evaluate Pr[ADV-WIN], we evaluate the binomial distribution for 2 successes to obtain $\Pr[\text{ADV-WIN}] = \frac{r(r-1)}{2} p_{LB}^2 (1-p_{LB})^{r-2}$. The number of blocks generated by the honest parties after the diffusion of B follows the binomial distribution with r repetitions and probability f. To evaluate Pr[HON-LOSE], we evaluate the binomial distribution for 0 successes to obtain $\Pr[\text{HON-LOSE}] = (1-f)^r$. Note that this is an upper bound in the probability, in particular because there can be multiple blocks during a non-uniquely successful round during which a μ-superblock was generated.

Then observe that the two events ADV-WIN and HON-LOSE are independent and therefore $p_{sup} = \Pr[\text{ADV-WIN}] \Pr[\text{HON-LOSE}] = \frac{r(r-1)}{2} p_{LB}^2 (1-p_{LB})^{r-2} (1-f)^r$.

Now that we have evaluated p_{sup}, we will calculate the honest chain length in two chunks: The superblocks generated and adopted by the honest parties during secret chain rounds, $\mathcal{C}_1$, and the superblocks generated and adopted by the honest parties during suppression rounds, $\mathcal{C}_2$ (and note that these sets of blocks are not blockchains on their own).

$|\mathcal{C}_1|$ is a random variable following the binomial distribution with $s(n-t)q$ repetitions and probability $p2^{-\mu}(1-p_{sup})$. In the evaluation of this distribution, we give the honest parties the liberty to belong to a mining pool and share mining information within a round, an assumption which only makes matters for the adversary worse. We can now calculate the expected length of $\mathcal{C}_1$ to find $E[|\mathcal{C}_1|] = s(n-t)qp2^{-\mu}(1-p_{sup})$. Applying a Chernoff bound, we find that $\Pr[|\mathcal{C}_1| \geq (1+\delta)E[|\mathcal{C}_1|]] \leq exp(-\frac{\delta^2}{3}E[|\mathcal{C}_1|])$, which is negligible in s.

Finally, some additional μ-superblocks could have been generated by the honest parties while the adversary is spending r rounds attempting to suppress a previous μ-superblock. These μ-superblocks will be adopted in the case the adversary fails to suppress the previous μ-superblock. As the adversary does not devote any decision rounds to these new μ-superblocks, they will never be suppressed if the previous μ-superblock is not suppressed. We collect these in the set $\mathcal{C}_2$. To calculate $|\mathcal{C}_2|$, observe that the number of unsuppressed μ-superblocks which caused an adversarial suppression period is $|\mathcal{C}_1|$. For each of these blocks, the honest parties spend r rounds attempting to form further μ-superblocks on top. The total number of such attemps is $r|\mathcal{C}_1|$. Therefore, the number of further honestly generated μ-superblocks attained during the $|\mathcal{C}_1|$ different r-round periods follows a binomial distribution with $|\mathcal{C}_1|rq(n-t)$ repetitions and probability $p2^{-\mu}$. Here we allow the honest parties to use repeated queries within a round even after a shallow success and to work in a pool to obtain an upper bound for the expectation. Therefore $E[|\mathcal{C}_2|] = |\mathcal{C}_1|rq(n-t)p2^{-\mu}$ and applying a Chernoff bound we obtain that $\Pr[|\mathcal{C}_2| \geq (1+\delta)E[|\mathcal{C}_2|]] \leq exp(-\frac{\delta}{3}E[|\mathcal{C}_2|])$, which is negligible in s and has a quadratic error term. We deduce that $|\mathcal{C}_2|$ will have a very small length compared to the rest of the honest chain, as it is a vanishing term in μ.

Concluding the calculation of the adversarial superchain, we get $E[|\alpha_B{\uparrow^\mu}|] = E[|\mathcal{C}_1|] + E[|\mathcal{C}_2|]$.

Finally, it remains to show that there exist values $p, n, t, q, r, \mu, \delta$ such that a $E[|\alpha_{\mathcal{A}}{\uparrow^\mu}|] \geq (1+\delta)E[|\alpha_B{\uparrow^\mu}|]$. Using the values $p = 10^{-5}, q = 1, n = 1000, t = 489, \mu = 25, r = 200$, we observe that the honest majority assumption is preserved. Replacing these values into the expectations formulae above, we obtain $E[|\alpha_{\mathcal{A}}{\uparrow^\mu}] \approx 1.457 * 10^{-10} * \omega$ and $E[|\alpha_B{\uparrow^\mu}] \approx 1.424 * 10^{-10} * \omega$, which result to a constant gap δ. Because the adversarial chain grows linearly in ω, the adversary only has to wait sufficient rounds for obtaining m blocks to create a valid proof. Therefore, for these values, the adversary will be able to generate a

convincing PoPoW at some level μ which is longer than the honest parties' proof, even when the adversary does not have a longer underlying blockchain. □

5 Blockchain suffix proofs

We now provide a concrete NIPoPoW construction which allows proving certain predicates Q of the chain $\mathcal{C}$. Among the predicates which are stable, in this section, we will limit ourselves to *suffix sensitive* predicates. These predicates are easier to prove, but at the same time allow us to use the construction as a scaffold for the more general setting to be presented in Section 6.

Definition 9 (Suffix sensitivity). *A chain predicate Q is called* k-suffix sensitive *if for all chains $\mathcal{C}, \mathcal{C}'$ with $|\mathcal{C}| \geq k$ and $|\mathcal{C}'| \geq k$ such that $\mathcal{C}[-k:] = \mathcal{C}'[-k:]$ we have that $Q(\mathcal{C}) = Q(\mathcal{C}')$.*

Given the suffix-sensitivity definition, we deduce that for a suffix-sensitive predicate Q there must be a quick way of deducing the value of $Q(\mathcal{C})$ by only examining the k-suffix of the chain.

Beyond acting as a building block for our more involved construction of Section 6, suffix-sensitive predicates are still useful in practice. Such predicates can describe an event that is visible at the exact block $\mathcal{C}[-k-1]$ and that will remain true as the chain grows in perpetuity (due to monotonicity). These predicates make sense for blockchains such as Ethereum which maintain state [24] that is propagated and committed to in every block. One example of such predicate is the existence of a particular Ethereum account.

5.1 Construction

We present a generic form of the verifier first and the prover afterwards. The generic form of the verifier works with any practical suffix proof protocol. Therefore, we describe the generic verifier first before we talk about the specific instantiation of our protocol. The generic verifier is given access to call a protocol-specific proof comparison operator $\leq_m$ that we define. We begin the description of our protocol by first illustrating the generic verifier. Next, we describe the prover specific to our protocol. Finally, we show the instantiation of the $\leq_m$ operator, which plugs into the generic verifier to make a concrete verifier for our protocol.

The generic verifier. The Verify function of our NIPoPoW construction for suffix predicates is described in Algorithm 2. The verifier algorithm is parameterized by a chain predicate Q and security parameters k, m; k pertains to the amount of proof-of-work needed to bury a block so that it is believed to remain stable (e.g., $k = 6$); m is a security parameter pertaining to the prefix of the proof, which connects the genesis block to the k-sized suffix. The verifier receives several proofs by different provers in a collection of proofs $\mathcal{P}$. Iterating over these proofs, it extracts the best.

Each proof is a chain. For honest provers, these are subchains of the adopted chain. Proofs consist of two parts, π and χ; $\pi\chi$ must be a valid chain; χ is the proof suffix; π is the prefix. We require $|\chi| = k$. For honest provers, χ is the last k blocks of the adopted chain, while π consists of a selected subset of blocks from the rest of their chain preceding χ. The method of choice of this subset will become clear soon.

The verifier compares the proofs provided to it by calling the $\geq_m$ operator. We will get to the operator's definition shortly. Proofs are checked for validity before comparison by ensuring $|\chi| = k$ and calling *validChain* which checks if $\pi\chi$ is an anchored blockchain.

Algorithm 2 The Verify algorithm for the NIPoPoW protocol

function $\mathsf{Verify}^{Q}_{m,k}(\mathcal{P})$
  $\tilde{\pi} \leftarrow$ (Gen) ▷ Trivial anchored blockchain
  for $(\pi, \chi) \in \mathcal{P}$ **do**
    if $\mathsf{validChain}(\pi\chi) \wedge |\chi| = k \wedge \pi \geq_m \tilde{\pi}$ **then**
      $\tilde{\pi} \leftarrow \pi$
      $\tilde{\chi} \leftarrow \chi$ ▷ Update current best
    end if
  end for
  return $\tilde{Q}(\tilde{\chi})$
end function

The verifier loops through all candidate proofs π and extracts the best one $\tilde{\pi}$ using the $\geq_m$ operator for comparisons; $\tilde{\chi}$ is set to be the suffix associated with the best known prefix. While $\tilde{\chi}$ is needed for the final predicate evaluation, it is not used as part of any comparison, as it has the same size k for all proofs.

After the end of the for loop, the verifier will have determined the best proof $(\tilde{\pi}, \tilde{\chi})$. We will later prove that this proof will necessarily belong to an honest prover with overwhelming probability. Since the proof has been generated by an honest prover, it is associated with an underlying honestly adopted chain $\mathcal{C}$. The verifier then extracts the value of the predicate Q on the underlying chain.

The concrete prover. The NIPoPoW honest prover construction is shown in Algorithm 3. The honest prover is supplied with an honestly adopted chain $\mathcal{C}$ and security parameters m, k, δ and returns proof $\pi\chi$, which is a chain. The suffix χ is the last k blocks of $\mathcal{C}$. The prefix π is constructed by selecting various blocks from $\mathcal{C}[: -k]$ and adding them to π, which consists of a number of blocks for every level μ. At the highest possible level at which at least m blocks exist, all these blocks are included. Then, inductively, for every superchain of level μ that is included in the proof, the suffix of length m is taken. Then the underlying superchain of level $\mu - 1$ spanning the same blocks as that suffix is also included, until level 0 is reached. This underlying superchain will have $2m$ blocks in expectation and always at least m blocks.

Algorithm 3 The Prove algorithm for the NIPoPoW

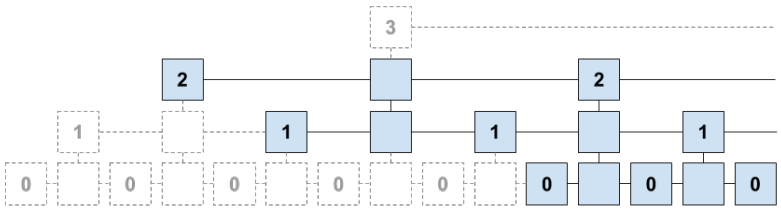
function $\mathsf{Prove}_{m,k,\delta}(\mathcal{C})$
  $B \leftarrow \mathcal{C}[0]$ ▷ Genesis
  for $\mu = |\mathcal{C}[-k].\mathsf{interlink}|$ down to 0 **do**
    $\alpha \leftarrow \mathcal{C}[: -k]\{B :\}\uparrow^{\mu}$
    $\pi \leftarrow \pi \cup \alpha$
    if $\mathrm{good}_{\delta,m}(\mathcal{C}, \alpha, \mu)$ **then**
      $B \leftarrow \alpha[-m]$
    end if
  end for
  $\chi \leftarrow \mathcal{C}[-k :]$
  return $\pi\chi$
end function

When we take a μ-superchain and are interested in its last m blocks, we fill the same range of blocks with blocks from the superchain of level $\mu - 1$ below. All the μ-superblocks which are within this m blocks range will also be $(\mu - 1)$-superblocks and so we do not want to keep them in the proof twice. Note that no check is necessary to make sure the

top-most level has at least m blocks, even though the verifier requires this. The reason is the following: Assume the blockchain has at least m blocks in total. Then, when a superchain of level μ has less than m blocks in total, these blocks will all be necessarily included into the proof by a lower-level superchain $\mu - i$ for some $i > 0$. Therefore, it does not hurt to add them to π earlier.

Figure 4 contains an example proof constructed for parameters $m = k = 3$. The top superchain level which contains at least m blocks is level $\mu = 3$. For the m-sized suffix of that level, 5 blocks of superblock level 2 are included for support spanning the same range. For the last 3 blocks of the level 2 superchain, blocks of level 1 are included for support.

Fig. 4. NIPoPoW prefix π for $m = 3$ on a good chain.



The concrete verifier. The $\geq_m$ operator which performs the comparison of proofs is presented in Algorithm 4. It takes proofs π_A and π_B and returns true if the first proof is winning, or false if the second is winning. It first computes the LCA block b between the proofs. As parties A and B agree that the blockchain is the same up to block b, arguments will then be taken for the diverging chains after b. The best possible argument from each player's proof is extracted by calling the best-arg$_m$ function. To find the best argument of a proof π given b, best-arg$_m$ collects all the μ indices which point to superblock levels that contain valid arguments after block b. Argument validity requires that there are at least m μ-superblocks following block b, which is captured by the comparison $|\pi{\uparrow^\mu}\ \{b :\}| \geq m$. 0 is always considered a valid level, regardless of how many blocks are present there. These level indices are collected into set M. For each of these levels, the score of their respective argument is evaluated by weighting the number of blocks by the level as $2^\mu |\pi{\uparrow^\mu}\ \{b :\}|$. The highest possible score across all levels is returned. Once the score of the best argument of both A and B is known, they are directly compared and the winner returned. An advantage is given to the first proof in case of a tie by using the $\geq$ operator favouring A.

Algorithm 4 The algorithm implementation for the $\geq$ operator to compare two proofs in the NIPoPoW protocol parameterized with security parameter m. Returns True if the underlying chain of player A is deemed longer than the underlying chain of player B

function best-arg$_m(\pi, b)$
  $M \leftarrow \{\mu : |\pi{\uparrow^\mu}\ \{b :\}| \geq m\} \cup \{0\}$
  return $\max_{\mu \in M}\{2^\mu \cdot |\pi{\uparrow^\mu}\ \{b :\}|\}$
end function
operator $\pi_A \geq_m \pi_B$
  $b \leftarrow (\pi_{\mathcal{A}} \cap \pi_B)[-1]$ ▷ LCA
  return best-arg$_m(\pi_A, b) \geq$ best-arg$_m(\pi_B, b)$
end operator

5.2 Security

Theorem 2. *The non-interactive proofs-of-proof-of-work construction for k-stable monotonic suffix-sensitive predicates is secure with overwhelming probability in κ.*

We will first give a draft of the proof before getting into the technical details. Suppose an adversary produces a proof $\pi_{\mathcal{A}}$ and an honest party produces a proof π_B such that the two proofs cause the predicate Q to evaluate to different values, while at the same time all honest parties have agreed that the correct value is the one obtained by π_B. Because of bitcoin's security, $\mathcal{A}$ will be unable to make these claims for an actual underlying 0-level chain. We now argue that the operator $\leq_m$ will signal in favour of the honest parties.

Suppose b is the LCA block between $\pi_{\mathcal{A}}$ and π_B. If the chain forks at b, there can be no more adversarial blocks after b than honest blocks after b, provided there are at least k honest blocks (due to the Common Prefix property). We will now argue that, further, there can be no more disjoint $\mu_{\mathcal{A}}$-level superblocks than honest μ_B-level superblocks after b.

To see this, let b be an honest block generated at some round r_1 and let the honest proof have been generated at some round r_3. Then take the sequence of consecutive rounds $S = (r_1, \cdots, r_3)$. Because the verifier requires at least m blocks from each of the provers, the adversary must have m $\mu_{\mathcal{A}}$-superblocks in $\pi_{\mathcal{A}}\{b:\}$ which are not in $\pi_B\{b:\}$. Therefore, using a negative binomial tail bound argument, we see that $|S|$ must be long; intuitively, it takes a long time to produce a lot of blocks $|\pi_{\mathcal{A}}\{b:\}|$. Given that $|S|$ is long and that the honest parties have more mining power, they must have been able to produce a longer $\pi_B\{b:\}$ argument (of course, this comparison will have the superchain lengths weighted by $2^{\mu_{\mathcal{A}}}, 2^{\mu_B}$ respectively). To prove this, we use a binomial tail bound argument; intuitively, given a long time $|S|$, a lot of μ_B-superblocks $|\pi_B\{b:\}|$ will have been honestly produced.

We therefore have a fixed value for the length of the adversarial argument, a negative binomial random variable for the number of rounds, and a binomial random variable for the length of the honest argument. By taking the expectations of the above random variables and applying a Chernoff bound, we see that the actual values will be close to their means with overwhelming probability, completing the proof.

We now go on to formalize the above proof sketch.

Assume t adversarial and n total parties, each with q PoW random oracle queries per round. We will call a query to the RO *μ-successful* if the RO returns a value h such that $h \leq 2^{-\mu}T$.

We define boolean random variables X_r^μ, Y_r^μ and Z_r^μ. Fix some round r, query index j and adversarial party index k (out of t). If at round i an honest party obtains a PoW with $id < 2^{-\mu}T$, set $X_r^\mu = 1$, otherwise $X_r^\mu = 0$. If at round r exactly one honest party obtains a PoW with $id < 2^{-\mu}T$, set $Y_r^\mu = 1$, otherwise $Y_r^\mu = 0$. If at round r the j-th query of the k-th corrupted party is μ-successful, set $Z_{ijk}^\mu = 1$, otherwise $Z_{ijk}^\mu = 0$. Let $Z_r^\mu = \sum_{k=1}^{t}\sum_{j=1}^{q} Z_{ijk}^\mu$. For a set of rounds S, let $X^\mu(S) = \sum_{r\in S} X_r$ and similarly define $Y^\mu(S), Z^\mu(S)$.

Definition 10. *An execution of the protocol is (ϵ, η)-typical if:*

Block counts don't deviate. *For all $\mu \geq 0$ and any set S of consecutive rounds with $|S| \geq 2^\mu \eta\kappa$, we have:*

- $(1-\epsilon)E[X^\mu(S)] < X^\mu(S) < (1+\epsilon)E[X^\mu(S)]$ *and* $(1-\epsilon)E[Y^\mu(S)] < Y^\mu(s)$.
- $Z^\mu(S) < (1+\epsilon)E[Z^\mu(S)]$.

Round count doesn't deviate. *Let S be a set of consecutive rounds such that $Z^{\mu}(S) \geq k$ for some security parameter k. Then $|S| \geq (1-\epsilon)2^{\mu}\frac{k}{pqt}$ with overwhelming probability in k.*

Chain regularity. *No insertions, no copies, and no predictions [12] have occurred.*

Theorem 3 (Typicality). *Executions are (ϵ, η)-typical with overwhelming probability in κ.*

Proof. **Block counts and regularity.** For the blocks count and regularity, we refer the reader to [12] for the full proof.

Round count. First, observe that $Z^{\mu}_{ijk} \sim \mathsf{Bern}(2^{-\mu}p)$ and these are jointly independent. Therefore $Z^{\mu}_{S} \sim \mathsf{Bin}(tq|S|, 2^{-\mu}p)$ and $|S| \sim \mathsf{NB}(Z_S, 2^{-\mu}p)$. So $\mathbb{E}(|S|) = 2^{\mu}\frac{Z_S}{pqt}$. Applying a tail bound to the negative binomial distribution, we obtain that $\Pr[|S| < (1-\epsilon)\mathbb{E}(|S|)] \in \Omega(\epsilon^2 m)$. □

The following lemma is at the heart of the security proof that will follow.

Lemma 5. *Suppose S is a set of consecutive rounds $r_1 \ldots r_2$ and $\mathcal{C}_B$ is a chain adopted by an honest party at round r_2 of a typical execution. Let $\mathcal{C}^S_B = \{b \in \mathcal{C}_B : b$ was generated during $S\}$. Let $\mu_{\mathcal{A}}, \mu_B \in \mathbb{N}$. Suppose $\mathcal{C}^S_B{\uparrow}^{\mu_B}$ is good. Suppose $\mathcal{C}'_{\mathcal{A}}$ is a $\mu_{\mathcal{A}}$-superchain containing only adversarially generated blocks generated during S and suppose that $|\mathcal{C}'_{\mathcal{A}}| \geq k$. Then $2^{\mu_{\mathcal{A}}}|\mathcal{C}'_{\mathcal{A}}| < \frac{1}{3}2^{\mu_B}|\mathcal{C}^S_B{\uparrow}^{\mu_B}|$.*

Proof. From $|\mathcal{C}'_{\mathcal{A}}| \geq k$ we know that $Z_S \geq k$. Applying Theorem 3, we conclude that $|S| \geq (1-\epsilon')2^{\mu_{\mathcal{A}}}\frac{1}{pqt}|\mathcal{C}'_{\mathcal{A}}|$. Applying the chain growth theorem [12] we obtain that $|\mathcal{C}^S_B| \geq (1-\epsilon)f|S|$. But from the goodness of $\mathcal{C}^S_B{\uparrow}^{\mu_B}$, we know that $|\mathcal{C}^S_B{\uparrow}^{\mu_B}| \geq (1-\delta)2^{-\mu_B}|\mathcal{C}^S_B|$. Therefore $|\mathcal{C}^S_B{\uparrow}^{\mu_B}| \geq 2^{-\mu_B}(1-\delta)(1-\epsilon)f(1-\epsilon')2^{\mu_{\mathcal{A}}}\frac{1}{pqt}|\mathcal{C}'_{\mathcal{A}}|$ and so $2^{\mu_{\mathcal{A}}}|\mathcal{C}'_{\mathcal{A}}| < \frac{pqt}{(1-\delta)(1-\epsilon')(1-\epsilon)f}2^{\mu_B}|\mathcal{C}^S_B{\uparrow}^{\mu_B}|$. □

Definition 11 (Adequate level of honest proof). *Let π be an honestly generated proof constructed upon some adopted chain $\mathcal{C}$ and let $b \in \pi$.*

Then μ' is defined as $\mu' = \max\{\mu : |\pi\{b:\}{\uparrow}^{\mu}| \geq \max(m+1, (1-\delta)2^{-\mu}|\pi\{b:\}{\uparrow}^{\mu\downarrow}|)\}$. We call μ' the adequate level of proof π with respect to block b with security parameters δ and m. Note that the adequate level of a proof is a function of both the proof π and the chosen block b.

Lemma 6. *Let π be some honest proof generated with security parameters δ, m. Let $\mathcal{C}$ be the underlying chain, $b \in \mathcal{C}$ be any block and μ' be the adequate level of the proof with respect to b and the same security parameters.*

Then $\mathcal{C}\{b:\}{\uparrow}^{\mu'} = \pi\{b:\}{\uparrow}^{\mu'}$.

Proof. $\pi\{b:\}{\uparrow}^{\mu'} \subseteq \mathcal{C}\{b:\}{\uparrow}^{\mu'}$ is trivial. For the converse, we show that for all $\mu^* > \mu'$, we have that in the iteration of the Prove `for` loop with $\mu = \mu^*$, the block stored in variable B precedes b in $\mathcal{C}$.

Suppose $\mu = \mu^*$ is the first `for` iteration during which the property is violated. This cannot be the first iteration, as there $B = \mathcal{C}[0]$ and Genesis precedes all blocks. By induction hypothesis we see that during the iteration $\mu = \mu^* + 1$, B preceded b. From the definition of μ' we know that μ' is the highest level for which $|\pi\{b:\}{\uparrow}^{\mu'}[1:]| \geq \max(m, (1-\delta)2^{-\mu'}|\pi\{b:\}{\uparrow}^{\mu'}[1:]{\downarrow}|)$.

Hence, this property cannot hold for $\mu^* > \mu'$ and therefore $|\pi_B\{b:\}{\uparrow}^{\mu^*_B}[1:]| < m$ or $\neg\mathsf{local\text{-}good}_{\delta}(\pi\{b:\}{\uparrow}\,\mu^*[1:], \mathcal{C}, \mu^*)$.

In case local-good is violated, variable B remains unmodified and the induction step holds. If local-good is not violated, then $|\pi\{b:\}\uparrow^{\mu^*}[1:]| < m$ and so $\pi\uparrow^{\mu^*}[-m]$ precedes b. □

Lemma 7. *Suppose the verifier evaluates $\pi_{\mathcal{A}} \geq \pi_B$ in a protocol interaction where B is honest and assume during the comparison that the compared level of the honest party is μ_B. Let $b = \mathsf{LCA}(\pi_{\mathcal{A}}, \pi_B)$ and let μ'_B be the adequate level of π_B with respect to b. Then $\mu'_B \geq \mu_B$.*

Proof. Because μ_B is the compared level of the honest party we have $2^{\mu_B}|\mathcal{C}\{b:\}\uparrow^{\mu_B}| > 2^{\mu_B}|\mathcal{C}\{b:\}|$. The proof is by contradiction. Suppose $\mu'_B < \mu_B$. By definition, μ'_B is the maximum level such that $|\pi_B\{b:\}\uparrow^{\mu}[1:]| \geq \max(m, (1-\delta)2^{-\mu}|\pi_B\{b:\}\uparrow^{\mu}[1:]\downarrow|)$, therefore μ_B does not satisfy this condition. But we know that $|\pi_B\{b:\}\uparrow^{\mu_B}[1:]| \geq m$ because μ_B was selected by the Verifier. Therefore $2^{\mu_B}|\mathcal{C}_B\{b:\}\uparrow^{\mu_B}| < (1-\delta)|\mathcal{C}\{b:\}|$. But μ'_B satisfies goodness, so $2^{\mu'_B}|\mathcal{C}_B\{b:\}\uparrow^{\mu'_B} > (1-\delta)|\mathcal{C}\{b:\}|$. From the last two equations, we obtain $(1-\delta)|\mathcal{C}\{b:\}| > 2^{\mu'_B}|\mathcal{C}\{b:\}\uparrow^{\mu'_B}|$, which contradicts the previous equation. □

Theorem 2. *The non-interactive proofs-of-proof-of-work construction for k-stable monotonic suffix-sensitive predicates is secure with overwhelming probability in κ.*

Proof. By contradiction. Let $m = k_1 + k_2 + k_3$ and let k_1, k_2, k_3 be polynomial functions of κ. Let Q be a k-stable suffix sensitive chain predicate. Assume NIPoPoWs on Q is insecure. Then, during an execution at some round r_3, $Q(\mathcal{C})$ is defined and the verifier V disagrees with all honest miners. Assume the execution is typical. V communicates with adversary $\mathcal{A}$ and honest prover B. The verifier receives proofs $\pi_{\mathcal{A}}, \pi_B$. Because B is honest, π_B is a proof constructed based on underlying blockchain $\mathcal{C}_B$ (with $\pi_B \subseteq \mathcal{C}_B$) which B has adopted during round r_3 at which π_B was generated. Furthermore, $\pi_{\mathcal{A}}$ was generated at round $r'_3 \leq r_3$.

The verifier outputs $\neg Q(\mathcal{C}_B)$, and so $\mathsf{Verify}^{Q}_{m,k} = \neg Q(\mathcal{C}_B)$. Thus it is necessary that $\pi_{\mathcal{A}} \geq \pi_B$, otherwise, because Q is suffix sensitive, Verify^{Q} would have returned $Q(\mathcal{C}_B)$. We now show that $\pi_{\mathcal{A}} \geq \pi_B$ is a negligible event.

Let $b = \mathsf{LCA}(\pi_{\mathcal{A}}, \pi_B)$ and let b^* be the most recent honestly generated block in $\mathcal{C}_B$ preceding b (and note that b^* necessarily exists because Genesis is honestly generated). Let the levels of comparison decided by the verifier be $\mu_{\mathcal{A}}$ and μ_B respectively. Let μ'_B be the adequate level of proof π_B with respect to block b. Call $\alpha_{\mathcal{A}} = \pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}\{b:\}$, $\alpha'_B = \pi_B\uparrow^{\mu'_B}\{b:\}$.

We show showing three successive claims: First, $\alpha_{\mathcal{A}}$ and $\alpha'_B\downarrow$ are mostly disjoint. Second, $\alpha_{\mathcal{A}}$ contains mostly adversarially-generated blocks. And third, the adversary is able to produce this $\alpha_{\mathcal{A}}$ with negligible probability.

Claim 1a: If $\mu'_B \leq \mu_{\mathcal{A}}$ then $\alpha_{\mathcal{A}}[1:]$ and $\alpha_B[1:]\downarrow$ are completely disjoint.

Applying Lemma 6 to $\mathcal{C}_B\{b:\}\uparrow^{\mu'_B}$ we see that $\mathcal{C}_B\{b:\}\uparrow^{\mu'_B} = \pi_B\uparrow^{\mu'_B}\{b:\}$ and so $\pi_B\uparrow^{\mu'_B}\{b:\}[1:] \cap \pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}\{b:\}[1:] = \emptyset$.

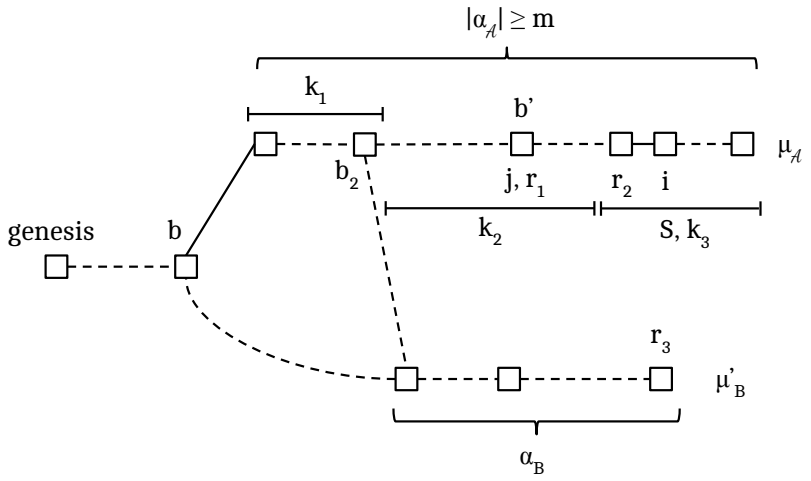
Claim 1b: If $\mu_{\mathcal{A}} < \mu'_B$ then $|\alpha_{\mathcal{A}}[1:] \cap \alpha_B\downarrow[1:]| \leq 2^{\mu'_B - \mu_{\mathcal{A}}}k_1$.

First, observe that, because the adversary is winning, therefore $|\alpha_{\mathcal{A}}| > 2^{\mu'_B-\mu_{\mathcal{A}}}m$. Suppose for contradiction that $|\alpha_{\mathcal{A}}[1:] \cap \alpha_B\downarrow[1:]| > 2^{\mu'_B-\mu_{\mathcal{A}}}k_1$. This means there are indices $1 \leq i < j$ such that $|\mathcal{C}_B\uparrow^{\mu_{\mathcal{A}}}[i:j]| > 2^{\mu'_B-\mu_{\mathcal{A}}}k_1$ but $|\mathcal{C}_B\uparrow^{\mu_{\mathcal{A}}}[i:j]\downarrow\uparrow^{\mu'_B}| = 0$. But this contradicts the goodness of $\mathcal{C}_B\uparrow^{\mu'_B}$. Therefore there are more than $2^{\mu'_B-\mu_{\mathcal{A}}}(k_2+k_3)$ blocks in $\alpha_{\mathcal{A}}$ that are not in α_B, and clearly also more than k_2+k_3 blocks.

From Claim 1a and Claim 1b, we conclude that there are at least k_2+k_3 blocks after block b in $\alpha_{\mathcal{A}}$ which do not exist in α_B. We now set $b_2 = \mathsf{LCA}(\mathcal{C}_B, \alpha_{\mathcal{A}})$.

Claim 2: At least k_3 superblocks of $\alpha_{\mathcal{A}}$ are adversarially generated.

Fig. 5. Two competing proofs at different levels.



We show this by showing that $\alpha_{\mathcal{A}}[k_2+1:]$ contains no honestly mined blocks. By contradiction, assume that the block $\alpha_{\mathcal{A}}[i]$ for some $i \geq k_1+k_2+1$ was honestly generated. This means that an honest party had adopted the chain $\alpha_{\mathcal{A}}[i-1]$ at some round $r_2 \leq r_3$. Because of the way the honest parties adopt chains, the superchain $\alpha_{\mathcal{A}}[:i-1]$ has an underlying properly constructed 0-level anchored chain $\mathcal{C}_{\mathcal{A}}$ such that $\mathcal{C}_{\mathcal{A}} \subseteq \alpha_{\mathcal{A}}[:i-1]$. Let j be the index of block b_2 within $\mathcal{C}_{\mathcal{A}}$. As $\alpha_{\mathcal{A}} \subseteq \mathcal{C}_{\mathcal{A}}$, observe that $|\mathcal{C}_{\mathcal{A}}[j+1:]| > i-1 \geq k_2+k_1$. Therefore $\mathcal{C}_{\mathcal{A}}[:-(k_2+k_1)] \not\preccurlyeq \mathcal{C}_B$. But $\mathcal{C}_{\mathcal{A}}$ was adopted by an honest party at round r_2 which is prior to round r_3 during which $\mathcal{C}_B$ was adopted by an honest party. This contradicts the Common Prefix [12] property with parameter k_2. It follows that with overwhelming probability in k_2, the $k_3 = m - k_2 - k_1$ last blocks of the adversarial proof have been adversarially mined.

Claim 3: $\mathcal{A}$ is able to produce a $\alpha_{\mathcal{A}}$ that wins against α_B with negligible probability.

Let b' be the latest honestly generated block in $\alpha_{\mathcal{A}}$, or b^* if no such block exists in $\alpha_{\mathcal{A}}$. Let r_1 be the round when b^* was generated. Let j be the index of b^*. Consider the set S of consecutive rounds $r_1 \ldots r_3$. Every block in $\alpha_{\mathcal{A}}[-k_3:]$ has been adversarially generated during S and $|\alpha_{\mathcal{A}}[-k_3:]| = k_3$. $\mathcal{C}_B$ is a chain adopted by an honest party at round r_3 and filtering the blocks by the rounds during which they were generated to obtain $\mathcal{C}_B^S$, we see that $\mathcal{C}_B^S = \mathcal{C}_B\{b^*:\}$. But chain $\mathcal{C}_B^S{\uparrow}^{\mu'_B}$ is good with respect to $\mathcal{C}_B^S$. Applying Lemma 5, we obtain that with overwhelming probability $2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}\{b':\}| < \frac{1}{3}2^{\mu'_B}|\mathcal{C}_B^S{\uparrow}\ \mu'_B|$.

But $|\alpha_B| \geq |\mathcal{C}_B^S{\uparrow}\ \mu'_B|$ and $|\alpha_{\mathcal{A}}\{b':\}| \geq |\alpha_{\mathcal{A}}| - k_2$, therefore $2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}| - k_2 < \frac{1}{3}2^{\mu'_B}|\alpha_B|$. But $|\alpha_{\mathcal{A}}| - k_2 \geq k_3$, therefore $\frac{1}{3}2^{\mu'_B}|\alpha_B| > k_3$ and so $2^{\mu'_B}|\alpha_B| > 3k_3$ Taking $k_2 = k_3$, we obtain $2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}| < \frac{1}{3}3k_3 + k_3 = 2k_3 < 2^{\mu'_B}|\alpha_B|$. But this contradicts the fact that $\pi_{\mathcal{A}} \geq \pi_A$, and so the claim is proven.

Therefore we have proven that $2^{\mu'_B}|\pi_B{\uparrow}^{\mu'_B}| > 2^{\mu_{\mathcal{A}}}|\pi_{\mathcal{A}}^{\mu_{\mathcal{A}}}|$. From the definition of μ_B, we know that $2^{\mu_B}|\pi_B{\uparrow}^{\mu_B}| \geq 2^{\mu'_B}|\pi_B{\uparrow}^{\mu'_B}|$, and therefore we conclude that $2^{\mu_B}|\pi_B{\uparrow}^{\mu_B}| > 2^{\mu_{\mathcal{A}}}|\pi_{\mathcal{A}}{\uparrow}^{\mu_{\mathcal{A}}}|$. □

Remark 1 (Variance attacks). The critical issue addressed by this security proof is to avoid Bahack-style attack [25] where the adversary constructs "lucky" high-difficulty superblocks without filling in the underlying proof-of-work in the lower levels. Observe that, while setting $m = 1$ "preserves" the proof-of-work in the sense that expectations remain the

same, the probability of an adversarial attack becomes approximately proportional to the adversary power if the adversary follows a suitable strategy (for a description of such a strategy, see the parameterization section). With higher values of m, the probability of an adversarial attack drops exponentially in m, even though they maintain constant computational power, and hence satisfy a strong notion of security.

Remark 2. Intuitively, the attack of Section 4 is neutralized, because our prover takes "goodness" of blockchains into account and the verifier does not compare proofs strictly at the same level.

5.3 Succinctness

We now prove that our construction produces succinct proofs. We first observe that full succinctness in the standard honest majority model is impossible to prove for our construction. To see why, recall that an adversary with sufficiently large mining power can significantly harm superquality as described in Section 4.2. By reducing superquality for a sufficiently low level μ, for example $\mu = 3$, the adversary can cause the honest prover to digress in their proofs from high-level superchains down to low-level superchains, causing a linear proof to be produced. For instance, if superquality is harmed at $\mu = 3$, the prover will need to digress down to level $\mu = 2$ and include the whole 2-superchain, which, in expectation, will be of size $|\mathcal{C}|/2$.

Having established security in the general case of the standard honest majority model, we now concentrate our succinctness claims to the particular "optimistic" case where the adversary does not use their (minority) computational power or network power. Therefore, the superquality of the chain must be the same as a fully honestly-generated chain generated with no network adversary. Last, for now, we will not allow the adversary to produce any proofs; that is, all proofs consumed by the verifier are honestly-generated. We will lift this last assumption shortly.

Theorem 4 (Number of levels). *The number of superblock levels which have at least m blocks are at most* $\log(|S|)$*, where S is the set of all blocks produced, with overwhelming probability in m.*

Proof. Let S be the set of all blocks successfully produced by the honest parties or the adversary. Each block id is generated by the random oracle, so $\Pr[\text{id} \leq T2^{-\mu}] = 2^{-\mu}$. These are independent Bernoulli trials. For each $B \in S$, let $X_B^\mu \in \{0, 1\}$ be the random variable indicating whether the block belongs to level μ and let D_μ indicate their sum, which is a Binomial distribution with parameters $(|S|, 2^{-\mu})$ and expectation $E[D_\mu] = |S|2^{-\mu}$.

For level μ to exist in any valid proof, at least m blocks of level μ must have been produced by the honest parties or the adversary. We show that m blocks of level $\mu = \log(|S|)$ are produced with negligible probability in m.

All of the X^μ are independent. We apply a Binomial Chernoff bound to the sum. We have $\Pr[D_\mu \geq (1 + \Delta)E[D_\mu]] \leq \exp(-\frac{\Delta^2}{2+\Delta}E[D_\mu])$. But for this μ we have that $E[D_\mu] = 1$. Therefore $\Pr[D_\mu \geq 1 + \Delta] \leq \exp(-\frac{\Delta^2}{2+\Delta})$. Requiring $1 + \Delta = m$, we get $\Pr[D_\mu \geq m] \leq \exp(-\frac{(m-1)^2}{m+1})$, which is negligible in m. □

The above theorem establishes that the number of superchains is small. What remains to be shown is that few blocks will be included at each superchain level.

Theorem 5 (Large upchain expansion). *Let $\mathcal{C}$ be an honestly generated chain and let $\mathcal{C}' = \mathcal{C}{\uparrow}^{\mu-1}[i : i + \ell]$ with $\ell \geq 4m$. Then $|\mathcal{C}'{\uparrow}^\mu| \geq m$ with overwhelming probability in m.*

Proof. Assume the $(\mu-1)$-level superchain has $4m$ blocks. Because each block of level $\mu-1$ was generated as a query to the random oracle, it constitutes an independent Bernoulli trial and the number of blocks in level μ, namely $\pi\uparrow^{\mu}$, is a Binomial distribution with parameters $(4m, 1/2)$. Clearly $\Pr[|\pi\uparrow^{\mu}| = m] \leq \Pr[|\pi\uparrow^{\mu}| \leq m]$. Observing that $E[\pi\uparrow^{\mu}] = 2m$ and applying a Chernoff bound, we get $\Pr[|\pi\uparrow^{\mu}| \leq (1-\frac{1}{2})2m] \leq \exp(-\frac{(1/2)^2}{2}2m)$ which is negligible in m.

This probability bounds the probability of fewer than m blocks occurring in the μ level restriction of $(\mu-1)$-level superchains of more than $4m$ blocks. □

Lemma 8 (Small downchain support). *Assume an honestly generated chain $\mathcal{C}$ and let $\mathcal{C}' = \mathcal{C}\uparrow^{\mu}[i : i+m]$. Then $|\mathcal{C}'\downarrow\uparrow^{\mu-1}| \leq 4m$ with overwhelming probability in m.*

Proof. Assume the $(\mu-1)$-level superchain had at least $4m$ blocks. Then by Theorem 5 it follows that more than m blocks exist in level μ with overwhelming probability in m, which is a contradiction. □

This last theorem establishes the fact that the support of blocks needed under the m-sized chain suffix to move from one level to the one below is small. Based on this, we can establish our theorem on succinctness:

Theorem 6 (Optimistic succinctness). *Non-interactive proofs-of-proof-of-work produced by honest provers in the optimistic case are succinct with the number of blocks bounded by $4m\log(|\mathcal{C}|)$, with overwhelming probability in m.*

Note the linear dependency between the round r that a proof is generated and the length $|\mathcal{C}|$ of the chain of each honest prover.

Proof. Assume $\mathcal{C}$ is an honest parties' chain. From Theorem 4, the number of levels in the NIPoPoW is at most $\log(|\mathcal{C}|)$ with overwhelming probability in m. First, observe that the count of blocks in the highest level will be less than $4m$ from Theorem 5; otherwise a higher superblock level would exist. From Corollary 4, we know that at all levels μ the chain will be good. Therefore, for each μ superchain $\mathcal{C}$ the supporting $(\mu-1)$-superchain will only need to span the m-long suffix of the μ-superchain above. For the m-long suffix of each superchain of level μ, the supporting superchain of level $\mu-1$ will have at most $4m$ blocks from Lemma 8. Therefore the size of the proof is $4m\log(|\mathcal{C}|)$. □

5.4 Certificates of badness and denial of service attacks.

In summary, with our arguments so far, we proved security in the general case (for any adversarial behavior) and succinctness only in the optimistic case where the adversary does not interfere with superchain quality. In this setting, it is possible for the adversary to produce large dummy (incorrect) proofs that expand the verification time; security will not be hurt but it would take more time to complete verification. One may dismiss this as a trivial denial of service attack (and have a resource bounded verifier simply stop if it is confronted with such a processing task). Nevertheless, it would be useful for honest provers to have the ability to signal to the verifier that such time expansion is indeed necessary because of an attack on superchain quality rather than because a malicious prover is simply sending long proofs that will eventually be rejected. With such signaling mechanism, a resource bounded verifier can distinguish between a denial of service attack that may be directed solely to it from a denial of service attack that is launched by an attacker that has the ability to interfere globally with superchain quality.

To facilitate the above signaling, we offer a simple generalization of our construction that achieves this. Specifically, we require the prover to produce a *certificate of badness* in case there is a violation of *goodness* in the blockchain. This certificate will always be logarithmic in size and must be sent prior to the rest of the proof by the prover to the verifier. Because the certificate will be logarithmic in size even in the case of an adversarial attack on the chain, the honest verifier can stop processing the certificate after a logarithmic time bound. If the certificate is claimed to be longer, the honest verifier can reject early by deciding that the prover is adversarial. Looking at the certificate, the honest verifier determines whether there is a possibility for a lack of goodness in the underlying chain. If there's no adversarial computational power in use, the certificate is impossible to produce.

The certificates of badness are produced easily as follows. First, the honest verifier finds the maximum level max-μ at which there are at least m max-μ-superblocks and includes it in the certificate. Then, because there is a violation of goodness there must exist two levels $\mu < \mu'$ such that $2^{\mu}|\mathcal{C}{\uparrow}^{\mu}| > (1+\delta)2^{\mu'}|\mathcal{C}{\uparrow}^{\mu'}|$ in some part $\mathcal{C}$ of the honestly adopted chain. But $\mu' - \mu \leq$ max-μ. Therefore, there must exist two adjacent levels $\mu_1 < \mu_2$ which break goodness but with error parameter $(1+\delta)^{1/\text{max-}\mu}$. In particular, it will hold that $2^{\mu_1}|\mathcal{C}{\uparrow}^{\mu_1}| > (1+\delta)^{1/\text{max-}\mu}2^{\mu_2}|\mathcal{C}{\uparrow}^{\mu_2}|$. This condition is direct for the prover to find and trivial for the verifier to check and completes the construction. Note that it is possible that a certificate of badness is produceable where two adjacent levels have more than $(1+\delta)^{1/\text{max-}\mu}$ error even if there is no harm to global goodness; however, these certificates cannot be produced when no adversarial power is in use. The algorithm to do this is shown in Algorithm 5.

Algorithm 5 The badness prover which generates a succinct certificate of badness

```
1: function badness_{m,δ}(C)
2:     M ← {μ : |C↑^μ| ≥ m} \ {0}
3:     ρ ← 1/max(M)
4:     for μ ∈ M do
5:         for B ∈ C↑^μ do
6:             C' ← C↑^μ {B :}[: m]
7:             if |C'| = m then                          ▷ Sliding m-sized window
8:                 C* ← C↓↑^{μ−1}
9:                 if 2|C'| < (1 − δ)^ρ |C*| then
10:                    return C*                          ▷ Chain is bad
11:                end if
12:            end if
13:        end for
14:    end for
15:    return ⊥                                           ▷ Chain is good
16: end function
```

6 Blockchain Infix proofs

In the previous section we have seen how to construct proofs for suffix predicates. As mentioned, the main purpose of this construction is to serve as a stepping stone for the construction of this section that presents a most useful class of proofs allow proving more general predicates that can depend on multiple blocks even buried deep within the blockchain.

More specifically, the generalized prover for *infix proofs* allows proving any predicate $Q(\mathcal{C})$ that depends on a number of blocks that can appear anywhere within the chain

(except the k suffix for stability). These blocks constitute a *subset* $\mathcal{C}'$ of blocks which may not necessarily be a stand-alone blockchain. This allows proving powerful statements such as, for example, whether a transaction is confirmed. We define next formally the class of predicates that will be of interest.

Definition 12 (Infix sensitivity). *A chain predicate $Q_{\ell,d,k}$ is* infix sensitive *if it can be written in the form*

$$Q_{\ell,d,k}(\mathcal{C}) = \begin{cases} \textit{undefined, if } |\mathcal{C}[:-k]| < \ell, \textit{ otherwise:} \\ \textit{true, if } \exists \mathcal{C}' \subseteq \mathcal{C}[:-k] : |\mathcal{C}'| \leq d \wedge D(\mathcal{C}') \\ \textit{false, otherwise} \end{cases}$$

Where D is an arbitrary predicate. Note that $\mathcal{C}'$ is a blockset and may not necessarily be a blockchain.

We next show how to express the predicate that asks whether a certain transaction with id $txid$ has been confirmed by a certain time as an infix sensitive predicate. First, we need specify a time bound as the maximum number of blocks ℓ up to which we want to test whether $txid$ is included (this is necessary as without such a time bound the predicate cannot be monotonic). Then we define the predicate D^{txid} that receives a single block and tests whether a transaction with id $txid$ is included. The predicate $Q^{txid}_{\ell,1,k}$ is now defined as in Definition 12 using the predicate D^{txid} and the parameter k which in this case determines the desired stability of the assertion that $txid$ is included.

Similarly to suffix-sensitive predicates, infix-sensitive predicates Q can be evaluated very efficiently. Intuitively this is possible because of their localized nature and dependency on the $D(\cdot)$ predicate which requires only a small number of blocks to conclude whether the predicate should be true. Nevertheless, some special care will be needed to ensure the condition on the length of the chain with respect to ℓ is captured.

The construction of these proofs is shown in Algorithm 7. The infix prover accepts two parameters: The chain $\mathcal{C}$ which is the full blockchain and $\mathcal{C}'$ which is a sub-blockset of the blockchain whose blocks are of interest for the predicate in question. The prover calls the previous suffix prover to produce a proof as usual. Then, having the prefix π and suffix χ of the suffix proof in hand, the infix prover adds a few auxiliary blocks to the prefix π. The prover ensures that these auxiliary blocks form a chain with the rest of the proof π. Such auxiliary blocks are collected as follows: For every block B of the subchain $\mathcal{C}'$, the immediate previous (E') and next (E) blocks in π are found. Then, a chain of blocks R which connects E back to B' is found by the algorithm followDown. If E' is of level μ, there can be no other μ-superblock between E' and B', otherwise it would have been included in π. Therefore, B' already contains a pointer to E' in its interlink, completing the chain.

Algorithm 6 The followDown function which produces the necessary blocks to connect a superblock hi to a preceeding regular block lo.

```
1: function followDown(hi, lo, depth)
2:     B ← hi
3:     aux ← [ ]
4:     μ ← level(hi)
5:     while B ≠ lo do
6:         B′ ← blockById[B.interlink[μ]]
7:         if depth[B′] < depth[lo] then
8:             μ ← μ − 1
9:         else
10:            aux ← aux ∪ {B}
11:            B ← B′
12:        end if
13:    end while
14:    return aux
15: end function
```

The way to connect a superblock to a previous lower-level block is implemented in Algorithm 6. Block B' cannot be of higher or equal level than E, otherwise it would be equal to E and the followDown algorithm would return. The algorithm proceeds as follows: Starting at block $hi = E$, it tries to follow a pointer to as far as possible. If following the pointer surpasses $lo = B'$, then the following is aborted and a lower level is tried, which will cause a smaller step within the skiplist. If a pointer was followed without surpassing B', the operation continues from the new block, until eventually B' will be reached, which concludes the algorithm.

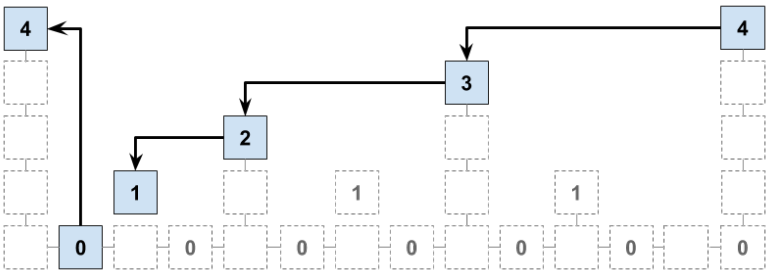
Algorithm 7 The Prove algorithm for infix proofs

```
1: function ProveInfix_{m,k}(C, C′, depth)
2:     (π, χ) ← Prove_{m,k}(C)
3:     for B′ ∈ C′ do
4:         for E ∈ π do
5:             if depth[E] ≥ depth[B′] then
6:                 R ← followDown(E, B′, depth)
7:                 aux ← aux ∪ R
8:                 break
9:             end if
10:            E′ ← E
11:        end for
12:    end for
13:    return (aux ∪ π, χ)
14: end function
```

An example of the output of followDown is shown in Figure 6. This is a portion of the proof shown at the point where the superblock levels are at level 4. A descend to level 0 was necessary so that a regular block would be included in the chain. The level 0 block can jump immediately back up to level 4 because it has a high-level pointer.

The verification algorithm must then be modified as in 8.

Fig. 6. An infix proof descend. Only blue blocks are included in the proof. Blue blocks of level 4 are part of π, while the blue blocks of level 1 and 3 are produced by followDown to get to the block of level 0 which is part of $\mathcal{C}'$.



Algorithm 8 The verify algorithm for the NIPoPoW infix protocol

function verify-infx$^{D}_{\ell,m,k}(\mathcal{P})$
  blockById $\leftarrow \emptyset$ ▷ Initialize empty hashmap
  for $(\pi, \chi) \in \mathcal{P}$ **do**
    for $B \in \pi$ **do**
      blockById$[B.\text{id}] \leftarrow B$
    end for
  end for
  $\tilde{\pi} \leftarrow$ best $\pi \in \mathcal{P}$ according to suffix verifier
  if $|\tilde{\pi}[-1].\text{index}| < \ell$ **then**
    return undefined
  end if
  return $D(\text{ancestors}(\tilde{\pi}[-1], \text{blockById}))$
end function

The algorithm works by calling the old verifier. It also maintains a blockDAG collecting blocks from all proofs (it is a DAG because *interlink* can be adversarially defined). This DAG is maintained in the blockById hashmap. Using it, ancestors uses simple graph search to extract the set of ancestors of a block present. In the final predicate evaluation, the set of ancestors of the best blockchain tip is passed to the predicate. The ancestors are included to avoid an adversary who presents an honest chain but skips the blocks of interest.

6.1 Security

The security theorem follows easily along the same lines of the security argument for suffix proofs. We state it below.

Theorem 7. *The infix NIPoPoW construction is secure for all infix-sensitive stable predicates Q, except with negligible probability in κ.*

Proof. Assume a typical execution. It suffices to show that the verifier will output the same value $Q(\mathcal{C})$ as some honest prover. Assume honest prover B has adopted a chain $\mathcal{C}$ with $Q(\mathcal{C}) = v$ and has provided proof π_B. By Theorem 2 and because the evaluation of $\tilde{\pi}$ is identical in the suffix-sensitive and in the infix-sensitive case, we deduce that $b = \tilde{\pi}[-1]$ will be an honestly adopted block. Furthermore, due to the Common Prefix property [12], b will belong to all honest parties' chains and in the same position, as it is buried under $|\tilde{\chi}| = k$ blocks.

Because Q is infix-sensitive and stable, $\exists \mathcal{C}' \subseteq \mathcal{C}[: -k] : P_v(\mathcal{C}')$. But $\mathcal{C}' \subseteq \pi_B$. Let $S = \text{ancestors}(b)$ be the ancestors evaluated by the verifier. $\mathcal{C}' \subseteq S$, therefore $Q(S) = Q(\mathcal{C}') = v$. □

6.2 Succinctness

As long as the number of blocks on which the predicate depends is polylogarithmic ($< d$) with respect to the chain length, our proofs remain succinct. Specifically, the proof size for the suffix has exactly the same size. Then the part of the proof that is of interest is the output of the followDown algorithm. However, notice that this algorithm will on average produce as many blocks as the difference of levels between B' and E, which is at most logarithmic in the chain size. Hence the proof sizes will be in expectation $(m+|\mathcal{C}'|)\log(|\mathcal{C}|)$, which remains succinct if $|\mathcal{C}'| \in O(polylog(|\mathcal{C}|))$.

7 Gradual Deployment Paths

Our construction requires an upgrade to the consensus layer. We envision that new cryptocurrencies will adopt our construction in order to support efficient light clients. However, existing cryptocurrencies could also benefit by adopting our construction as an upgrade. In this section we outline several possible upgrade paths. We also contribute a novel upgrade approach, a "velvet fork," which allows for gradual deployment without harming unupgraded miners.

7.1 Hard Forks and Soft Forks

The obvious way to upgrade a cryptocoin to support our protocol is a hard fork: the block header is modified to include the interlink structure, and the validation rules modified to require that new blocks (after a "flag day") contain a correctly-formed interlink hash.

Hard forks are not "forward compatible," and are best avoided. A soft fork construction requires including the interlink not in the block header, but in the coinbase transaction. It is enough to only store a hash of the interlink structure. The only requirement for the NiPoPoWs to work is that the PoW commits to all the pointers within interlink so that the adversary cannot cause a chain reorg. If we take that route, then each NiPoPoW will be required to present not only the block header, but also a proof-of-inclusion path within the Merkle tree of transactions proving that the coinbase transaction is indeed part of the block. Once that is established, the coinbase data can be presented, and the verifier will thereby know that the hash of the interlink data structure is correct. Given that in Bitcoin implementation there is a block size limit, observe that including such proofs-of-inclusion will only increase the NiPoPoW sizes by a constant factor per block, allowing for the communication complexity to remain polylogarithmic.

7.2 Velvet Forks

We now describe a novel upgrade path that avoids the need for a fork at all. The key idea is that clients can make use of our scheme, even if only some blocks in the blockchain include the interlink structure. Given that intuitively the changes we will propose require no rule modifications to the consensus layer, we call this technique a *velvet fork*.

We require upgraded miners to include the interlink data structure in the form of a new Merkle tree root hash in their coinbase data, similar to a soft fork. An unupgraded miner will ignore this data as comments. We further require the upgraded miners to accept all previously accepted blocks, regardless of whether they have included the interlink data structure or not. Even if the interlink data structure is included and contains invalid data, we require the upgraded miners to accept their containing blocks. Malformed interlink data could be simply of the wrong format, or the pointers could be pointing to superblocks of

incorrect levels. Furthermore, the pointers could be pointing to superblocks of the correct level, but not to the most recent block. By requiring upgraded miners to accept all such blocks, we do not modify the set of accepted blocks. Therefore, the upgrade is simply a "recommendation" for miners and not an actual change in the consensus rules. The velvet fork has the effect that blocks produced by either upgraded or unupgraded clients are valid for either. Hence, the blockchain is never forked. Only the codebase is upgraded, and the data on the blockchain is interpreted differently.

The reason this can work is because provers and verifiers of our protocol can check the validity of the claims of miners who make false interlink chain claims. An upgraded prover can check whether a block contains correct interlink data and use it. If a block does not contain correct interlink data, the prover can opt not to use those pointers in their proofs. The Verifier verifies all claims of the prover, so adversarial miners cannot cause harm by including invalid data. The one thing the Verifier cannot verify in terms of interlink claims is whether the claimed superblock of a given level is the most recent previous superblock of that level. However, an adversarial prover cannot make use of that to construct winning proofs, as they are only able to present shorter chains in that case. The honest prover can simply ignore such pointers as if they were not included at all.

The velvet prover works as usual, but additionally maintains a *realLink* data structure, which stores the *correct* interlink for each block. Whenever a new winning chain is received from the network, the prover checks it for blocks that it hasn't seen before. For those blocks, it maintains its own realLink data structure which it updates accordingly to make sure it is correct regardless of what the interlink data structure of the received block claims.

The velvet $\mathcal{C}\uparrow$ operator shown in Algorithm 9 is implemented identically as before, except that instead of following the interlink pointer blindly it now calls the helper function *followUp*, shown in Algorithm 10. It accepts block B and level μ and creates a connection from B back to the most recent preceding μ-superblock, by following the interlink pointer if it is correct. Otherwise, it follows the previd link which is available in all blocks, and tries to follow the interlink pointer again from there. Finally, the velvet prover shown in Algorithm 11 simply applies the velvet $\mathcal{C}\uparrow$ operator and includes the auxiliary connecting nodes within the final proof. No changes in the verifier are needed; note that in the case of infix proofs the index of the block is used by the verifier; if this information is not provided by the underlying blockchain headers, the index should be included in the interlink structure.

Algorithm 9 The modified constructInnerChain that allows for a velvet fork.

```
1: function constructInnerChain'(C, μ, b, realLink, blockById)
2:     B ← C[−1]
3:     aux ← [B]
4:     π ← [B]
5:     while B ≠ b do
6:         (B, aux') ← followUp(B, μ, realInterlink, blockById)
7:         aux.append(aux')
8:         π.append(B)
9:     end while
10:    return π, aux
11: end function
```

Algorithm 10 followUp produces the blocks to connect two superblocks in velvet forks.

```
1: function followUp(B, μ, realLink, blockById)
2:     aux ← [B]
3:     while B ≠ Gen do
4:         if B.interlink[μ] = realLink[id(B)][μ] then
5:             id ← B.interlink[μ]
6:         else                                          ▷ Invalid interlink
7:             id ← B.interlink[0]
8:         end if
9:         B ← blockById[id]
10:        aux ← aux ∪ {B}
11:        if level(B) = μ then
12:            return B, aux
13:        end if
14:    end while
15:    return B, aux
16: end function
```

Algorithm 11 The Prove algorithm for the NIPoPoW protocol, modified for a velvet fork.

```
1: function Prove'_{m,k}(C, realLink, blockById)
2:     size ← |C| − (k − 1)
3:     i ← |C[size].interlink|
4:     b ← C[0]                                          ▷ Genesis block
5:     Ĩ ← ∅
6:     for μ = |realLink[C[−k − 1]]| down to 0 do
7:         π, aux ← constructInnerChain'(C[: −k], μ, b, realLink, blockById)
8:         if |π| ≥ m then
9:             b ← π[−m]
10:        end if
11:        Ĩ ← Ĩ ∪ aux
12:    end for
13:    return Ĩ
14: end function
```

Velvet NIPoPoWs preserve security. Additionally, if a constant minority of miners has upgraded their nodes, then succinctness is also preserved as shown below.

Theorem 8. *Velvet non-interactive proofs-of-proof-of-work on honest chains by honest provers remain succinct as long as a constant percentage g of miners has upgraded, with overwhelming probability.*

Proof. From Theorem 6 we know that the proofs π contain only a $O(polylog(m))$ amount of blocks. For each of these blocks, the velvet client needs to include a followUp tail of blocks. Assume a percentage $0 < g \leq 1$ of miners have upgraded with NIPoPoW support. Then the question of whether each block in the honest chain is upgraded follows a Bernoulli distribution. If the velvet proof were to be larger than Δ times the soft fork proof in the number of blocks included, then this would require at least one of the followUp tails to include at least Δ sequential unupgraded blocks. But since the upgrade status of each block is independent, the probability of this occurring is g^{Δ}, which is negligible in Δ. □

The reason we were able to upgrade using a velvet fork was because the changes we made were helpful but verifiable by those looking at the chain. We would not have been

able to pull off this upgrade without modifications to the consensus layer in the sense that the interlink data structure could not have been maintained somewhere independently of the blockchain: It is critical that the proof-of-work commits to the interlink data structure. Interestingly, the interlink data structure does not need to be part of coinbase and can be produced and included in regular transactions by users (such as OP_RETURN transactions). Thus, the miners can be completely oblivious to it, while users and provers make use of the feature. Interested users regularly create transactions containing the most recent interlink pointers so that they are included in the next block. If the transaction makes it to the next block, it can be used by the prover who keeps track of these. Otherwise, if it becomes part of a subsequent block, in which case some of the pointers it contains are invalid, it can be ignored or only partially used.

8 Implementation & Parameters

8.1 Interlink optimizations

We now discuss the manner in which the interlink data structure is hashed for coinbase inclusion. A Merkle tree is used to hash the interlink data structure into a single hash [13]. NIPoPoWs form a chain of various levels which can omit blocks. For each block in the proof, only a single pointer needs to be presented to convince the Verifier. Near genesis, the pointers needed correspond to high levels; near the tip, the pointers to low levels. In the construction of the proof, the highest superchain with at least m blocks is included, and assume it is of level μ. The level $\mu-1$ superchain is fully included and has an expected number of $2m$ blocks. Since all μ-superblocks are also $(\mu-1)$-superblocks, they only need to be counted once, for $\mu-1$. Among the expected $2m$ $(\mu-1)$-superblocks, the last m will be supported by level $\mu-2$. As before, since $(\mu-1)$-superblocks are $(\mu-2)$-superblocks, in expectation only m $(\mu-1)$-superblocks are counted. The argument continues inductively, until $2m$ 0-blocks are included in expectation immediately before the χ suffix. This gives an estimation on the proof size: a total of $m(\log(|\mathcal{C}|) - \log(m))$ blocks in expectation, m at each of the $\mu-1$ levels and $2m$ 0-blocks.

Organizing interlink into a Merkle tree of $\log(|\mathcal{C}|)$ items, a proof-of-inclusion is provided in $\log\log(|\mathcal{C}|)$ space; 0-level pointers need not be included in it, but the genesis block does. The root of the tree can be proved to have been included in the block header in $\log(|\overline{x}|)$ using the standard Merkle tree of transactions. This makes the proof size require $\log(|\overline{x}|) + \log\log(|\mathcal{C}|)$ hashes per block for a total of $m(\log(|\mathcal{C}|) - \log(m))(\log(|\overline{x}|) + \log\log(|\mathcal{C}|))$ hashes. In addition, $m(\log(|\mathcal{C}|) - \log(m))$ headers and coinbase transactions are needed. Concretely, given that currently in bitcoin $|\mathcal{C}| = 464,185$ and $|\overline{x}| = 2000$, we have $\log(|\mathcal{C}|) = 18, \log\log(|\mathcal{C}|) = 5, \log(|\overline{x}|) = 11$. For the k-suffix, only k headers are needed. We set $k = 6$ and see that headers are 80 bytes and hashes 32 bytes. For the k-suffix as well as the $2m$ 0-blocks in π, neither coinbase data nor prev ids are needed, limiting header size to 48 bytes. The root and leaves of the pointers tree can be omitted from coinbase when transmitting the proof. In fact, no block ids need to be transmitted. From these observations, we estimate our scheme's proof sizes for various parameterizations of m in Table 1.

8.2 Concrete parameterization

To determine concrete values for security parameter m, we focus on a particular adversarial strategy and analyze its probability of success. This adversarial strategy is only one possible strategy; an advanced adversary can perform more sophisticated attacks. However, the fact that this attack is reasonable and possible to simulate allows us to extract specific values for m. The attack is an extension of the stochastic processes described in [1] and [26].

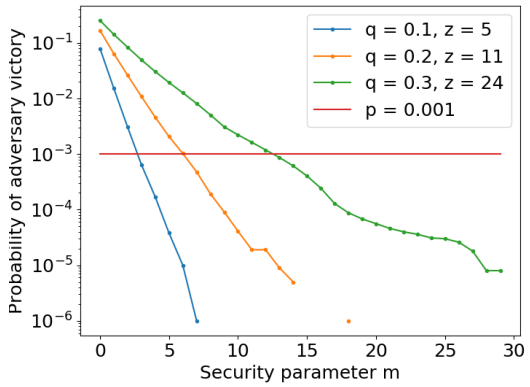
Table 1. Size of NIPoPoWs applied to Bitcoin today (≈450k blocks) for various values of m, setting $k = 6$.

m	NIPoPoW size	Blocks	Hashes
6	70 kB	108	1440
15	146 kB	231	2925
30	270 kB	426	5400
50	412 kB	656	8250
100	750 kB	1206	15000
127	952 kB	1530	19050

The experiment works as follows: m is fixed and some adversarial computational power percentage q of the total network computational power is chosen; k is chosen based on q according to Nakamoto [1]. The number of blocks y during which parallel mining will occur is also fixed. The experiment begins with the adversary and honest parties sharing a common blockchain which ends in block B. After B is mined, the adversary starts mining in secret and in parallel with the honest parties on her own private fork on top of B. She ignores the honest chain, so that the two chains remain disjoint after B. As soon as y blocks have been mined in total, the adversary attempts a double spend via a NiPoPoW by creating two conflicting transactions which are committed to an honest block and an adversarial block respectively on top of each current chain. Finally, the adversary mines k blocks on top of the double spending transaction within her private chain. After these k blocks have been mined, she publishes her private chain in an attempt to overcome the honest chain.

We measure the probability of success of this attack. We experiment with various values of m for $y = 100$, indicating 100 blocks of secret parallel mining. We make the assumption that honest party communication is perfect and immediate. We ran $1,000,000$ Monte Carlo executions [6] of the experiment for each value of m from 1 to 30. We ran the simulation for values of computational power percentage $q = 0.1$, $q = 0.2$ and $q = 0.3$. The results are plotted in Figure 7.

Fig. 7. Simulation results for a private mining attacker with k according to Nakamoto and parallel mining parameter $y = 100$. Probabilities in logarithmic scale. The horizontal line indicates the threshold probability of [1] is indicated by the horizontal line.



[6] The URL to the GitHub repository of this MIT-licensed experiment has been redacted for anonymity and will be provided in the proceedings version.

Based on this data, we conclude that $m = 5$ is sufficient to achieve a 0.001 probability of failure against an adversary with 10% mining power. To secure against an adversary with more than 30% mining power, a choice of $m = 15$ is needed.

9 Applications & Evaluation

9.1 Multi-blockchain wallets

An application of our technique is an efficient multi-cryptocoin client. Consider a merchant who wishes to accept payments in any cryptocoin, not just the popular ones. The naïve approach would be to install an SPV client for each known cryptocoin. This approach would entail downloading the header chain for each cryptocoin, and periodically syncing up by fetching any newly generated block headers. An alternative would be to use an online service supporting multiple currencies, but this introduces reliance on a third party (e.g. Jaxx and Coinomi rely on third party networks).

A NIPoPoW-based client would not download the entire header chain, but would intead only receive NiPoPoW proofs each time a payment is received. When a peer informs the client about a payment, they include a block index ℓ and NiPoPoW proof of transaction inclusion. The peer must then query *all* of their connected peers, requesting any better better proof for the same predicate. After waiting a short time period for a response, the client runs the `verify-infix` routine on all received proofs, and accepts the transaction if the output is *true*. Although initially such proofs must be relative genesis, the client may store the most recently-known (k-stable) blockhash for each cryptocoin, such that future payments can include NiPoPoW proofs relative to that. Thus for popluar cryptocoins, the NIPoPoW-based client downloads nearly every block header, like an ordinary SPV client; but for cryptocoins used infrequently, the NIPoPoW-based client can skip over most blocks.

9.2 Simulation

We simulated the resources savings resulting from the use of a NIPoPoW-based client. We model the arrival of payments in each cryptocoin as a Poisson process and assume that the market cap of a cryptocoin is a proxy for usage. Currently, a total of 731 cryptocurrencies are listed on coin market directories[7]. We narrow our focus to the 80 cryptocurrencies that have their own PoW blockchains (i.e., no PoS) with a market cap of over USD $100,000.

In Table 2 we show aggregate statistics about these 80 cryptocurrencies, grouped according to the their PoW puzzle. While the entire chain in Bitcoin only amounts to 40 MB, taken together, the 80 cryptocurrencies comprise 10 GB of proofs-of-work, and generate 10 MB more each day. In Table 3 we show the resulting bandwidth costs from simulating a period of 60 days with $m = 24, k = 6$, with varying rates of payments received. For the naïve SPV client, the total bandwidth cost is dominated by fetching the entire chain of headers, which the NIPoPoW client avoids. The marginal cost for naïve SPV depends on the number of blocks generated per day, as well as the transaction inclusion proofs associated with each payment. The NIPoPoW based client provides the most savings when the number of transactions per day is smallest, reducing the necessary bandwidth per day (excluding the initial sync up) by 90%.

[7] `https://coinmarketcap.com/`

Table 2. Cost of header chains for all active PoW-based cryptocoins (collected from `coinwarz.com`)

Hash	Coins	Size today	Growth rate
Scrypt	44	4.3 GB	5.5 MB / day
SHA-256	15	1.4 GB	937.0 kB / day
X11	5	581.1 MB	556.3 kB / day
Quark	3	647.9 MB	518.4 kB / day
CryptoNight	2	199.0 MB	115.2 kB / day
EtHash	2	588.6 MB	921.6 kB / day
Groestl	2	300.3 MB	184.2 kB / day
NeoScrypt	2	310.6 MB	153.6 kB / day
Others	5	266.2 MB	311.1 kB / day
Total	80	8.5 GB	9.2 MB / day

Table 3. Simulated bandwidth of multi-blockchain clients after two months (Averaged over 10 trials each)

Daily	Naive SPV	NIPoPoW	
tx	Total (Daily)	Total (Daily)	Savings
100	5.5 GB (5.5 MB)	31.7 MB (507 kB)	99% (91%)
500	5.5 GB (5.7 MB)	68.2 MB (1.1 MB)	99% (81%)
1000	5.5 GB (6.0 MB)	99.1 MB (1.6 MB)	98% (73%)
3000	5.6 GB (7.0 MB)	192 MB (3.1 MB)	97% (56%)

9.3 Certificate Transparency

In Catena [7] the Bitcoin blockchain is used as an equivocation-resistant public log in which to publish SSL certificate commitments. The client is based on the BitcoinJ library, and therefore requires downloading the entire chain. Catena could therefore immediately be improved using NIPoPoWs. The Catena authors anticipate needing to launch a dedicated Header Relay Network [7] to accommodate the extra bandwidth demands from new Catena clients. A variant based on NIPoPoWs could obviate this, since it eliminates the need to bootstrap new clients by transmitting the entire 40MB header chain. Second, the steady state cost of operating a Catena client depends on how frequently certificate digests are published. For example, one usage scenario cited by Catena [7] is Keybase, a service which publishes certificate digests every 6 hours. During a 6 hour period, Bitcoin would generate 6 kilobytes of headers, whereas a NIPoPoW proof covering the same range would require less than half this size. The savings would increase further if Catena were implemented using any PoW blockchain with more frequent blocks.

9.4 Sidechains

It is widely known that Bitcoin faces significant scaling hurdles [27], but upgrading Bitcoin is notoriously difficult. A widely anticipated solution is to treat the Bitcoin blockchain as a host for "sidechains," which are proof-of-work blockchains separate from Bitcoin, but that can be backed by Bitcoin deposits. Our efficient NIPoPoW construction solves an open problem posed by Back et al., and thus enables this vision.

The idea behind sidechains is to implement an SPV verifier for one blockchain (the "sidechain") as a smart contract within another blockchain (the "host chain"). An (inefficient) implementation of this idea, called BTCRelay [3], is already running on Ethereum. It comprises a smart contract that allows users to submit Bitcoin block headers, which it validates and stores. This allows Ethereum smart contracts to condition their behavior based on committed Bitcoin transactions. The downside is that every Bitcoin header must be stored, which is expensive due to the cost of Ethereum blockchain storage (7 cents per kB). If Bitcoin were upgraded with the interlink data structure, the BTCRelay functionality could be provided inexpensively.

The most anticipated use of sidechains is to implement a virtual asset on the sidechain backed by currency on the host chain (a two-way peg) [9]. This is accomplished by defining two new transaction types: "deposit" transactions lock coins on the host chain and create new assets on the side chain; "withdrawal" transactions do the opposite. Deposit transactions committed on the host chain are delivered to the sidechain using a proof; vice versa for withdrawal transactions. Whereas in the multi-blockchain wallet application the client queries each of its peers (one of which is assumed to be honest), in this setting peers must listen for event notifications when a "withdrawal" transaction is commited, and if they have a better NiPoPoW proof that invalidates this claim, they must submit it as a transaction within some time bound. The smart contract should collect a security deposit from the withdrawer that can be used to compensate peers for providing invalidation claims. Succinctness (and certificates of badness) are thus especially important for sidechains, since the size of the largest proof under normal conditions should determine what is an acceptable security deposit.

10 Future work

Our security proof is constructed in the Backbone [12] model with constant difficulty. Our construction can be extended to the variable difficulty setting [28]. We worked with a construction that allows a limited class of chain predicates to be proved. More predicates may be provable using a different NIPoPoW construction.

We have constructed Proofs-of-Proofs-of-Work, but an open question remains whether a Proof-of-Proof-of-Stake protocol is possible, interactively or non-interactively.

Our construction hints at a direction of sidechains, but no formalism exists that allows us to reason about whether NIPoPoW protocols are appropriate as sidechain solutions. A proper security definition for the desirable properties of sidechains would allow this evaluation to take place.

References

1. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
2. Nakamoto, S.: Bitcoin open source implementation of p2p currency. P2P Foundation **18** (2009)
3. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper **151** (2014)
4. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Annual International Cryptology Conference, Springer (1992) 139–147
5. Back, A., et al.: Hashcash-a denial of service counter-measure (2002)
6. Shin, L.: For first time bitcoin accounts for less than half of market cap of all cryptocurrencies. `https://www.forbes.com/sites/laurashin/2017/05/16/for-first-time-bitcoin-accounts-for-less-than-half-of-market-cap-of-all-cryptocurrencies/` (May 2017)
7. Tomescu, A., Devadas, S.: Catena: Efficient non-equivocation via bitcoin. In: IEEE Symp. on Security and Privacy. (2017)
8. Nolan, T.: Alt chains and atomic transfers. `bitcointalk.org` (May 2013)
9. Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., Wuille, P.: Enabling blockchain innovations with pegged sidechains. URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains (2014)
10. Poelstra, A.: On stake and consensus. (2015)
11. Friedenbach, M.: Compact spv proofs via block headercommitments. `https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg04318.html` (2014)
12. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer (2015) 281–310
13. Kiayias, A., Lamprou, N., Stouka, A.P.: Proofs of proofs of work with sublinear complexity. In: International Conference on Financial Cryptography and Data Security, Springer (2016) 61–78

14. Miller, A.: The high-value-hash highway, bitcoin forum post (2012)
15. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: Security and Privacy (SP), 2015 IEEE Symposium on, IEEE (2015) 104–121
16. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM conference on Computer and communications security, ACM (1993) 62–73
17. Apostolaki, M., Zohar, A., Vanbever, L.: Hijacking bitcoin: Routing attacks on cryptocurrencies. arXiv preprint arXiv:1605.07524 (2016)
18. Douceur, J.R.: The sybil attack. In: International Workshop on Peer-to-Peer Systems, Springer (2002) 251–260
19. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Conference on the Theory and Application of Cryptographic Techniques, Springer (1987) 369–378
20. Pass, R., Shi, E.: Fruitchains: A fair blockchain. In: Proceedings of the ACM Symposium on Principles of Distributed Computing, ACM (2017) 315–324
21. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer (2017) 643–673
22. Pugh, W.: Skip lists: a probabilistic alternative to balanced trees. Communications of the ACM **33**(6) (1990) 668–676
23. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: International conference on financial cryptography and data security, Springer (2014) 436–454
24. Buterin, V., et al.: A next-generation smart contract and decentralized application platform. white paper (2014)
25. Bahack, L.: Theoretical bitcoin attacks with less than half of the computational power (draft). arXiv preprint arXiv:1312.7013 (2013)
26. Rosenfeld, M.: Analysis of hashrate-based double spending. arXiv preprint arXiv:1402.2009 (2014)
27. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.G., et al.: On scaling decentralized blockchains. In: International Conference on Financial Cryptography and Data Security, Springer (2016) 106–125
28. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol with chains of variable difficulty. (2016)