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, IOHK

January 10, 2020

Abstract. Decentralized consensus protocols based on proof-of-work (PoW) mining require nodes to download data linear in the size of the blockchain even if they make use of Simplified Payment Verification (SPV). In this work, we put forth a new formalization of proof-of-work verification by introducing a primitive called Non-Interactive Proofs of Proof-of-Work (NIPoPoWs). We improve upon the previously known SPV NIPoPoW by proposing a novel NIPoPoW construction using superblocks, blocks that are much heavier than usual blocks, which capture the fact that proof-of-work took place without sending all of it. Unlike a traditional blockchain client which must verify the entire linearly-growing chain of PoWs, clients based on superblock NIPoPoWs require resources only logarithmic in the length of the chain, instead downloading a compressed form of the chain. Superblock NIPoPoWs are thus *succinct* proofs and, due to their non-interactivity, require only a single message between the prover and the verifier of the transaction. Our construction allows the creation of *superlight* clients which can synchronize with the network quickly even if they remain offline for large periods of time. Our scheme is provably secure in the Bitcoin Backbone model. From a theoretical point of view, we are the first to propose a cryptographic prover–verifier definition for decentralized consensus protocols and the first to give a construction which can synchronize non-interactively using only a logarithmically-sized message.

1 Introduction

Proof-of-work blockchain clients such as mobile wallets today are based on the Simplified Payment Verifications (SPV) protocol, which was described in the original Bitcoin paper [18], and allows them to sychronize with the network by downloading only block headers and not the entire blockchain with transactions. However, such initial synchronization still requires receiving all the block headers. In this work, we study the question of whether better protocols exist and in particular if downloading fewer block headers is sufficient to securely synchronize with the rest of the blockchain network. Our requirement is that the system remains decentralized and that useful facts about the blockchain (such as the Merkle root of current account balances in Ethereum [7,25]) can be deduced from the downloaded data.

Our contributions. We put forth a cryptographic security definition for Non-Interactive Proofs of Proof-of-Work protocols which describes what such a synchronization protocol must achieve (Section 2). We then construct a protocol which solves the problem and requires sending only a logarithmic number of blocks from the chain. We construct a protocol which can synchronize recent blocks, the *suffix proofs* protocol (Section 4). We analyze the security and succinctness of our protocol in Section 5. In the Appendix, we show a simple addition to the suffix proofs protocol which allows synchronizing any part of the blockchain that the client may be interested in, the *infix proofs* protocol (Section A). We give formal proofs of security and succinctness, provide concrete parameters for the implementation of our scheme, present applications beyond superlight clients including cross-chain applications, propose a mechanism with which our scheme can be deployed to existing cryptocurrencies without a fork, and propose a variable difficulty construction.

Previous work. The need for succinct clients was first identified by Nakamoto in his original paper [18]. Predicates pertaining to events occurring in the blockchain have been explored in the setting of sidechains [2]. It has also been implemented for simple classes of predicates such as atomic swaps [14, 19], which do not allow full synchronization. Non-succinct certificates about proof-of-*stake* blockchains have been proposed in [12], but their scheme is not applicable to proof-of-*work*. Superblocks were first described in the Bitcoin Forum [17] and later formalized [15] to describe their *Proofs of Proof-of-Work* which have limited applications due to interactivity, lack of security, and inability to prove facts buried deep within the blockchain. We improve upon their work with a security definition, an interactive construction, and an attack against their scheme which works with overwhelming probability.

2 Model and Definitions

Our model is based on the “backbone” model for proof-of-work cryptocurrencies [10], extended with SPV. Following their model, we assume *synchrony* (*partial synchrony* with *bounded delay* [20] is left for future work) and constant difficulty (we briefly discuss variable difficulty in Appendix H).

Backbone model. 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. Miners commit new transactions they receive from clients. (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 do not store the entire blockchain, but instead 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, for example whether a particular payment has been finalized.

Our main challenge is to design a protocol so that clients can sieve through the responses they receive from the network and reach a conclusion that should never disagree with the conclusion of a full node who is faced with the same objective and infers it from its local blockchain state.

We model proof-of-work discovery attempts by using a random oracle [4]. 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 per round, 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 a *uniquely successful round.* We assume there is an honest majority, i.e., that $t/n < 0.5$ with a constant minimum gap [10]. We further assume the network is adversarial, but there is no eclipsing attacks [13]. More specifically, we allow the adversary to reorder messages transmitted at a particular round, to inject new messages thereby capturing Sybil attacks [8], 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 [16] root of the transactions confirmed in this block; and *interlink* [15], 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 prover and verifier model. In our protocol, the nodes include a *proof* along with their responses to clients. We need to assume that clients are able to connect to at least one correctly functioning node (i.e., that they cannot be eclipsed from the network [1,13]). Each client makes the same request to every node, and by verifying the proofs the client identifies the correct response. Henceforth we will call clients *verifiers* and nodes *provers.*

The prover-verifier interaction is parameterized by a predicate (e.g. "the transaction tx is committed in the blockchain"). 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 [10], 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 (having only Genesis), receives a pair of proofs (π_A, π_B) from both an honest party and the adversary and returns its decision $d \in \{T, F\}$ 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.

Notation. 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 [21]. 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. The *id* function returns the id of a block given its data, i.e., $\mathsf{id} = H(ctr, G(x, \mathsf{interlink}))$.

2.1 Provable chain predicates

Our aim is to prove statements about the blockchain, such as "The transaction tx is included in the current blockchain" without transmitting all block headers. We consider a general class of predicates that take on values *true* or *false*. Since a Bitcoin-like blockchain can experience delays and intermittent forks, not all honest parties will be in exact agreement about the entire chain. However, when all honest parties are in agreement about the truth value of the predicate, we require that the verifier also arrives at the same truth value.

To aid the construction of our proofs, we focus on predicates that are *monotonic*; they start with the value *false* and, as the blockchain grows, can change their value to *true* but not back.

Definition 1. *(Monotonicity) A chain predicate $Q(\mathcal{C})$ is* monotonic *if for all chains $\mathcal{C}$ and for all blocks B we have that $Q(\mathcal{C}) \Rightarrow Q(\mathcal{C}B)$.*

Additionally, we require that our predicates only depend on the *stable* portion of the blockchain, blocks that are buried under k subsequent blocks. This ensures that the value of the predicate will not change due to a blockchain reorganization.

Definition 2. *(Stability) Parameterized by $k \in \mathbb{N}$, a chain predicate Q is k-*stable *if its value only depends on the prefix $\mathcal{C}[: -k]$.*

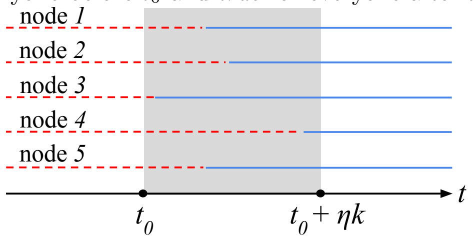
2.2 Desired properties

We now define two desired properties of a non-interactive blockchain proof protocol, *succinctness* and *security.*

Definition 3. *(Security) A* blockchain proof protocol (P, V) *about a predicate Q is* secure *if for all environments and for all PPT adversaries $\mathcal{A}$ and for all rounds $r \geq \eta k$, if V receives a set of proofs $\mathcal{P}$ at the beginning of round r, at least one of which has been generated by the honest prover P, then the output of V at the end of round r has the following constraints:*

- *If the output of V is* false*, then the evaluation of $Q(\mathcal{C})$ for* all *honest parties must be* false *at the end of round $r - \eta k$.*
- *If the output of V is* true*, then the evaluation of $Q(\mathcal{C})$ for* all *honest parties must be* true *at the end of round $r + \eta k$.*

Fig. 1. The truth value of a fixed predicate Q about the blockchain, as seen from the point of view of 5 honest nodes, drawn on the vertical axis, over time, drawn as the horizontal axis. The truth value evolves over time starting as *false* at the beginning, indicated by a dashed red line. At some point in time t_0, the predicate is ready to be evaluated as *true*, indicated by the solid blue line. The various honest nodes each realize this independently over a period of ηk duration, shaded in gray. The predicate remains *false* for everyone before t_0 and *true* for everyone after $t_0 + \eta k$.



Some explanation is needed for the rationale of the above definition. The parameter η is borrowed from the Backbone [10] work and indicates the rate at which new blocks are produced, i.e., the number of rounds needed on average to produce a block. If the scheme is secure, this means that the output of the verifier should match the output of a *potential honest full node*. However, in various executions, not all potential honest full node behaviors will be instantiated. Therefore, we require that, if the output of the proof verifier is *true* then, consistently with honest behavior, all other honest full nodes will converge to the value *true*. Conversely, if the output of the proof verifier is *false* then, consistently with honest behavior, all honest full nodes must have indicated *false* sufficiently long in the past. The period ηk is the period needed for obtaining sufficient confirmations (k) in a blockchain system. A predicate's value has the potential of being *true* as seen by an honest party starting at time t_0. Before time t_0, all honest parties agree that the predicate is *false*. It takes ηk time for all parties to agree that the predicate is *true*, which is certain after time $t_0 + \eta k$. The adversary may be able to convince the verifier that the predicate has any value during the period from t_0 to $t_0 + \eta k$. However, our security definition mandates that before time t_0 the verifier will necessarily output *false* and after time $t_0 + \eta k$ the verifier will necessarily output *true*.

Definition 4. *(Succinctness) A* blockchain proof protocol (P, V) *about a predicate Q is* succinct *if for all PPT provers $\mathcal{A}$, any proof π produced by $\mathcal{A}$ at some round r, the verifier V only reads a $O(polylog(r))$-sized portion of π.*

It is easy to construct a *secure but not succinct* protocol for any computable predicate Q: The prover provides the entire chain $\mathcal{C}$ as a proof and the verifier simply selects the longest chain: by the *common-prefix property* of the backbone protocol (c.f. [10]), this is consistent with the view of every honest party (as long as Q depends only on a *prefix* of the chain, as we explain in more detail shortly). In fact this is how widely-used cryptocurrency clients (including SPV clients) operate today.

It is also easy to build *succinct but insecure* clients: The prover simply sends the predicate value directly. This is roughly what hosted wallets do [5].

The challenge we will solve is to provide a non-interactive protocol that at the same time achieves security and succinctness over a large class of useful predicates. We call this primitive a NIPoPoWs. Our particular instantiation for NIPoPoWs is a *superblock-based NIPoPoW construction.*

3 Consensus layer support

3.1 The interlink pointers data structure

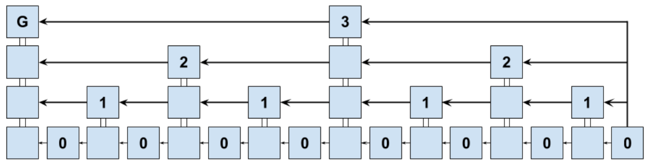
In order to construct our protocol, we rely the *interlink data structure* [15]. This is an additional hash-based data structure that is proposed to be included in the header of each block. The interlink data structure is a skip-list [22] that makes it efficient for a verifier to process a sparse subset of the blockchain, rather than only consecutive blocks.

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[4]. 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. The level of a block is given as $\mu = \lfloor \log(T) - \log(\mathsf{id}(\mathsf{B})) \rfloor$ and denoted $\mathit{level}(B)$. By convention, for Gen we set $id = 0$ and $\mu = \infty$.

Observe that in a blockchain protocol execution it is expected $1/2$ 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 $\Theta(\log(\mathcal{C}))$ [15]. Figure 2 illustrates the blockchain superblocks starting from level 0 and going up to level 3 in case these blocks are distributed exactly according to expectation. Here, each level contains half the blocks of the level below.

We wish to connect the blocks at each level with a *previous block* pointer pointing to the most recent block of the same level. These pointers must be included in the data of the block so that proof-of-work commits to them. As the level of a block cannot be prediced before its proof-of-work is calculated, we extend the *previous block id* structure of classical blockchains to be a vector, the *interlink vector*. The interlink vector points to the most recent preceding block of every level μ. Genesis is of infinite level and hence a pointer to it is included in every block. The number of pointers that need to be included per block is in expectation $\log(|\mathcal{C}|)$.

Fig. 2. The hierarchical blockchain. Higher levels have achieved a lower target (higher difficulty) during mining. All blocks are connected to the genesis block G.



The algorithm for this construction is shown in Algorithm 1 and is borrowed from [15]. The interlink data structure turns the blockchain into a skiplist-like [22] data structure.

[4] We discuss how this assumption can be relaxed in the Appendix.

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 $\leftarrow B'$.interlink
3:     for $\mu = 0$ to $level(B')$ do
4:         interlink[$\mu$] $\leftarrow$ id($B'$)
5:     end for
6:     return interlink
7: end function
```

Traversing the blockchain. As we have now extended blocks to contain multiple pointers to previous blocks, if certain blocks are omitted from the middle of a chain we will obtain a subchain, as long as the *blockchain property* is maintained (i.e., that each block must contain an interlink pointer to its previous block in the sequence).

Blockchains are sequences, but it is more convenient to use set notation for some operations. Specifically, $B \in \mathcal{C}$ and $\emptyset$ have the obvious meaning. $\mathcal{C}_1 \subseteq \mathcal{C}_2$ means that all blocks in $\mathcal{C}_1$ exist in $\mathcal{C}_2$, perhaps with additional blocks intertwined. $\mathcal{C}_1 \cup \mathcal{C}_2$ 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, as pointers may be missing). 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: $(\mathcal{C}{\uparrow^\mu}){\uparrow^{\mu+i}} = \mathcal{C}{\uparrow^{\mu+i}}$. Given a set of consecutive rounds $S = \{r, r+1, \cdots, r+j\} \subseteq \mathbb{N}$, we define $\mathcal{C}^S = \{B \in \mathcal{C} : B \text{ was generated during } S\}$.

4 Non-interactive blockchain *suffix* proofs

In this section, we introduce our non-interactive suffix proofs. With foresight, we caution the reader that the non-interactive construction we present in this section is *insecure*. A small patch will later allow us to modify our construction to achieve security.

We allow provers to prove general predicates Q about the chain $\mathcal{C}$. Among the predicates which are stable, in this section, we will limit ourselves to *suffix sensitive* predicates. We extend the protocol to support more flexible predicates (such as transaction inclusion, as needed for our applications) which are not limited to the suffix in Section A.

Definition 5 (Suffix sensitivity). *A chain predicate Q is called k-*suffix sensitive *if its value can be efficienty computed given the last k blocks of the chain.*

Example. In general our applications will require predicates that are not suffix-sensitive. However, as an example, consider the predicate "an Ethereum contract at address C has been initialized with code h at least k blocks ago" where h does not invoke the `selfdestruct` opcode. This can be implemented in a suffix-sensitive way because, in Ethereum, each block includes a Merkle Trie over all of the contract codes [7, 25], which cannot be changed after initialization. This predicate is thus also monotonic and k-stable. Any predicate which is both *suffix-sensitive* and *k-stable* must solely depend on data at block $\mathcal{C}[-k]$.

4.1 Construction

We next 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 $\mathcal{P}$ at least one of which will be honest. 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.

Algorithm 2 The Verify algorithm for the NIPoPoW protocol

```
1: function $\mathsf{Verify}^{Q}_{m,k}(\mathcal{P})$
2:     $\tilde{\pi} \leftarrow (\text{Gen})$                              ▷ Trivial anchored blockchain
3:     for $(\pi, \chi) \in \mathcal{P}$ do                           ▷ Examine each proof $(\pi, \chi)$ in $\mathcal{P}$
4:         if $\mathsf{validChain}(\pi\chi) \land |\chi| = k \land \pi \geq_m \tilde{\pi}$ then
5:             $\tilde{\pi} \leftarrow \pi$
6:             $\tilde{\chi} \leftarrow \chi$                                ▷ Update current best
7:         end if
8:     end for
9:     return $\tilde{Q}(\tilde{\chi})$
10: end function
```

The verifier compares the proof prefixes 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.

At each loop iteration, the verifier compares the next candidate proof prefix π against the currently best known proof prefix $\tilde{\pi}$ by calling $\pi \geq_m \tilde{\pi}$. If the candidate prefix is better than the currently best known proof prefix, then the currently known best prefix is updated by setting $\tilde{\pi} \leftarrow \pi$. When the best known prefix is updated, the suffix $\tilde{\chi}$ associated with the best known prefix is also updated to match the suffix χ of the candidate proof by setting $\tilde{\chi} \leftarrow \chi$. 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. The best known proof prefix is initially set to (Gen), the trivial anchored chain containing only the genesis block. Any well-formed proof compares favourably against the trivial chain.

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. Note that, because the full chain is not available to the verifier, the verifier here must evaluate the predicate on the suffix. Because the predicate is suffix-sensitive, it is possible to do so. As a technical detail, we denote $\tilde{Q}$ the predicate which accepts only a k-suffix of a blockchain and outputs the same value that Q would have output if it had been evaluated on a chain with that suffix.

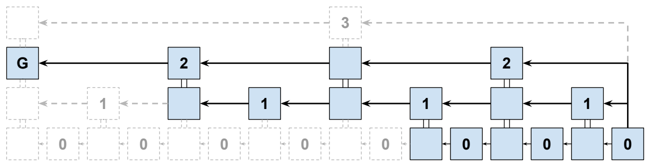
Algorithm 3 The Prove algorithm for the NIPoPoW protocol

```
1: function $\mathsf{Prove}_{m,k}(\mathcal{C})$
2:     $B \leftarrow \mathcal{C}[0]$                                  ▷ Genesis
3:     for $\mu = |\mathcal{C}[-k-1].\mathsf{interlink}|$ down to 0 do
4:         $\alpha \leftarrow \mathcal{C}[:-k]\{B:\}\uparrow^{\mu}$
5:         $\pi \leftarrow \pi \cup \alpha$
6:         if $m < |\alpha|$ then
7:             $B \leftarrow \alpha[-m]$
8:         end if
9:     end for
10:    $\chi \leftarrow \mathcal{C}[-k:]$
11:    return $\pi\chi$
12: end function
```

The concrete prover. The NIPoPoW prover construction is shown in Algorithm 3. The honest prover is supplied with an honestly adopted chain $\mathcal{C}$ and security parameters m, k 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 from the highest level $|\mathcal{C}[-k].\mathsf{interlink}|$ down to 0. 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 from this suffix until the end of the blockchain is also included. All the μ-superblocks which are within this range of m blocks will also be $(\mu-1)$-superblocks and so we do not want to keep them in the proof twice (we use the union set notation to indicate this). Each underlying superchain will have $2m$ blocks in expectation and always at least m blocks. This is repeated until level $\mu = 0$ is reached. 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 3 contains an example proof constructed for parameters $m = k = 3$. The top superchain level which contains at least m blocks is level $\mu = 2$. For the m-sized suffix of that level, 6 blocks of superblock level 1 are included to span the same range ($2m$ blocks at this level). For the last 3 blocks of the 1-superchain, blocks of level 0 spanning the same range are included (again $2m$ blocks at this level). Note that the superchain at a lower levels may reach closer to the end of the blockchain than a higher level. Level 3 was not used, as it does not yet have a sufficient number of blocks.

Fig. 3. NIPoPoW *prefix* π for $m = 3$. It includes the Genesis block G, three 2-superblocks, six 1-superblocks, and six 0-blocks.



Algorithm 4 The algorithm implementation for the $\geq_m$ 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.

```
1: function best-arg_m(π, b)
2:     $M \leftarrow \{\mu : |\pi{\uparrow}^{\mu}\ \{b :\}| \geq m\} \cup \{0\}$        ▷ Valid levels
3:     return $\max_{\mu \in M}\{2^{\mu} \cdot |\pi{\uparrow}^{\mu}\ \{b :\}|\}$        ▷ Score for level
4: end function
5: operator $\pi_A \geq_m \pi_B$
6:     $b \leftarrow (\pi_A \cap \pi_B)[-1]$        ▷ LCA
7:     return best-arg_m($\pi_A, b$) $\geq$ best-arg_m($\pi_B, b$)
8: end operator
```

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. An *argument* is a subchain of a proof provided by a prover such that its blocks are after the LCA block b and they are all at the same level μ. 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 indices μ 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 making the $\geq_m$ operator favour the adversary $\mathcal{A}$.

Looking ahead, the core of the security argument will be that, given a block b, it will be difficult for a mining minority adversary to produce blocks descending from b faster than the honest party. This holds for blocks of any level.

5 Analysis

We now give a sketch indicating why our construction is secure. The fully formal security proof, together with a detail in the construction which ensures statistical *goodness* and is necessary for withstanding full 1/2 adversaries, appears in the appendix.

Theorem 1 (Security). *Assuming honest majority, the Non-interactive Proofs of Proof-of-Work construction for computable k-stable monotonic suffix-sensitive predicates is secure with overwhelming probability in κ.*

Proof (Sketch). 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 be 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. $\square$

We formalize the above proof sketch in the Appendix.

Lastly, the following theorem illustrates that our proofs are succinct. Intuitively, the number of levels exchanged is logarithmic in the length of the chain, and the number of blocks in each level is constant. The formal proofs are included in the Appendix.

Theorem 2 (Optimistic succinctness). *In an optimistic execution, Non-Interactive Proofs of Proof-of-Work produced by honest provers are succinct with the number of blocks bounded by* $4m\log(|\mathcal{C}|)$*, with overwhelming probability in* m*.*

Appendix

Our Appendix is structured as follows. In Section A we discuss how to extend our scheme to prove facts buried deep within the blockchain. This ability can be easily added to our protocol, but is important in order to be able to show that a transaction took place in the past. In Section B we measure the concrete probability of success of our scheme in order to provide concrete parameters for implementors. In Section C we give further applications of NIPoPoWs beyond the context of a simple client, in particular a *multiblockchain wallet* and a *cross-chain ICO*. We also implement our scheme and perform measurements to give concrete performance numbers. Section D gives lemmas and proofs about the statistical properties of chains, which are useful for further results. Section E contains an attack against these statistical properties, which mandates that a full construction needs to check for them. Section F gives a formal proof of our security claims through a computational reduction. Section I includes the formal proof that our construction is succinct. In Section G, we illustrate gradual deployment paths. One of our techniques allows adoption of our scheme without requiring miner consensus. We term this technique a *velvet fork* in contrast to the classical *soft* and *hard forks* which require approval by a majority of miners. This technique is a novel contribution and may be of independent interest for other blockchain protocols. We conclude with Section H which gives an intuition about creating a construction for variable difficulty NIPoPoWs by modifying the construction presented in this paper.

A Non-interactive blockchain *infix* proofs

In the main body we have seen how to construct proofs for suffix predicates. As mentioned, the main purpose of that construction is to serve as a stepping stone for the construction of this section that presents a more useful class of proofs. This class of proofs allows 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, the *witness*, which may not necessarily form a chain. This allows proving useful statements such as, for example, whether a transaction is confirmed. We next formally define the class of predicates that will be of interest.

Definition 6 (Infix sensitivity). *A chain predicate* $Q_{d,k}$ *is* infix sensitive *if it can be written in the form*

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

where D *is an arbitrary efficiently computable predicate such that, for any block sets* $\mathcal{C}_1 \subseteq \mathcal{C}_2$ *we have that* $D(\mathcal{C}_1) \rightarrow D(\mathcal{C}_2)$*.*

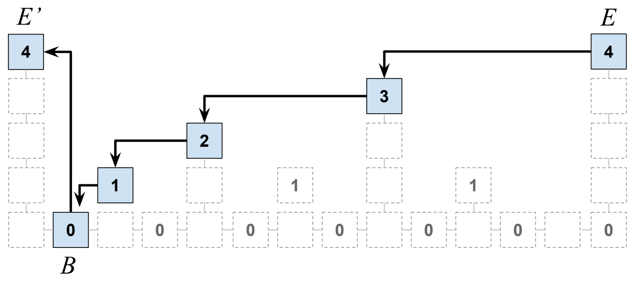
Note that $\mathcal{C}'$ is a blockset and may not necessarily be a blockchain. Furthermore, observe that for all blocksets $\mathcal{C}' \subseteq \mathcal{C}$ we have that $Q(\mathcal{C}') \rightarrow Q(\mathcal{C})$. This will allow us to later argue that adding more blocks to a blockchain cannot invalidate its witness.

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.

Example. We next show how to express the predicate that asks whether a certain transaction with id *txid* has been confirmed as an infix sensitive predicate. We define the predicate D^{txid} that receives a single block and tests whether a transaction with id *txid* is included. The predicate $Q^{txid}_{1,k}$ is defined as in Definition 6 using the predicate D^{txid} and the parameter k which in this case determines the desired stability of the assertion that *txid* is included (e.g., $k = 6$). Q alone proves that a particular block is included in the blockchain. Some auxiliary data is supplied by the prover to aid the provability of transaction inclusion: the Merkle Tree proof-of-inclusion path to the transactions Merkle Tree root, similar to an SPV proof. This data is logarithmic in the number of transactions in the block and, hence, constant with respect to blockchain size. In case of a vendor awaiting transaction confirmation to ship a product, the proof that a certain transaction paid into a designated address for the particular order is sufficient. In this scheme it is impossible to determine whether the money has subsequently been spent in a future block, and so must only be used by the vendor holding the respective secret keys.

In the above example, note that if the verifier outputs *false*, this behavior will generally be inconclusive in the sense that the verifier could be outputting *false* either because the payment has not yet been confirmed or because the payment was never made.

Fig. 4. 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 through 3 are produced by followDown to get to the block of level 0 which is part of $\mathcal{C}'$.



Algorithm 5 The Prove algorithm for infix proofs

```
 1: function ProveInfix$_{m,k}$($\mathcal{C}$, $\mathcal{C}'$, height)
 2:     $aux \leftarrow \emptyset$
 3:     $(\pi, \chi) \leftarrow$ Prove$_{m,k}(\mathcal{C})$                ▷ Start with a suffix proof
 4:     for $B \in \mathcal{C}'$ do
 5:         for $E \in \pi$ do
 6:             if height$[E] \geq$ height$[B]$ then
 7:                 $aux \leftarrow aux \cup$ followDown$(E, B,$ height$)$
 8:                 break
 9:             end if
10:         end for
11:     end for
12:     return $(aux \cup \pi, \chi)$
13: end function
```

A.1 Construction

The construction of these proofs is shown in Algorithm 5. 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 and 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 subset $\mathcal{C}'$, 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 B and E', otherwise it would have been included in π. Therefore, B already contains a pointer to E' in its interlink, completing the chain.

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 E, it tries to follow a pointer to as far as possible. If following the pointer surpasses B, then the procedure at this level 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 is reached, which concludes the algorithm.

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

```
function followDown($E$, $B$, height)
    $aux \leftarrow \emptyset$; $\mu \leftarrow level(E)$
    while $E \neq B$ do
        $B' \leftarrow$ blockById[$E$.interlink[$\mu$]]
        if height[$B'$] < height[$B$] then
            $\mu \leftarrow \mu - 1$
        else
            $aux \leftarrow aux \cup \{E\}$
            $E \leftarrow B'$
        end if
    end while
    return $aux$
end function
```

An example of the output of followDown is shown in Figure 4. 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 Algorithm 7.

The algorithm works by calling the suffix verifier. It also maintains a blockDAG collecting blocks from all proofs (it is a DAG because *interlink* can be adversarially defined in adversarially mined blocks). This DAG is maintained in the blockById hashmap. Using it, ancestors uses simple graph search to extract the set of ancestor blocks of a block. 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. In particular, such an adversary would work by including a complete suffix proof, but "forgetting" to include the blocks generated by followDown for the infix proof pertaining to blocks in $\mathcal{C}'$.

Algorithm 7 The verify algorithm for the NIPoPoW infix protocol

```
function ancestors($B$, blockById)
    if $B = \text{Gen}$ then
        return $\{B\}$
    end if
    $\mathcal{C} \leftarrow \emptyset$
    for id $\in B$.interlink do
        if id $\in$ blockById then
            $B' \leftarrow$ blockById[id]
            $\mathcal{C} \leftarrow \mathcal{C} \cup$ ancestors($B'$, blockById)        ▷ Collect into DAG
        end if
    end for
    return $\mathcal{C} \cup \{B\}$
end function
function verify-infx$^{D}_{\ell,m,k}(\mathcal{P})$
    blockById $\leftarrow \emptyset$
    for $(\pi, \chi) \in \mathcal{P}$ do
        for $B \in \pi$ do
            blockById[id($B$)] $\leftarrow B$
        end for
    end for
    $\tilde{\pi} \leftarrow$ best $\pi \in \mathcal{P}$ according to suffix verifier
    return $D$(ancestors($\tilde{\pi}[-1]$, blockById))
end function
```

B Implementation & Parameters

We now discuss the size of NIPoPoW proofs and evaluate concrete parameters. Organizing the interlink data structure as a Merkle tree of $\log(|\mathcal{C}|)$ items, a proof-of-inclusion is provided in $\log\log(|\mathcal{C}|)$ space in expectation; the proof need not include 0-level pointers, but must include the genesis block. Transaction inclusion [5] can be proved in the block header in $\log(|\overline{x}|)$ using the standard Merkle tree of transactions, where $\overline{x}$ denotes the vector of all transactions included in the particular block. This makes the proof size require $\log(|\overline{x}|) + \log\log(|\mathcal{C}|)$ hashes per block for a total of $(2m(\log|\mathcal{C}| - \log m) + m)(\log|\overline{x}| + \log\log|\mathcal{C}|)$ hashes. In addition, $m(\log(|\mathcal{C}|) - \log(m))$ headers and coinbase transactions are needed. As an example, given that currently in bitcoin $|\mathcal{C}| = 464{,}185$ and $|\overline{x}| = 2000$, we

[5] This additional data is needed if a *soft* or *hard fork* is to be avoided. For more information about gradual deployment paths, consult the relevant section in the appendix.

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.

Concrete parameterization. To determine concrete values for security parameter m, we focus on a particular adversarial strategy and analyze its probability of success. The attack is an extension of the stochastic processes described in [18] and [23].

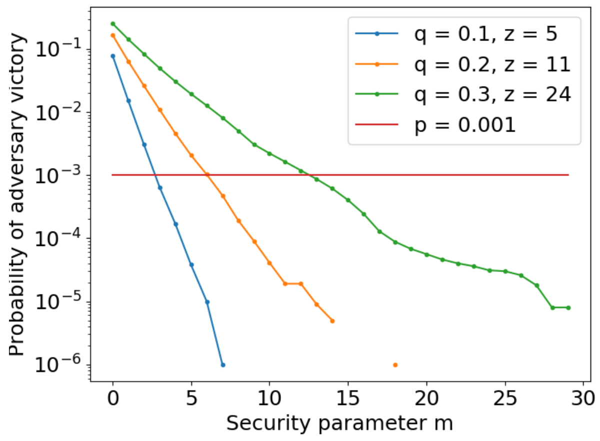
The experiment works as follows: m is fixed and some adversarial computational power percentage q of the total network computational power is chosen; k is chosen based on q according to Nakamoto [18]. The number of blocks y during which parallel mining will occur is also fixed. The experiment begins 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.

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

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 5. Based on these 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.

Fig. 5. 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 [18].



C Evaluation & Applications

In this section we evaluate the cost of NIPoPoWs when used in realistic blockchain applications. First we simulated the resources savings resulting from the use of a NIPoPoW-based client compared to ordinary SPV. We model the arrival of payments in each cryptocurrency as a Poisson process and assume that the market cap of a cryptocurrency 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 with a market cap of over USD $100,000.

[6] Our experiment can be reproduced by running our code available under an open source MIT license at `https://github.com/dionyziz/popow/tree/master/experiment`

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

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%.

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)

tx/	**Naive SPV**		**NIPoPoW**		
day	**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%)

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 coin. This approach would entail downloading the header chain for each coin, 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 instead 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 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 to genesis, the client may store the most recently-known (k-stable) blockhash for each coin such that future payments can include NIPoPoW proofs relative to that. Thus for popular cryptocurrencies, the NIPoPoW-based client downloads nearly every block header, like an ordinary SPV client; but for coins used infrequently, the NIPoPoW-based client can skip over most blocks.

Cross-chain ICOs. As an example use-case of our construction, we present the case of an ICO in which tokens are distributed in one blockchain, but funds are raised in another. It works as follows: There are two designated blockchains, the *source* and the *destination blockchain*. The source is the blockchain where the fund-raising will take place, while the destination is the blockchain where the newly issued tokens will be distributed and subsequently exchanged. The destination blockchain must be smart-contract-enabled in order to allow for the distribution of ERC-20-style [24] tokens. In addition, the smart contracts on the destination blockchain must allow for programming the verification of a NIPoPoW proof by including, for example, the appropriate hash functions. The source blockchain must be NIPoPoW-enabled. This setup allows the creation of NIPoPoWs *about* the source blockchain which will be included in the destination blockchain. For example, a source blockchain can be Litecoin and a destination blockchain Ethereum.

In order to run the ICO, the fund-raising entity first creates a designated account in the source blockchain in which funds will be deposited. It then creates the ERC-20-style smart contract in the destination blockchain. When someone wishes to participate in the ICO, they transfer funds into the designated account on the source blockchain. Once they have made the transfer and it becomes confirmed, the payer generates a NIPoPoW about the transaction paying into the designated account. This NIPoPoW is then sent as a parameter to a method call on the ICO smart contract on the destination blockchain. The method call stores the proof and waits for a certain period of time for possible contestations, which can be accepted and compared using the $\leq_m$ mechanism previously described. If no contesting proof is presented within the contestation period, the prover receives their respective ICO tokens on the target blockchain. In order for only the rightful owner to be able to receive the tokens, they are required to sign the destination address on the destination blockchain using the private key corresponding to their source account used to make the payment within the source blockchain.

We implemented the NIPoPoW verifier algorithm as a Solidity smart contract[8]. The contract consists of two functions. The `submit_nipopow` function is used by the provers to provide their proof vectors. Instead of passing the block headers of the proof, the provers pass the hashes of the block headers and the hashes of the interlink vector. The reason is that the full data of the block header (especially the Merkle tree root) is only useful for the blocks of interest. Thus, we reduce the amount of data needed for the proof by a factor of 2. The rest of the parameters are used in the inclusion proof of the block. After confirming the validity of the proof, the `compare_proofs` function is called between the current and the best proof. If the current proof is better then it is assigned to the best proof in the contract's storage. The gas costs are summarized in Table 4. The $USD column represents the current price of this much gas on Ethereum.

Table 4. Verifier contract functions

Function	Data	Gas cost	$USD
`compare_proofs`	~8Kb	~5M	$4
`submit_nipopow`	~65Kb	~40M	$32

D Superchain Quality Distributions

In order to argue formally about the security properties of blockchains that are equipped with the interlink data structure we introduce the new concept of *superchain quality*, which generalizes the chain quality property from the backbone model [10]. Superchain quality is a new contribution in this paper and is essential for identifying and overcoming the attack on PoPoW.

[8] The source code of the smart contract is available under an open source MIT license at `https://github.com/dionyziz/popow/blob/master/experiment/contractNipopow.sol`

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.

Intuitively, these definitions tell us that μ-superblocks occur approximately once every 2^μ blocks. Below, we make this notion more formal.

Definition 7 (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 $\textsf{local-good}_\delta(\mathcal{C}', \mathcal{C}, \mu)$, if $|\mathcal{C}'| > (1-\delta)2^{-\mu}|\mathcal{C}|$.*

Definition 8 (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 $\textsf{local-good}_\delta(C{\uparrow^\mu}[-m' :], C{\uparrow^\mu}[-m' :]{\downarrow}, \mu)$. That is, all sufficiently large suffixes are locally good.*

Definition 9 (Multilevel quality). *A μ-superchain $\mathcal{C}'$ is said to have* multilevel quality*, written* $\text{multi-good}_{\delta,k_1}(\mathcal{C}, \mathcal{C}', \mu)$ *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 10 (Good superchain). *A μ-superchain $\mathcal{C}'$ is said to be* good*, written* $\text{good}_{\delta,k_1}(\mathcal{C}, \mathcal{C}', \mu)$*, 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. We formalize this in the following theorems.

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, $\textsf{local-good}_\delta(\mathcal{C}', \mathcal{C}, \mu)$, 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 any level μ, and any $0 < \delta < 0.5$, a chain $\mathcal{C}$ containing only honestly-generated blocks adopted by an honest party in an execution with random network 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. □

E Proof of attack on PoPoW

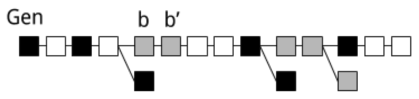
We now show that, if the statistical properties of blockchains are not respected in some execution, our construction presented in the main paper is insecure by illustrating an explicit attack against our scheme. During the exposition of this attack, a simple patch for our construction, which will also lead to a correct generic security proof, will become clear.

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 assuming an attacker of sufficiently high hashing power (but still below 50%).

E.1 Attacking chain superquality

We construct an adversary $\mathcal{A}$ that breaks the superchain quality 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 $\mathit{level}(\mathcal{C}_B[-1]) < \mu$, then $\mathcal{A}$ remains idle. However, if $\mathit{level}(\mathcal{C}_B[-1]) \geq \mu$, 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 6.

Fig. 6. Superquality attack on prior work (PoPoW) [15]. The adversary performs a selfish-mining [9] attack (gray blocks) whenever any honest parties have recently mined a rare μ-superblock (black). The attack reduces the honest chain's superquality, while the attacker's private chain is unaffected.



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.

E.2 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 [9]. 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. We just remark here that for appropriate choice of system parameters, the attack can be made to succeed with overwhelming probability.

We now calculate the exact probability of success of the attack. 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 [9].

[9] The attack could be further optimized, but we simplify it for exposition.

The adversary monitors the network for superblocks. Whenever an honest party diffuses an honestly-generated μ-superblock, at the end of a given round r_1, the adversary starts devoting their mining power to block suppression starting from the next round.

The 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 3 (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 the constructions of Section 4 and Section A 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.

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[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[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. $\square$

E.3 Interactive Proofs of Proof-of-Work

Our attack also applies against the protocol described in [15].

In [15], 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 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 is chosen to ensure that the probability of the attacker producing a long superchain is negligible.

It is worth isolating the mistake in their security proof. 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 correctly 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.

F Formal security treatment

Based on the attack explored above, it is now easy to see that our construction can be patched in a straightforward manner to achieve security. In particular, since the manner in which the adversary was able to subvert the prover was by the violation of *goodness*, we can mandate that the prover only tries to use succinct proofs to prove claims about chains that are *good at every level*. In case goodness is violated, the prover simply falls back to providing the whole chain. This allows us to argue that the construction is secure by distinguishing two cases. In case goodness is violated, the honest prover will fall back to providing the whole chain, in which case security will be reduced to the security of the standard blockchain protocol choosing the longest 0-chain. In case goodness is not violated, we will argue that the adversary is unable to win in these comparisons.

The previous construction was designed to prevent Bahack-style attacks [3], where the adversary constructs "lucky" high-difficulty superblocks without filling in the underlying proof-of-work in the lower levels. We now patch our protocol which, while retaining this high level approach, adds a defence against the double-spending attack of Section E. The attack is neutralized since our verifier is more permissive, allowing the prover to construct a proof that takes superquality "goodness" into account when comparing forks. The modified construction is shown in Algorithm 8. The algorithm has been modified to check the current portion of the subchain α for *goodness* prior to moving to the lower superchain level. If goodness is indeed maintained at the current level μ, the prover only tries to cover the span of the last m blocks of level μ at level $\mu - 1$, as seen in Line 7. Otherwise, if goodness is violated at the part of the subchain α at level μ, then the prover completely ignores level μ and tries to use the lower level $\mu - 1$ to cover the whole span of α.

Algorithm 8 The *goodness aware* Prove algorithm for the NIPoPoW protocol

```
1: function $\mathsf{Prove}^{\mathsf{good}}_{m,k,\delta}(\mathcal{C})$
2:     $B \leftarrow \mathcal{C}[0]$                                  ▷ Genesis
3:     for $\mu = |\mathcal{C}[-k].\mathsf{interlink}|$ down to 0 do
4:         $\alpha \leftarrow \mathcal{C}[: -k]\{B :\}\uparrow^{\mu}$
5:         $\pi \leftarrow \pi \cup \alpha$
6:         if $\mathrm{good}_{\delta,m}(\mathcal{C}, \alpha, \mu)$ then
7:             $B \leftarrow \alpha[-m]$
8:         end if
9:     end for
10:    $\chi \leftarrow \mathcal{C}[-k :]$
11:    return $\pi\chi$
12: end function
```

Only the concrete prover needs to be modified. The verifier and $\leq_m$ operator remain as defined previously.

To aid intuition, we give a sketch of the proof before giving the full technical proof.

Theorem 4 (Security). *Assuming honest majority, the Non-interactive Proofs of Proof-of-Work construction for computable k-stable monotonic suffix-sensitive predicates is secure with overwhelming probability in κ.*

Proof (Sketch). 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 be 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. $\square$

We now give a formal treatment of the security proof. 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 11 (Typical execution). *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 $|S| \geq (1-\epsilon)2^\mu \frac{Z^\mu(S)}{pqt}$ with overwhelming probability in k.*

Chain regularity. *No insertions, no copies, and no predictions [10] have occurred.*

Theorem 5 (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 [10] 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 $\Pr[|S| < (1-\epsilon)\mathbb{E}(|S|)] \in \Omega(\epsilon^2 m)$. $\square$

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* $\mathcal{C}^S_B = \{b \in \mathcal{C}_B : b \text{ was generated during } S\}$*. 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 $Z^{\mu}(S) \geq k$. From the definition of typicality, we have $|S| \geq (1-\epsilon')2^{\mu_{\mathcal{A}}}\frac{1}{pqt}|\mathcal{C}'_{\mathcal{A}}|$. Applying the chain growth theorem [10] 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 $2^{\mu_{\mathcal{A}}}|\mathcal{C}'_{\mathcal{A}}| < \frac{pqt}{(1-\delta)(1-\epsilon')(1-\epsilon)f}2^{\mu_B}|\mathcal{C}^S_B{\uparrow}^{\mu_B}|$. $\square$

Definition 12 (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 the 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 $|\pi_B\{b:\}{\uparrow^{\mu^*}}[1:]| < m$ or $\neg\mathsf{local\text{-}good}_\delta(\pi\{b:\}{\uparrow^{\mu^*}}[1:], \mathcal{C}, \mu^*)$.

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

Remark 1 (Goodness adequacy). If the *goodness* of the chain can be assumed, then the adequate level of an honest proof is nothing else than the highest level having a sufficient (m) number of blocks after the fork point b. In that case, the proof for the above lemma is easy and follows from the prover construction. It always covers the last m blocks of level μ with the respective blocks in level $\mu - 1$.

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 4 (Security). *Assuming honest majority, the Non-interactive Proofs of Proof-of-Work construction for computable 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 monotonic 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 *some* honest participant. 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 $\pi_{\mathcal{A}} \geq_m \pi_B$, otherwise, because Q is suffix sensitive, Verify^Q would have returned $Q(\mathcal{C}_B)$. We now show that $\pi_{\mathcal{A}} \geq_m \pi_B$ 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 now show 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 $|\alpha_{\mathcal{A}}[1 :] \cap \alpha_B{\downarrow}\ [1 :]| \leq 2^{\mu'_B - \mu_{\mathcal{A}}} k_1$.

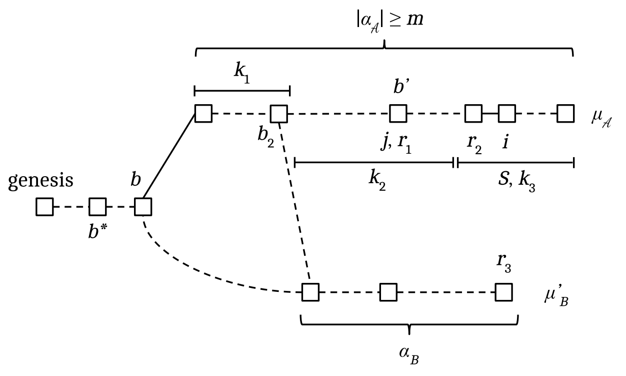
First, observe that, because the adversary is winning, therefore $|\alpha_{\mathcal{A}}| > 2^{\mu'_B - \mu_{\mathcal{A}}} m$. Suppose for contradiction that $|\alpha_{\mathcal{A}}[1 :] \cap \alpha_B{\downarrow}\ [1 :]| > 2^{\mu'_B - \mu_{\mathcal{A}}} k_1$. This means there are indices $1 \leq i < j$ 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.

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

Fig. 7. Two competing proofs at different levels.



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 $\mathcal{C}_{\mathcal{A}}[: -(k_2+k_1)] \not\preceq \mathcal{C}_B$. 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 [10] 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 2 (Variance attacks). The critical issue addressed by this security proof is to avoid Bahack-style attack [3] 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. Intuitively, the attack of Section E is neutralized, because our prover takes "goodness" of blockchains into account and the verifier does not compare proofs strictly at the same level.
Remark. We have explored security in the *synchronous* model. We remark that the same construction can work in a *partially synchronous* model by setting $k' = 2k$, where k' is the security parameter of the partially synchronous model and k is the security parameter in the synchronous model. We leave the full treatment of this for future work.

F.1 Infix security

We observe that now that we have proven the modified suffix construction secure, the security of infix proofs follows without any modifications in the infix construction. We formally state this in the following corollary.

Corollary 1. *Under honest majority, the infix NIPoPoW protocol (P, V) is secure for all computable infix-sensitive k-stable monotonic 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 4 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 [10], 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, it will be defined using a witness predicate D. Because Q is stable, we will have $\exists \mathcal{C}' \subseteq \mathcal{C}[: -k] : D(\mathcal{C}')$. But $\mathcal{C}' \subseteq \pi_B$. Let $S = \mathsf{ancestors}(b)$ be the ancestors evaluated by the verifier. As $\mathcal{C}' \subseteq S$, therefore $Q(\mathcal{C}') = Q(S) = v$. □

G 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.

G.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.

The safety of a hard fork is debated [6], as they are not "forward compatible". NIPoPoWs can also be implemented by a soft fork. 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 the interlink so that the adversary cannot cause a chain reorganization. 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 the 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.

G.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* [10].

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. Hence, while a hard fork makes new upgraded blocks invalid to unupgraded clients and a soft fork makes new unupgraded blocks invalid to upgraded clients, the velvet fork has the effect that blocks produced by either upgraded or unupgraded clients are valid for either. In reality, 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

[10] After the first manuscript of the present paper was published on the *ePrint* archive, velvet forks were subsequently explored in detail in the excellent follow-up work by Zamyatin et. al. [26]

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.

Algorithm 9 Supplying the necessary data to calculate a connected $\mathcal{C}\uparrow^\mu$ during a velvet fork.

```
1: function find $\mathcal{C}\uparrow^\mu(b, \text{realLink}, \text{blockById})$
2:     $B \leftarrow \mathcal{C}[-1]$
3:     $\mathsf{aux} \leftarrow \{B\}$
4:     $\pi \leftarrow [\,]$
5:     if $\textit{level}(B) \geq \mu$ then
6:         $\pi \leftarrow \pi B$
7:     end if
8:     while $B \neq b$ do
9:         $(B, \mathsf{aux'}) \leftarrow \mathsf{followUp}(B, \mu, \mathsf{realLink}, \mathsf{blockById})$
10:        $\mathsf{aux} \leftarrow \mathsf{aux} \cup \mathsf{aux'}$
11:        $\pi \leftarrow \pi B$
12:    end while
13:    return $\pi$, aux
14: end function
```

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

```
function followUp($B$, $\mu$, realLink, blockById)
    aux $\leftarrow \{B\}$
    while $B \neq$ Gen do
        if $B$.interlink[$\mu$] = realLink[id($B$)][$\mu$] then
            $id \leftarrow B$.interlink[$\mu$]
        else                                          ▷ Invalid interlink
            $id \leftarrow B$.previd
        end if
        $B \leftarrow$ blockById[$id$]
        aux $\leftarrow$ aux $\cup \{B\}$
        if $level(B) = \mu$ then
            return $B$, aux
        end if
    end while
    return $B$, aux
end function
```

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

```
1: function Prove'$_{m,k}$($\mathcal{C}$, realLink, blockById)
2:     $max\mu \leftarrow |\mathsf{realLink}[\mathsf{id}(\mathcal{C}[-k-1])]|$
3:     $b \leftarrow \mathcal{C}[0]$                                    ▷ Genesis block
4:     $\tilde{\Pi} \leftarrow \emptyset$
5:     for $\mu = max\mu$ down to 0 do
6:         $\pi, aux \leftarrow$ find $\mathcal{C}[:-k]{\uparrow}^{\mu}$ ($b$, realLink, blockById)
7:         if $|\pi| \geq m$ then
8:             $b \leftarrow \pi[-m]$
9:         end if
10:        $\tilde{\Pi} \leftarrow \tilde{\Pi} \cup aux$
11:    end for
12:    $\chi \leftarrow \mathcal{C}[-k:]$
13:    return $\tilde{\Pi}\chi$
14: end function
```

Velvet NIPoPoWs preserve security. Additionally, if a constant minority of miners has upgraded their nodes, then succinctness is also preserved as there is only a constant factor penaltyas proven in the following theorem.

Theorem 6. *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 9 we know that the proofs π contain only a $O(poly\text{-}log(m))$ 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. □

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, making it a *user-activated velvet fork*. 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.

The necessary changes needed in the various construction algorithms in order to support a velvet fork are shown in Algorithm 9, Algorithm 10, and Algorithm 11.

Supporting clients with different beliefs. The interlink format does not depend on parameters m, k. Therefore, it is not necessary to agree on a particular value of these parameters. Instead, the choice of m and k can be a user-configurable parameter to clients. Clients would send a particular m and k as part of their requirement to the prover.

H Chains of Variable Difficulty

In this paper, we have explored constructing PoPoWs always assuming that the mining target T is constant. In this section, we give the intuition behind a *variable difficulty* PoPoW construction based on our constant difficulty construction.

The construction is modified as follows. Assume the chain adjusts its difficulty in the usual manner as formalized, e.g., in [11]. In this case, an *epoch* e is a 0-subchain of fixed length, say ℓ, during which the difficulty remains the same and which is examined in order to perform the difficulty recalculation. The target has a maximum value in which the difficulty is the easiest. Suppose this value is $T_0 = \frac{1}{2^\kappa}$ such that all attempted blocks are valid. We consider the case where the current difficulty is possible to adjust by factors of 2; that is, the target of each epoch e is quantized as $T_e = \frac{T_0}{2^{\mu^*}}$ for some non-negative integer μ^*, the current *mining level.* In this context, changing the difficulty is equivalent to saying that the mining game will be played at some level $\mu^* \geq 0$ and every node considers only chains of that level, ignoring chains of inferior level. As such, superblock NIPoPoWs can be constructed as usual, except the used target for NIPoPoW generation and verification will be the maximum target T_0 instead of the current mining target T_e. The only difference is that the honest prover will only provide chains of level $\mu \geq \mu^*$, as inferior blocks will neither be mined nor successfully validated by honest participants.

The size of variable difficulty NIPoPoWs is the same as in the constant difficulty case.

The security arguments follow through as before when the comparison of two proofs $\pi_{\mathcal{A}}, \pi_B$ happens at some level $\mu_{\mathcal{A}}, \mu_B \geq \mu^*$. We must therefore only concern ourselves for the case, during the comparison, when the adversary presents blocks of level inferior to μ^* after the LCA block b. In the case of an interactive PoPoW, the honest prover can be informed of the attempt of the adversary to present blocks of level inferior to μ^*. However, the adversary has committed to the LCA block b. The honest party can subsequently provide a *certificate of hardness* by presenting the full epoch immediately preceding block b, which will consist of a constant number of blocks, ℓ. This will allow the honest party to challenge the adversary's ability to descend to level $\mu_B < \mu^*$ and allow the verifier to decide that the honest party B should be victorious. This PoPoW protocol can be made non-interactive by including a certificate of hardness for every block among the last m blocks of each level presented. This harms the succinctness by a constant factor of ℓ (in Bitcoin, $\ell = 2016$).

The above sketch gives an intuition of how proofs can be modified to work in a variable difficulty setting. Formal analysis of its security and succinctness in various conditions in the model of [11], including relaxing the requirement of quantization $\mu^* \in \mathbb{N}$, will be explored in future work.

I Succinctness

I.1 Optimistic succinctness

We analyse the patched scheme we saw in Algorithm 8. We will illustrate why our construction is succinct in the honest setting. We then discuss techniques to make the construction succinct in broader adversarial settings.

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 E.1. By reducing superquality for a sufficiently low level μ, 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.

Definition 13 (Optimistic execution). *We will call an execution* optimistic *if the adversary has $q = 0$ random oracle queries and the messages diffused by honest parties are delivered in random (and not adversarial) order.*

In this setting, 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.

Theorem 7 (Number of levels). *In any execution, let S denote the set of all blocks produced honestly or adversarially. The number of superblock levels which have at least m blocks are at most $\log(|S|)$, with overwhelming probability in m.*

Proof. Each block id in S 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 $D_\mu = \sum_{B \in S} X_B^\mu$ indicate their sum, which is a Binomial distribution with parameters $(|S|, 2^{-\mu})$ and expectation $E[D_\mu] = |S|2^{-\mu}$.

All of the X^μ are independent. We apply a Binomial Chernoff bound to the sum. We have $\Pr[D_\mu \geq (1 + \delta)E[D_\mu]] \leq \exp(-\frac{\delta^2}{3}E[D_\mu])$. Letting $\mu = \log(|S|)$ we have that $E[D_\mu] = 1$. Therefore $\Pr[D_\mu \geq 1 + \delta] \leq \exp(-\frac{\delta^2}{3})$. Requiring $1 + \delta = m$, we get $\Pr[D_\mu \geq m] \leq \exp(-\frac{(m-1)^2}{3})$, which is negligible in m. $\square$

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 8 (Large upchain expansion). *Consider an optimistic execution and let $\mathcal{C}$ be an honestly adopted chain and let $\mathcal{C}' = \mathcal{C}{\uparrow}^{\mu-1} [i : i + 4m]$ for any i. Then $|\mathcal{C}'{\uparrow}^\mu| \geq m$ with overwhelming probability in m.*

Proof. 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 $(4m, 1/2)$. Observing that $E[\mathcal{C}'{\uparrow}^{\mu}] = 2m$ and applying a Chernoff bound, we get $\Pr[|\mathcal{C}'{\uparrow}^{\mu}| \leq m] = \Pr[|\mathcal{C}'{\uparrow}^{\mu}| \leq (1 - \frac{1}{2})2m] \leq \exp(-\frac{(1/2)^2}{2}2m)$ which is negligible in m.

$\square$

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). *Consider an optimistic execution and let $\mathcal{C}$ be an honestly adopted chain and $\mathcal{C}' = \mathcal{C}{\uparrow}^{\mu}[i : i + m]$. 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 8 it follows that more than m blocks exist in level μ with overwhelming probability in m, which is a contradiction. $\square$

This last lemma 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 9 (Optimistic succinctness). *In an optimistic execution, Non-Interactive Proofs of Proof-of-Work produced by honest provers are succinct with the number of blocks bounded by $4m\log(|\mathcal{C}|)$, with overwhelming probability in m.*

Proof. Assume $\mathcal{C}$ is an honest party's chain. From Theorem 7, the number of levels in the NIPoPoW is at most $\log(|\mathcal{C}|)$ with overwhelming probability in m (note that $|\mathcal{C}| \sim \Theta(|S|)$). First, observe that the count of blocks in the highest level will be less than $4m$ from Theorem 8; otherwise a higher superblock level would exist. From Lemma 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 $4m\log(|\mathcal{C}|)$. $\square$

In the above theorem, note the linear dependency between the round r during which a proof is generated and the length $|\mathcal{C}|$ of the chain of each honest prover.

I.2 Succinctness of adversarial proofs

In the stronger adversarial setting, however, 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. However, simply dismissing superpolylogarithmic proofs is an incorrect strategy, as honest provers can produce such longer proofs in case an adversarial miner harms the goodness of the blockchain.

It would therefore 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. Our extended construction allows the verifier to stop processing input early, in a streaming fashion, thereby only requiring logarithmic communication complexity per proof received. To achieve this, observe that honest proofs need to be large only if there is a violation of *goodness*. However, goodness is not harmed when the chain is not under attack by the adversarial computational power or network. Therefore, 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 $\mu' - \mu \leq \text{max-}\mu$. 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 12.

Algorithm 12 The badness prover which generates a succinct certificate of badness

```
1: function badness_{m,δ}($\mathcal{C}$)
2:     $M \leftarrow \{\mu : |\mathcal{C}{\uparrow}^{\mu}| \geq m\} \setminus \{0\}$
3:     $\rho \leftarrow 1/\max(M)$
4:     for $\mu \in M$ do
5:         for $B \in \mathcal{C}{\uparrow}^{\mu}$ do
6:             $\mathcal{C}' \leftarrow \mathcal{C}{\uparrow}^{\mu}\{B:\}[:m]$
7:             if $|\mathcal{C}'| = m$ then
8:                 ▷ Sliding m-sized window
9:                 $\mathcal{C}^* \leftarrow \mathcal{C}'{\downarrow}{\uparrow}^{\mu-1}$
10:                if $2|\mathcal{C}'| < (1-\delta)^{\rho}|\mathcal{C}^*|$ then
11:                    return $\mathcal{C}^*$                ▷ Chain is bad
12:                end if
13:            end if
14:        end for
15:    end for
16:    return $\perp$                                     ▷ Chain is good
17: end function
```

Therefore, we augment the NIPoPoW construction as follows. The honest prover sends a tuple of two items. The first item is empty if the second item is polylogarithmic in the size of the chain; otherwise it is a certificate of badness. The second item is the NIPoPoW proof as in the previous construction. The verifier processes only the first polylogarithmic number of bytes from the incoming proof. If within that portion a certificate of badness is found, it is checked for validity. If it is found to be valid, the whole proof is checked, regardless of size. If it is found to be invalid or no certificate has been provided, then the proof is rejected as invalid. We call the augmented construction *certified NIPoPoWs*.

Lemma 9 (Certified NIPoPoWs succinctness). *If all miners are honest and the network scheduling is random, certified non-interactive proofs-of-proof-of-work produced by the adversary are processed in poly-*

logarithmic time in the size of the chain by honest verifiers, except with negligible probability in m.

Proof. Because all miners are honest and the network scheduling is random, therefore certificates of badness exist with negligible probability in m. Conditioning on the event that certificates of badness do not exist, the honest verifier will reject the proof in polylogarithmic time. □

We also establish that the modified construction does not harm security below. Security is established in the general case where the adversary has minority mining power.

Theorem 10 (Certified NIPoPoWs security). *Assuming honest majority, certified non-interactive proofs-of-proof-of-work are secure, except with negligible probability in κ.*

Proof. We distinguish two cases: Either goodness has been violated; or it has not been violated. Suppose that goodness has been violated. In that case, an honest prover will include a certificate of badness in their proof and their proof will be processed by an honest verifier.

In the case where goodness is not violated, all honest proofs will be logarithmic in size as established by Lemma 9. Therefore, all honest proofs will be processed by an honest verifier.

Under the condition that all honest proofs will be processed, the rest of the security argument follows immediately from Theorem 4. □

I.3 Infix succinctness

Having established the succinctness of the modified suffix construction, the succinctness of the infix construction follows in the next corollary.

Corollary 2. *The infix NIPoPoW protocol (P, V) is succinct for all computable infix-sensitive k-stable predicates Q in which the witness predicate D depends on a* polylogarithmic *number of blocks $d(|\mathcal{C}|)$.*

Proof. 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 $|\mathcal{C}'| \in O(polylog(|\mathcal{C}|))$. □

J Acknowledgements

The authors wish to thank Giorgos Panagiotakos at the University of Edinburgh, Peter Gaži at IOHK, and Nikos Leonardos at the University of Athens for their insights regarding the attack against the interactive Proofs of Proof-of-Work scheme. We also wish to thank Giorgos Christoglou at Imperial College London and Kostis Karantias at the University of Ioannina for their insights and comments while they implemented the schemes described in this paper as part of their master theses, as well as Sebastien Guillemot at Emurgo R&D who provided valuable feedback. All three of them read many versions of this paper in detail and their contributions helped significantly shape and improve it.

References

1. Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 375–392. IEEE, 2017.
2. Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 2014.
3. Lear Bahack. Theoretical bitcoin attacks with less than half of the computational power (draft). *arXiv preprint arXiv:1312.7013*, 2013.
4. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.
5. Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 104–121. IEEE, 2015.
6. Vitalik Buterin. Hard forks, soft forks, defaults and coercion, 2017.
7. Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.
8. John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.
9. Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.
10. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.
11. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *Annual International Cryptology Conference*, pages 291–323. Springer, 2017.
12. Peter Gazi, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *IEEE Symposium on Security & Privacy*, 2019.

13. Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security Symposium*, pages 129–144, 2015.
14. Maurice Herlihy. Atomic cross-chain swaps. *arXiv preprint arXiv:1801.09515*, 2018.
15. Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In *International Conference on Financial Cryptography and Data Security*, pages 61–78. Springer, 2016.
16. Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.
17. Andrew Miller. The high-value-hash highway, bitcoin forum post, 2012.
18. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
19. Tier Nolan. Alt chains and atomic transfers. `bitcointalk.org`, May 2013.
20. Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2017.
21. Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 315–324. ACM, 2017.
22. William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.
23. Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014.
24. Fabian Vogelsteller and Vitalik Buterin. Erc-20 token standard. sept. 2017. *URl: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-tokenstandard.md*, 2015.
25. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151:1–32, 2014.
26. Alexei Zamyatin, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Edgar Weippl, William Knottenbelt, and Alexei Zamyatin. A wild velvet fork appears! inclusive blockchain protocol changes in practice. In *International Conference on Financial Cryptography and Data Security*. Springer, 2018.