# Non-Interactive Proofs of Proof-of-Work under Velvet Fork

Anonymous Author(s)

## ABSTRACT

Superlight clients allow decentralized wallets to learn facts about the blockchain without downloading all the block headers. They achieve exponentially faster communication compared to SPV clients. For proof-of-work, they implement the so-called NIPoPoW primitive, for which there exist two variants: Superblock clients and FlyClient. Both of these protocols require consensus changes to existing blockchains and at least a soft fork to implement. In this paper, we discuss how a blockchain can be upgraded to support superblock clients without a soft fork. We show that it is possible to implement the needed changes without modifying the consensus protocol and by requiring only a minority of miners to upgrade, an upgrade termed a "velvet fork" in the literature. While previous work conjectured that NIPoPoW can be safely deployed using velvet forks as-is, we show that previous constructions are insecure, and that using velvet techniques to interlink a blockchain can pose insidious security risks. We describe a novel attack which we term a "chain-sewing" attack which is only possible in velvet situations: An adversary can cut-and-paste portions of various chains from independent forks, sewing them together to form a proof that looks like a chain but is not. We demonstrate that a minority adversary can thwart previous constructions with overwhelming probability. We put forth the first provably secure velvet NIPoPoW construction. Our construction is secure against adversaries that are bounded by 1/4 of the upgraded honest miner population. We prove our construction achieves persistence and liveness and analyze the trade-offs between the upgraded population parameter and the succinctness of the construction. Like non-velvet NIPoPoWs, our approach allows proving generic predicates about chains using infix proofs and as such can be adopted in practice for fast synchronization of transactions and accounts.

## CCS CONCEPTS

• **Security and privacy** → Use https://dl.acm.org/ccs.cfm to generate actual concepts section for your paper;

## KEYWORDS

blockchain, consensus, lightclient, NIPoPoW

## 1 INTRODUCTION

Blockchain systems such as Bitcoin [22] and Ethereum [3, 25] have a predetermined expected rate of block production and maintain chains of blocks that are growing linearly with time. A node synchronizing with the rest of the blockchain network for the first time therefore has to download and validate the whole chain, if it does not wish to rely on a trusted third party. While a lightweight node (SPV) can avoid downloading and validating transactions beyond their interest, it must still download the block headers that contain the proof-of-work[5] of each block in order to determine which chain contains the most proof-of-work. Block headers, while smaller by a significant constant factor, still grow linearly with time. An Ethereum node synchronizing for the first time must download more than 300 MB of block header data for the purpose of proof-of-work verification, even if it elects not to download any transactions. This has become a central problem to the usability of blockchain systems, especially for vendors who are using mobile phones to accept payments or sit behind limited internet bandwidth. They are forced to make a difficult choice between decentralization and the ability to start accepting payments in a timely manner.

Towards the goal of alleviating the burden of this download for SPV clients, a series of *superlight* clients has emerged. These clients are able to choose the best proof-of-work chain by only requesting a small number of *sample* block headers instead of all the block headers. These sample blocks form a Non-Interactive Proof of Proof-of-Work (NIPoPoW), a short string which compresses the proof-of-work of the underlying chain. It has been shown that the block headers in these proofs are secure representatives of the proof-of-work of the underlying chain: A minority adversary can only convince a NIPoPoW client that a certain transaction is confirmed, only if they can convince an SPV client, too.

There are two general directions for superlight client implementations: In the *superblock* [10**?** ] approach, the client relies on *superblocks*, blocks that have achieved much better proof-of-work than required for block validity. In the *FlyClient* [2] approach, blocks are sampled at random using a predetermined distribution, akin to a Schnorr protocol [23] modified with the Fiat–Shamir heuristic [7]. The number of block headers that need to be sent then grows only logarithmically with time. The NIPoPoW client, termed a *verifier*, still relies on a connection to full nodes, termed *provers*, which perform the sampling of blocks from the full blockchain. However, no trust assumptions are made for these provers, as the verifier can check the veracity of their claims. As long as the verifier is connected to at least one honest prover (an assumption also made in the SPV protocol [9, 26]), they are able to discern the best chain from the rest.

In both approaches, it is essential for the verifier to check that the blocks sampled one way or another have been generated in the same order as they have been presented by the prover. As such, each block in the proof must contain a pointer to the previous block in the proof. As blocks in these proofs are far apart in the underlying blockchain, the legacy *previous block pointer*, which typically appears within block headers, does not suffice. Both approaches require modifications to the consensus layer of the underlying blockchain to work. In the case of superblock NIPoPoWs, the block header must be modified to include, in addition to a pointer to the previous block, pointers to a small amount of recent high-proof-of-work blocks. In the case of FlyClient, each block must additionally contain pointers to all previous blocks in the chain. Both of these modifications can be made efficiently by organizing these pointers into Merkle Trees [21] or Merkle Mountain Ranges [18, 24] whose root is stored in the block header. The inclusion of extra pointers within blocks is termed *interlinking the chain* [5].

The modified block format, which includes the extra pointers, must be respected and validated by all full nodes and thus requires either a hard fork or at least a soft fork. However, even soft forks require the approval of a supermajority of miners, and new features that are considered non-essential by the community have taken years to receive approval [19]. Towards the goal of implementing superlight clients sooner, we study the question of whether it is possible to deploy superlight clients without a soft fork. We propose a series of modifications to blocks that are *helpful but untrusted*. These modifications mandate that some extra data is included in each block. The extra data is placed inside the block by upgraded miners only, while the rest of the network does not include the additional data into the blocks and does not verify its inclusion, treating them merely as comments. To maintain backwards compatibility, contrary to a soft fork, upgraded miners must accept blocks that do not contain this extra data that have been produced by unupgraded miners, or even blocks that contain invalid or malicious such extra data produced by a mining adversary. This acceptance is necessary in order to avoid causing a chain split with the unupgraded part of the network. Such a modification to the consensus layer is termed a *velvet fork* [28].

A summary of our contributions in this paper is as follows:

(1) We propose the first *backwards-compatible superlight client*. We put forth an interlinking mechanism implementable through a velvet fork. We then construct a superblock NIPoPoW protocol on top of the velvet forked chain.
(2) We prove our construction secure in the synchronous static difficulty model against adversaries bounded to $1/4$ of the mining power of the honest upgraded nodes. As such, our protocol works even if a constant minority of miners adopts it.
(3) We illustrate that, contrary to claims of previous work, superlight clients designed to work in a soft fork cannot be readily plugged into a velvet fork and expected to work. We present a novel and insidious attack termed the *chain-sewing* attack which thwarts the defenses of previous proposals and allows even a minority adversary to cause catastrophic failures.

**Previous work.** Proofs of Proof-of-Work have been proposed in the context of superlight clients [2, 5, 15], cross-chain communication [11, 16, 27], as well as local data consumption by smart contracts [12]. Superblock NIPoPoWs have been conjectured to work in velvet fork conditions [15], but we show here that these conjectures are ill-informed in the light of our chain-sewing attack. Velvet forks [28] have been studied for a variety of other applications. In this work, we focus on consensus state compression. Such compression has been explored in the hard-fork setting using zk-SNARKS [20] as well as in the Proof-of-Stake setting [13]. Complementary to consensus state compression is compression of application state, namely the State Trie, the UTXO, or transaction history. There is a series of works composable with ours that discusses the compression of application state [4, 17].

[8][6]

## 2 PRELIMINARIES

We consider a setting where the blockchain network consists of two different types of nodes: The first kind, *full nodes*, are responsible for the maintenance of the chain including verifying it and mining new blocks. The second kind, *verifiers* connect to full nodes and wish to learn facts about the blockchain without downloading it, for example whether a particular transaction is confirmed. The full nodes therefore also function as *provers* for the verifiers. Each verifier connects to multiple provers, at least one of which is assumed to be honest.

We model full nodes according to the Backbone model [8]. There are $n$ full nodes, of which $t$ are adversarial and $n - t$ are honest. All $t$ adversarial parties are controlled by one colluding adversary $\mathcal{A}$. The parties have access to a hash function $H$ which is modelled as a common Random Oracle [1]. To each novel query, the random oracle outputs $\kappa$ bits of fresh randomness. Time is split into distinct *rounds* numbered by the integers $1, 2, \cdots$. Our treatment is in the *synchronous model*, so we assume messages *diffused* (broadcast) by an honest party at the end of a round are received by all honest parties at the beginning of the next round. This is equivalent to a network connectivity assumption in which the round duration is taken to be the known time needed for a message to cross the diameter of the network. The adversary can inject messages, reorder them, sybil attack by creating multiple messages, but not suppress messages.

Each honest full node locally maintains a *chain* $\mathsf{C}$, a sequence of blocks. In understanding that we are developing an improvement on top of SPV, we use the term *block* to mean what is typically referred to as a *block header*. Each block contains the Merkle Tree root [21] of transaction data $\overline{x}$, the hash $s$ of the previous block in the chain known as the *previd*, as well as a nonce value $ctr$. As discussed in the Introduction, the compression of application data $\overline{x}$ is orthogonal to our goals in this paper and has been explored in independent work [4] which can be composed with ours. Each block $b = s \parallel \overline{x} \parallel ctr$ must satisfy the proof-of-work [5] equation $H(b) \leq T$ where $T$ is a constant *target*, a small value signifying the difficulty of the proof-of-work problem. Our treatment is in the *static difficulty* case, so we assume that $T$ is constant throughout the execution[1]. $H(B)$ is known as the *block id*.

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 $\mathcal{G}$, a special block known to all parties. This is the only node the verifier knows about when it boots up. For chain addressing we use Python brackets $\mathsf{C}[\cdot]$. 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., $\mathsf{C}[-1]$ is the tip of the blockchain. A range $\mathsf{C}[i{:}j]$ is a subarray starting from $i$ (inclusive) to j (exclusive). Given chains $\mathsf{C}_1, \mathsf{C}_2$ and blocks $A, Z$ we concatenate them as $\mathsf{C}_1\mathsf{C}_2$ or $\mathsf{C}_1 A$ (if clarity mandates it, we also use the symbol $\parallel$ for concatenation). Here, $\mathsf{C}_2[0]$ must point to $\mathsf{C}_1[-1]$ and $A$ must point to $\mathsf{C}_1[-1]$. We denote $\mathsf{C}\{A{:}Z\}$ the subarray of the chain from block $A$ (inclusive) to block $Z$ (exclusive). We can omit blocks or indices from either side

[1]A treatment of variable difficulty NIPoPoWs has been explored in the soft fork case [29], but we leave the treatment of velvet fork NIPoPoWs in the variable difficulty model for future work.

of the range to take the chain to the beginning or end respectively. As long as the blockchain property is maintained, we freely use the set operators $\cup$, $\cap$ and $\subseteq$ to denote operations between chains, implying that the appropriate blocks are selected and then placed in chronological order.

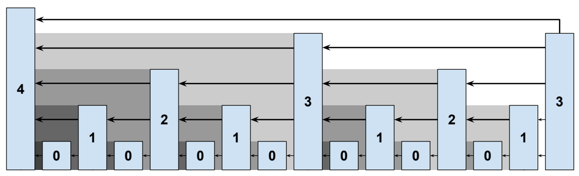
During every round, every party attempts to *mine* a new block on top of its currently adopted chain. Each party is given $q$ queries to the random oracle which it uses in attempting to mine a new block. Therefore the adversary has $tq$ queries per round while the honest parties have $(n-t)q$ queries per round. When an honest party discovers a new block, they extend their chain with it and broadcast the new chain. Upon receiving a new chain $\mathsf{C}'$ from the network, an honest party compares its length $|\mathsf{C}'|$ against its currently adopted chain $\mathsf{C}$ and adopts the newly received chain if it is longer. It is assumed that the honest parties control the majority of the computational power of the network. This *honest majority assumption* states that there is some $\delta$ such that $t < (1-\delta)(n-t)$. If so, the protocol ensures consensus among the honest parties: There is a constant $k$, the *Common Prefix* parameter, such that, at any round, all the chains belonging to honest parties share a common prefix of blocks; the chains can deviate only up to $k$ blocks at the end of each chain [8]. Concretely, if at some round $r$ two honest parties have $\mathsf{C}_1$ and $\mathsf{C}_2$ respectively, then either $\mathsf{C}_1[:-k]$ is a prefix of $\mathsf{C}_2$ or vice versa.

Some valid blocks satisfy the proof-of-work equation better than required. If a block $b$ satisfies $H(b) \leq 2^{-\mu}T$ for some natural number $\mu \in \mathbb{N}$ we say that $b$ is a *$\mu$-superblock* or a block *of level $\mu$*. The probability of a new valid block achieving level $\mu$ is $2^{-\mu}$. The number of levels in the chain will be $\log|\mathsf{C}|$ with high probability [14]. Given a chain $\mathsf{C}$, we denote $\mathsf{C}\uparrow^{\mu}$ the subset of $\mu$-superblocks of $\mathsf{C}$.

Non-Interactive Proofs of Proof-of-Work (NIPoPoW) protocols allow verifiers to learn the most recent $k$ blocks of the blockchain adopted by an honest full node without downloading the whole chain. The challenge lies in building a verifier who can find the suffix of the longest chain between claims of both honest and adversarial provers, while not downloading all block headers. Towards that goal, the *superblock* approach uses superblocks as samples of proof-of-work. The prover sends superblocks to the verifier to convince them that proof-of-work has taken place without actually presenting all this proof-of-work. The protocol is parametrized by a constant security parameter $m$. The parameter determines how many superblocks will be sent by the prover to the verifier and security is proven with overwhelming probability in $m$.

The prover selects various levels $\mu$ and for each such level sends a carefully chosen portion of its $\mu$-level *superchain* $\mathsf{C}\uparrow^{\mu}$ to the verifier. In standard blockchain protocols such as Bitcoin and Ethereum, each block $\mathsf{C}[i+1]$ in $\mathsf{C}$ points to its previous block $\mathsf{C}[i]$, but each $\mu$-superblock $\mathsf{C}\uparrow^{\mu}[i+1]$ does not point to its previous $\mu$-superblock $\mathsf{C}\uparrow^{\mu}[i]$. It is imperative that an adversarial prover does not reorder the blocks within a superchain, but the verifier cannot verify this unless each $\mu$-superblock points to its most recently preceding $\mu$-superblock. The proposal is therefore to *interlink* the chain by having each $\mu$-superblock include an extra pointer to its most recently preceding $\mu$-superblock. To ensure integrity, this pointer must be included in the block header and verified by proof-of-work. However, the miner does not know which level a candidate block will attain prior to mining it. For this purpose, each block is proposed to include a pointer to the most recently preceding $\mu$-superblock, for every $\mu$, as illustrated in Figure 1. As these levels are only $\log|\mathsf{C}|$, this only adds $\log|\mathsf{C}|$ extra pointers to each block header.

**Figure 1: The interlinked blockchain. Each superblock is drawn taller according to its achieved level. Each block links to all the blocks that are not being overshadowed by their descendants. The most recent (right-most) block links to the four blocks it has direct line-of-sight to.**



The exact NIPoPoW protocol works like this: The prover holds a full chain $\mathsf{C}$. When the verifier requests a proof, the prover sends the last $k$ blocks of their chain, the suffix $\chi = \mathsf{C}[-k:]$, in full. From the larger prefix $\mathsf{C}[:-k]$, the prover constructs a proof $\pi$ by selecting certain superblocks as representative samples of the proof-of-work that took place. The blocks are picked as follows. The prover selects the *highest* level $\mu^*$ that has at least $m$ blocks in it and includes all these blocks in their proof (if no such level exists, the chain is small and can be sent in full). The prover then iterates from level $\mu = \mu^* - 1$ down to 0. For every level $\mu$, it includes sufficient $\mu$-superblocks to cover the last $m$ blocks of level $\mu + 1$, as illustrated in Algorithm 1. Because the density of blocks doubles as levels are descended, the proof will contain in expectation $2m$ blocks for each level below $\mu^*$. As such, the total proof size $\pi\chi$ will be $\Theta(m \log|\mathsf{C}| + k)$. Such proofs that are polylogarithmic in the chain size constitute an exponential improvement over traditional SPV clients and are called *succinct*.

**Algorithm 1** The Prove algorithm for the NIPoPoW protocol in a soft fork

```
function Prove_{m,k}(C)
    $B \leftarrow \mathsf{C}[0]$                                      ▷ Genesis
    for $\mu = |\mathsf{C}[-k-1].\text{interlink}|$ down to 0 do
        $\alpha \leftarrow \mathsf{C}[:-k]\{B:\}\uparrow^{\mu}$
        $\pi \leftarrow \pi \cup \alpha$
        if $m < |\alpha|$ then
            $B \leftarrow \alpha[-m]$
        end if
    end for
    $\chi \leftarrow \mathsf{C}[-k:]$
    return $\pi\chi$
end function
```

Upon receiving two proofs $\pi_1\chi_1, \pi_2\chi_2$ of this form, the NIPoPoW verifier first checks that $|\chi_1| = |\chi_2| = k$ and that $\pi_1\chi_1$ and $\pi_2\chi_2$ form valid chains. To check that they are valid chains, the verifier ensures every block in the proof contains a pointer to its previous block inside the proof through either the *previd* pointer in the block header, or in the interlink vector. If any of these checks fail, the proof is rejected. It then compares $\pi_1$ against $\pi_2$ using the $\leq_m$ operator, which works as follows. It finds the lowest common ancestor block $b = (\pi_1 \cap \pi_2)[-1]$; that is, $b$ is the most recent block shared among the two proofs. Subsequently, it chooses the level $\mu_1$ for $\pi_1$ such that $|\pi_1\{b{:}\}{\uparrow}^{\mu_1}| \geq m$ (i.e., $\pi_1$ has at least $m$ superblocks of level $\mu_1$ following block $b$) and the value $2^{\mu_1}|\pi_1\{b{:}\}{\uparrow}^{\mu_1}|$ is maximized. It chooses a level $\mu_2$ for $\pi_2$ in the same fashion. The two proofs are compared by checking whether $2^{\mu_1}|\pi_1\{b{:}\}{\uparrow}^{\mu_1}| \geq 2^{\mu_2}|\pi_2\{b{:}\}{\uparrow}^{\mu_2}|$ and the proof with the largest score is deemed the winner. The comparison is illustrated in Algorithm 2.

**Algorithm 2** The implementation of the $\geq_m$ operator to compare two NIPoPoW proofs 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$_m(\pi, b)$
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
```

Blockchain protocols can be upgraded using hard or soft forks [? ]. In a *hard fork*, blocks produced by upgraded miners are not accepted by unupgraded miners. It is simplest to introduce interlinks using a hard fork by mandating that interlink pointers are included in additional fields in the block header. Unupgraded miners will not recognize these fields and will be unable to parse upgraded blocks. To ensure the block header is of constant size, instead of including all these superblock pointers in the block header individually, they are organized into a Merkle Tree of interlink pointers and only the root of the Merkle Tree is included in the block header. In this case, the NIPoPoW prover that wishes to show a block $b$ in their proof is connected to its more recently preceding $\mu$-superblock $b'$, also includes a Merkle Tree proof proving that $H(b')$ is a leaf in the interlink Merkle Tree root included in the block header of $b$. The verifier has the additional job of verifying the veracity of these Merkle inclusion proofs.

In a *soft forks*, blocks created by unupgraded miners are not accepted by upgraded miners, but blocks created by upgraded miners are accepted by unupgraded miners. As such, any additional data introduced by the upgrade must be included in a field that is treated like a comment by an unupgraded miner. To interlink the chain via a soft fork, the interlink Merkle Tree root is placed in the *coinbase* transaction instead of the block header. Upgraded miners include the correct interlink Merkle Tree root in their coinbase. Upgraded miners receiving a new block validate that the interlink Merkle Tree root is correct before accepting a block as valid. As this root can be calculated in a deterministic manner from the previous blocks in the chain, it can easily be validated. Unupgraded miners ignore this data and accept the block regardless of whether it exists. The success of the fork depends on the majority of the miner population upgrading. Whenever the NIPoPoW prover wishes to show that a block $b$ in the proof contains a pointer to its most recently preceding $\mu$-superblock $b'$, it must then accompany the block header of $b = s \parallel \overline{x} \parallel ctr$ with the coinbase transaction $\text{tx}_{\text{cb}}$ of $b$ as well as two Merkle Tree proofs: One proving that the coinbase transaction $\text{tx}_{\text{cb}}$ is in $\overline{x}$, and one proving that $H(b')$ is a leaf in the interlink Merkle Tree whose root is included in $\text{tx}_{\text{cb}}$.

Lastly, mostly recently, velvet forks have been introduced [28]. In a velvet fork, blocks created by upgraded miners (called *velvet blocks*) are accepted by unupgraded miners as in a soft fork. Additionally, blocks created by unupgraded miners are also accepted by upgraded miners. This allows the protocol to upgrade even if only a minority of miners chooses to upgrade. To maintain backwards compatibility and to avoid causing forks, the additional data included in a block is *advisory* and must be accepted whether it exists or not. Even if the additional data is invalid or malicious, upgraded nodes (in this context also called *velvet nodes*) are forced to accept the blocks. The simplest approach to velvet fork the chain for interlinking purposes is to have upgraded miners include the interlink pointer in the blocks they produce, but accept blocks with missing or incorrect interlinks. As we show in the next section, this approach is flawed and susceptible to unexpected attacks. A surginal change in the way velvet blocks are produced is necessary to achieve proper security.

## 3 VELVET INTERLINKS

### 3.1 Velvet fork parameter

In a velvet fork, only a minority of honest parties needs to support the protocol changes. We refer to this percentage as the "velvet parameter".

*Definition 3.1 (Velvet Parameter).* The *velvet parameter g* is defined as the percentage of honest parties that have upgraded to the new protocol. The absolute number of honest upgraded parties is denoted $n_h$ and it holds that $n_h = g(n - t)$.

### 3.2 Smooth and Thorny blocks

In order to be applied under velvet fork, a protocol has to change in a backwards-compatible manner. In essence, any additional information coming with the protocol upgrade is transparent to the non-upgraded players. This transparency towards the non-upgraded parties requires any block that conforms only to the old protocol rules to be considered a valid one. Considering superblock NIPoPoWs under a velvet fork, any block is to be checked for its validity regardless the validity of the NIPoPoWs protocol's additional information, which is the interlink structure.

A block generated by the adversary could thus contain arbitrary data in the interlink and yet be appended in the chain adopted by an honest party. In case that trash data are stored in the structure this could be of no harm for the protocol routines, since such blocks will be treated as non-upgraded. In the context of the attack that will be presented in the following section, we examine the case where the adversary includes false interlink pointers.

An interlink pointer is the hash of a block. A correct interlink pointer of a block $b$ for a specific level $\mu$ is a pointer to the most recent $b$'s ancestor of level $\mu$. From now on we will refer to correct interlink pointers as *smooth pointers.* Pointers of the 0-level *(prevIds)* are always smooth because of the performed proof-of-work.

*Definition 3.2 (Smooth Pointer).* Smooth pointer of a block $b$ for a specific level $\mu$ is the interlink pointer to the most recent $\mu$-level ancestor of $b$.

A non-smooth pointer may not point to the most recent ancestor of level $\mu$ or even point to a superblock of a fork chain, as shown in Figure 2.

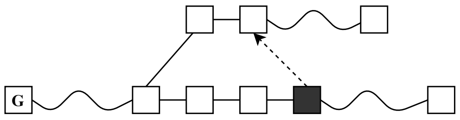


**Figure 2: A non-smooth pointer of an adversarial block, colored black, in an honest player's chain.**

In the same manner it is possible that a false interlink contain arbitrary pointers to blocks of any chain as illustrated in Figure 3. The interlink pointing to arbitrary directions resembles a thorny bush, so we will refer to blocks containing false interlink information as *thorny.*

*Definition 3.3 (Thorny Block).* Thorny block is a block which contains at least one non-smooth interlink pointer.

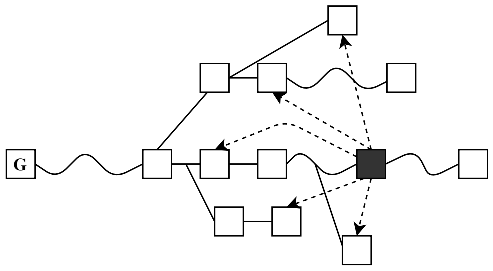


**Figure 3: A thorny block appended in an honest player's chain. The dashed arrows are interlink pointers.**

Opposite of the thorny are the *smooth* blocks, which may be blocks generated by non-upgraded players or blocks generated by upgraded players and contain only smooth pointers in their interlink.

*Definition 3.4 (Smooth Block).* Smooth block is any block which is not thorny.

## 4 THE CHAINSEWING ATTACK

We will now describe an explicit attack against the NIPoPoW suffix proof construction under velvet fork. As already argued, since the protocol is implemented under velvet fork, any thorny block be accepted as valid. Taking advantage of such blocks in the chain, the adversary could produce suffix proofs containing an arbitrary number of blocks belonging in several fork chains. The attack is described in detail in the following.

Assume that chain $C_B$ was adopted by an honest player B and chain $C_{\mathcal{A}}$, a fork of $C_B$ at some point, maintained by adversary $\mathcal{A}$. Assume that the adversary wants to produce a suffix proof in order to attack a light client to have him adopt a chain which contains blocks of $C_{\mathcal{A}}$. In order to achieve this, the adversary needs to include a greater amount of total proof-of-work in her suffix proof, $\pi_{\mathcal{A}}$, in comparison to that included in the honest player's proof, $\pi_B$, so as to achieve $\pi_{\mathcal{A}} \geq_m \pi_B$. For this she produces some thorny blocks in chains $C_{\mathcal{A}}$ and $C_B$ which will allow her to claim blocks of chain $C_B$ as if they were of chain $C_{\mathcal{A}}$ in her suffix proof.

The general form of this attack for an adversary sewing blocks to one forked chain is illustrated in Figure 4. Dashed arrows represent interlink pointers of some level $\mu_{\mathcal{A}}$. Starting from a thorny block in the adversary's forked chain and following the interlink pointers, a chain is formed which consists the adversary's suffix proof. Blocks of both chains are included in this proof and a verifier could not distinguish the non-smooth pointers participating in this proof chain and, as a result, would consider it a valid proof.

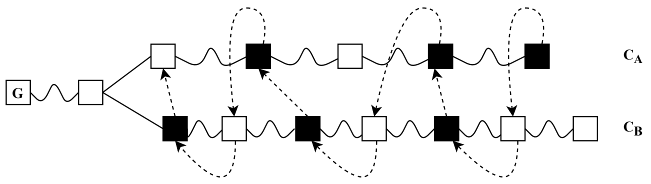


**Figure 4:** ***Generic Chainsewing Attack.*** $C_B$ ***is the chain of an honest player and*** $C_{\mathcal{A}}$ ***the adversary's chain. Adversarially generated blocks are colored black. Dashed arrows represent interlink pointers included in the adversary's suffix proof. Wavy lines imply one or more blocks.***

As the generic attack scheme may seem a bit complicated we will now describe a more specific attack case. Consider that the adversary acts as described below. Assume that the adversary chooses to attack at some level $\mu_{\mathcal{A}}$. As shown in Figure 5 she first generates a superblock $b'$ in her forked chain $C_{\mathcal{A}}$ and a thorny block $a'$ in the honest chain $C_B$ which points to $b'$. As argued earlier, block $a'$ will be accepted as valid in the honest chain $C_B$ despite the invalid interlink pointers. After that, the adversary may mine on chain $C_{\mathcal{A}}$ or $C_B$, or not mine at all. At some point she produces a thorny block $a$ in $C_{\mathcal{A}}$ pointing to a block $b$ of $C_B$. Because of the way blocks are generated by updated honest miners there will be successive interlink pointers leading from block $b$ to block $a'$. Thus following the interlink pointers a chain is formulated which connects $C_{\mathcal{A}}$ blocks $a$ and $b'$ and contains an arbitrarily large part of the honest player's chain $C_B$.

At this point the adversary will produce a suffix proof for chain $C_{\mathcal{A}}$ containing the subchain $C\{ab\} \cup C\{b{:}a'\} \cup C\{a'{:}b'\}$. Notice that following the interlink pointers constructed in such a way, a light client perceives $C\{ab\} \cup C\{b{:}a'\} \cup C\{a'{:}b'\}$ as a valid chain.

In this attack the adversary uses thorny blocks to "sew" portions of the chain adopted by an honest player to her own forked chain. This remark justifies the name given to the attack.

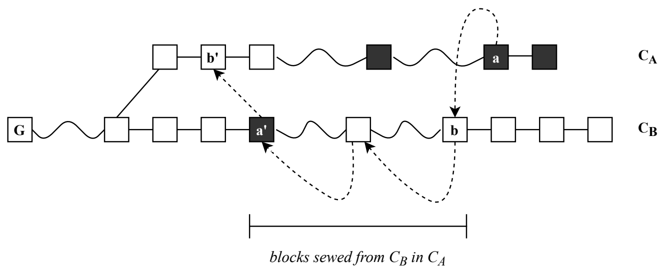


**Figure 5: *Chainsewing Attack.* $C_B$ *represents the chain of an honest player.* $C_{\mathcal{A}}$ *is an adversarial fork. Adversarially generated blocks are colored black. Dashed arrows represent interlink pointers included in the adversary's suffix proof. Wavy lines imply one or more blocks. Firm lines imply the previousId relationship between two sequential blocks.***

Note that in order to make this attack successful, the adversary has to produce only a few superblocks which let her arrogate an arbitrarily large number of blocks. Thus this attack is expected to succeed with overwhelming probability.

## 5 VELVET NIPOPOWS

In order to eliminate the Chainsewing Attack we propose an update to the velvet NIPoPoW protocol. The core problem is that in her suffix proof the adversary is able to claim not only blocks of forked chains, which are in majority adversarially generated due to the Common Prefix property, but also arbitrarily long parts of the chain adopted by an honest player. Since thorny blocks are accepted as valid, the verifier cannot distinguish blocks that actually belong in a chain from blocks that only seem to belong in the same chain because they are pointed to via a non-smooth pointer.

We describe a protocol patch that operates as follows. The NIPoPoW protocol under velvet fork works as usual but each miner constructs smooth blocks. This means that a block's interlink is constructed excluding thorny blocks. In this way, although thorny blocks are accepted in the chain, they are not taken into consideration when updating the interlink structure for the next block to be mined. No honest block could now point to a thorny superblock that may act as the passing point to the fork chain in an adversarial suffix proof. Thus, after this protocol update the adversary is only able to inject adversarially generated blocks from an honestly adopted chain to her own fork. At the same time, thorny blocks cannot participate in an honestly generated suffix proof except for some blocks in the proof's suffix ($\chi$). This argument holds because thorny blocks do not form a valid chain along with honestly mined blocks anymore. Consequently, as far as the blocks included in a suffix proof is concerned, we can think of thorny blocks as belonging in the adversary's fork chain for the $\pi$ part of the proof, which is the comparing part between proofs. Figure 6 illustrates this remark.

In order to make NIPoPoWs a secure protocol under velvet fork conditions we suggest that the NIPoPoW protocol under velvet fork works as usual but a miner constructs a block's interlink without pointers to thorny blocks.

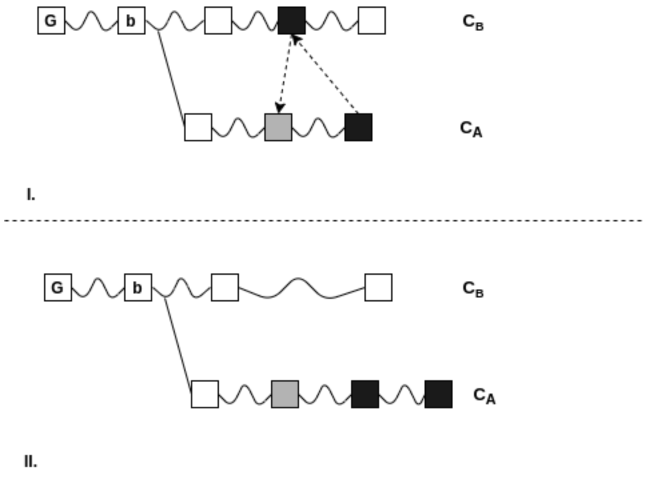


**Figure 6: The adversarial fork chain $C_A$ and chain $C_B$ of an honest player. Thorny blocks are colored black. Dashed arrows represent interlink pointers. Wavy lines imply one or more blocks. After the protocol update, when an adversarially generated block is sewed from $C_B$ into the adversary's suffix proof the verifier conceives $C_A$ as longer and $C_B$ as shorter. I: The real picture of the chains. II: Equivalent picture from the verifier's perspective considering the blocks included in the corresponding suffix proof for each chain.**

In order to prove the security of the updated protocol we need to strengthen the Honest Majority Assumption to 1/4 adversary with respect to upgraded honest miners.

*Definition 5.1 (Velvet Honest Majority). Let $n_h$ be the number of upgraded honest miners. Then $t$ out of total $n$ parties are corrupted such that $\frac{t}{n_h} < \frac{1-\delta_v}{3}$.*

The following Lemmas come as immediate results from the suggested protocol update.

LEMMA 5.2. *A velvet suffix proof constructed by an honest player cannot contain any thorny block.*

LEMMA 5.3. *Let $\mathcal{P}_{\mathcal{A}} = (\pi_{\mathcal{A}}, \chi_{\mathcal{A}})$ be a velvet suffix proof constructed by the adversary and block $b_s$, generated at round $r_s$, be the most recent smooth block in the proof. Then $\forall r : r < r_s$ no thorny blocks generated at round $r$ can be included in $\mathcal{P}_{\mathcal{A}}$.*

PROOF. By contradiction. Let $b_t$ be a thorny block generated at some round $r_t < r_s$. Suppose for contradiction that $b_t$ is included in the proof. Then, because $\mathcal{P}_{\mathcal{A}}$ is a valid chain as for interlink pointers, there exist a block path made by interlink pointers starting from $b_s$ and resulting to $b_t$. Let $b'$ be the most recently generated thorny block after $b_t$ and before $b_s$ included in $\mathcal{P}_{\mathcal{A}}$. Then $b'$ has been generated at a round $r'$ such that $r_t \leq r' < r_s$. Then the block right after block $b'$ in $\mathcal{P}_{\mathcal{A}}$ must be a thorny block since it points to $b'$ which is thorny. But $b'$ is the most recent thorny block after $b_t$, thus we have reached a contradiction. □

Lemma 5.4. *Let $\mathcal{P}_{\mathcal{A}} = (\pi_{\mathcal{A}}, \chi_{\mathcal{A}})$ be a velvet suffix proof constructed by the adversary. Let $b_t$ be the oldest thorny bock included in $\mathcal{P}_{\mathcal{A}}$ which is generated at round $r_t$. Then any block $b = \{b : b \in \mathcal{P}_{\mathcal{A}} \wedge b$ generated at $r \geq r_t\}$ is thorny.*

Proof. By contradiction. Suppose for contradiciton that $b_s$ is a smooth block generated at round $r_s > r_t$. Then from Lemma 5.3 any block generated at round $r < r_s$ is smooth. But $b_t$ is generated at round $r_t < r_s$ and is thorny, thus we have reached a contradiction. □

The following corollary emerges immediately from Lemmas 5.3, 5.4. This result is illustrated in Figure 7.

Corollary 5.5. *Any adversarial proof $\mathcal{P}_{\mathcal{A}} = (\pi_{\mathcal{A}}, \chi_{\mathcal{A}})$ containing both smooth and thorny blocks consists of a prefix smooth subchain followed by a suffix thorny subchain.*

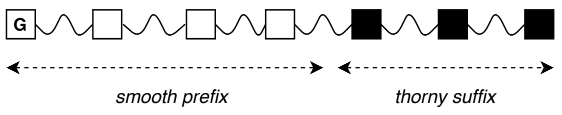


**Figure 7: In the general case the adversarial velvet suffix proof $\mathcal{P}_{\mathcal{A}} = (\pi_{\mathcal{A}}, \chi_{\mathcal{A}})$ consists of an initial part of smooth blocks followed by thorny blocks.**

We now describe the algorithms needed by the upgraded miner, prover and verifier. The upgraded miner acts as usual except for including the interlink of the newborn block in the coinbase transaction. In order to construct an interlink containing only the smooth blocks, the miner keeps a copy of the "smooth chain" ($C_S$) which consists of the smooth blocks existing in the original chain C. The algorithm for extracting the smooth chain out of C is given in Algorithm 3. Function *isSmoothBlock(B)* checks whether a block $B$ is a smooth velvet by calling *isSmoothPointer(B,p)* for every pointer $p$ in $B$'s interlink. Function *isSmoothPointer(B,p)* returns *true* if $p$ is a valid pointer, in essence a pointer to the most recent *smooth velvet* for the level denoted by the pointer itself. The *updateInterlink* algorithm is given in Algorithm 4, which is essentially the same as in the case of a hard/soft fork, except for working on the smooth chain $C_S$ instead of C.

The construction of the velvet suffix prover is given in Algorithm 5, which is essentially the same to that of a hard/soft fork except for working on smooth chain $C_S$ instead of C.

In conclusion the Verify algorithm for the NIPoPoW suffix protocol remains the same as in the case of hard or soft fork.

**Algorithm 3** Smooth chain for suffix proofs

```
function smoothChain(C)
    $C_S \leftarrow \{\mathcal{G}\}$
    $k \leftarrow 1$
    while $C[-k] \neq \mathcal{G}$ do
        if isSmoothBlock($C[-k]$) then
            $C_S \leftarrow C_S \cup C[-k]$
        end if
        $k \leftarrow k + 1$
    end while
    return $C_S$
end function

function isSmoothBlock($B$)
    if $B = \mathcal{G}$ then
        return true
    end if
    for $p \in B$.interlink do
        if $\neg$isSmoothPointer($B, p$) then
            return false
        end if
    end for
    return true
end function

function isSmoothPointer($B, p$)
    $b \leftarrow$ Block($B$.prevId)
    while $b \neq p$ do
        if level($b$) $\geq$ level($p$) $\wedge$ isSmoothBlock($b$) then
            return false
        end if
        if $b = \mathcal{G}$ then
            return false
        end if
        $b \leftarrow$ Block($b$.prevId)
    end while
    return isSmoothBlock($b$)
end function
```

**Algorithm 4** Velvet updateInterlink

```
function updateInterlinkVelvet($C_S$)
    $B' \leftarrow C_S[-1]$
    interlink $\leftarrow B'$.interlink
    for $\mu = 0$ to level($B'$) do
        interlink[$\mu$] $\leftarrow$ id($B'$)
    end for
    return interlink
end function
```

## 6 COMBINED ATTACK

After the suggested protocol update the honest prover cannot contain any thorny blokcs in his suffix NIPoPow even if these blocks

**Algorithm 5** Velvet Suffix Prover

```
function ProveVelvet$_{m,k}$(C$_S$)
    $B \leftarrow \mathsf{C}_S[0]$
    for $\mu = |\mathsf{C}_S[-k].\text{interlink}|$ down to 0 do
        $\alpha \leftarrow \mathsf{C}_S[:-k]\{B:\}\uparrow^\mu$
        $\pi \leftarrow \pi \cup \alpha$
        $B \leftarrow \alpha[-m]$
    end for
    $\chi \leftarrow \mathsf{C}_S[-k:]$
    return $\pi\chi$
end function
```

are part of $\mathsf{C}_B$. The adversary may exploit this fact and to try to suppress honestly generated blocks in $\mathsf{C}_B$, in order to reduce the blocks that can represent the honest chain in a proof. In parallel, while the adversary mines suppressive thorny blocks on $\mathsf{C}_B$ she can still use her blocks in her NIPoPoW proofs, by chainsewing them. Consequently, even if a suppression attempt does not succeed, in case for example that a second honestly generated block is soon enough published, she does not drop the thorny block she generated but include it in her proof.

More in detail, consider that the adversary wishes to attack a specific block level $\mu_B$ and generate a NIPoPow proof containing a block $b$ of a fork chain which contains a double spending transaction. Then she acts as follows. She may mine on her fork chain $\mathsf{C}_{\mathcal{A}}$ but when she observes a $\mu_B$-level block in $\mathsf{C}_B$ she mines a thorny block on $\mathsf{C}_B$ which jumps onto her fork chain, in order to suppress this $\mu_B$ block. If the suppression succeeds she has managed to damage the $\mu_B$ superchain and mine a block that she can afterwards use in her proof. If the suppression does not succeed she can still use the thorny in her proof. The above are illustrated in Figure 8.

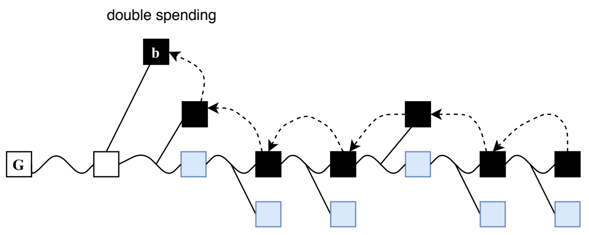


**Figure 8: *The adversary suppress honestly generated blocks and chainsew thorny blocks in $\mathsf{C}_B$. Blue blocks are honestly generated blocks of a specific level of attack. The adversary tries to suppress them. If the suppression is not successful, the adversary can still use the block she mined in her proof.***

The described attack consists a combined attack since both suppression and chainsewing are utilized. This combined attack forces us to adopt a stronger Honest Majority Assumption, so as to guarantee that the unsuppressed blocks in $\mathsf{C}_B$ suffice for constructing winning NIPoPoW proofs against the adversarial ones.

## 7 ANALYSIS

*Definition 7.1 (block property, Q-block).* [29] A *block property* is a predicate $Q$ defined on a hash output $h \in \{0,1\}^\kappa$. Given a block property $Q$, a valid block with hash h is called a $Q$-block if $Q(h)$ is true.

Lemma 7.2 (Unsuppressibility). *[29] Consider a collection of polynomially many block properties $\mathcal{Q}$. In a typical execution every set of consecutive rounds U has a subset S of uniquely successful rounds such that*

- $|S| \geq Y(U) - 2Z(U) - 2\lambda f(\frac{t}{n-t} \cdot \frac{1}{1-f} + \epsilon)$
- *for any $Q \in \mathcal{Q}$, Q-blocks generated during S follow the distribution as in an unsuppressed chain*
- *after the last round in S the blocks corresponding to S belong to the chain of any honest party.*

Lemma 7.3. *Consider Algorithm 4 under velvet fork with parameter g and (1/4)-bounded velvet honest majority. Let U be a set of consecutive rounds $r_1 \cdots r_2$ and* C *the chain of an honest party at round $r_2$ of a typical execution. Let $\mathsf{C}_U^S = \{b \in \mathsf{C} : b \text{ is smooth} \wedge b \text{ was generated during } U\}$. Let $\mu, \mu' \in \mathbb{N}$. Suppose $\mathsf{C}'$ a $\mu'$ superchain containing only adversarial blocks generated during U and suppose $|\mathsf{C}'| > k$. Then it holds that $2^{\mu'}|\mathsf{C}'| < 2^\mu|\mathsf{C}_U^S\uparrow^\mu|$.*

Proof. From the Unsuppressibility Lemma we have that there is a set of uniquely successful rounds $S$, subset of U, such that $|S| \geq Y(U) - 2Z(U) - \delta'$, where $\delta' = 2\lambda f(\frac{t}{n-t} \cdot \frac{1}{1-f} + \epsilon)$. We also know that $Q$-blocks of the property collection $\mathcal{Q} = \{Q(h) = true : h \in \{0,1\}^\kappa, \widetilde{\mu} \leq log(|\mathsf{C}|), h \leq 2^{-\widetilde{\mu}}T\}$ that were generated during $S$ are distributed as in an unsuppressed chain. Therefore for the number of interlinked $\mu$-blocks that were generated during $S$ it holds that $|\mathsf{C}_U^S\uparrow^\mu| \geq (1-\epsilon)g2^{-\mu}|S|$. For the total number of $\mu'$-blocks the adversary generated during $U$ it holds that $|\mathsf{C}'| \leq (1+\epsilon)2^{-\mu'}Z(U)$. Then we have to show that $(1-\epsilon)g(Y(U) - 2Z(U) - \delta') > (1+\epsilon)Z(U)$ or $((1+\epsilon) + 2g(1-\epsilon))Z(U) < g(1-\epsilon)(Y(U)+\delta')$. Let $\alpha_z = (1+\epsilon) + 2g(1-\epsilon)$ then we have $\alpha_z Z(U) < g(1-\epsilon)(Y(U)+\delta')$.

Because of the typical exevution we know that $Y(U), Z(U)$ cannot deviate much from their expected values. In particular, we have that[8]: $Z(U) < \mathbb{E}[Z(U)] + \epsilon\mathbb{E}[X(U)]$ and $Y(U) > (1-\epsilon)\mathbb{E}[Y(U)]$. From the Backbone analysis we also know that $\mathbb{E}[Z(U)] < (1+\frac{\delta}{2})f\frac{t}{n-t}|U|$, $\mathbb{E}[X(U)] < pq(n-t)|U|$ and $\mathbb{E}[Y(U)] > f(1-f)|U|$. By substituting the expectation values we have $\alpha_z[(1+\frac{\delta}{2})f\frac{t}{n-t}|U| + \epsilon pq(n-t)|U|] < (1-\epsilon)g[(1-\epsilon)f(1-f)|U| - \delta']$ or $\alpha_z f|U|\frac{t}{n-t}(1+\frac{\delta}{2}) + \alpha_z\epsilon pq(n-t)|U| < (1-\epsilon)g[(1-\epsilon)f(1-f)|U| - \delta']$ or

$$\frac{t}{n-t} < \frac{(1-\epsilon)g[(1-\epsilon)f(1-f)|U| - \delta'] - \alpha_z\epsilon pq(n-t)|U|}{f|U|\alpha_z(1+\frac{\delta}{2})}$$

or

$$\frac{t}{n-t} < \frac{(1-\epsilon)g[(1-\epsilon)f(1-f) - \frac{\delta'}{|U|}] - \alpha_z\epsilon pq(n-t)}{f\alpha_z(1+\frac{\delta}{2})}$$

But

$$\frac{\epsilon pq(n-t)}{f(1+\frac{\delta}{2})} = \epsilon' \ll 1$$

thus

$$\frac{t}{n-t} < \frac{(1-\epsilon)g[(1-\epsilon)f(1-f) - \frac{\delta'}{|U|}]}{f\alpha_z(1+\frac{\delta}{2})} - \epsilon'$$

or

$$\frac{t}{n-t} < \frac{g}{(1+\frac{\delta}{2})f\alpha_z} \cdot \frac{(1-\epsilon)^2 f(1-f) - \frac{(1-\epsilon)\delta'}{|U|}}{(1+\frac{\delta}{2})f\alpha_z} - \epsilon'$$

Consequently we have that $\frac{t}{n-t} < \frac{1-\delta_v}{3}g$ which is the (1/4) velvet honest majority assumption, since

$$\frac{1}{(1+\frac{\delta}{2})f\alpha_z} = \frac{1}{(1+\frac{\delta}{2})f(1+\epsilon) + (1+\frac{\delta}{2})f(1-\epsilon)2g} > \frac{1}{3}$$

and

$$\frac{(1-\epsilon)^2 f(1-f) - \frac{(1-\epsilon)\delta'}{|U|}}{(1+\frac{\delta}{2})f\alpha_z} > (1-\delta_v)$$

. The last holds because we have $|\mathsf{C}'| > k$ so $|U| \gg (1-\epsilon)\delta'$. ☐

Theorem 7.4 (Suffix Proofs Security under velvet fork). *Assuming honest majority under velvet fork conditions (5.1) such that $t \leq (1-\delta_v)\frac{n_h}{3}$ where $n_h$ the number of upgraded honest players, the non-interactive proofs-of-proof-of-work construction for computable k-stable monotonic suffix-sensitive predicates under velvet fork conditions in a typical execution is secure.*

Proof. By contradiction. Let $Q$ be a k-stable monotonic suffix-sensitive chain predicate. Assume for contradiction that NIPoPoWs under velvet fork on $Q$ is insecure. Then, during an execution at some round $r_3$, $Q(\mathsf{C})$ is defined and the verifier $V$ disagrees with some honest participant. $V$ communicates with adversary $\mathcal{A}$ and honest prover $B$. The verifier receives proofs $\pi_{\mathcal{A}}, \pi_B$ which are of valid structure. Because $B$ is honest, $\pi_B$ is a proof constructed based on underlying blockchain $\mathsf{C}_B$ (with $\pi_B \subseteq \mathsf{C}_B$), which $B$ has adopted during round $r_3$ at which $\pi_B$ was generated. Consider $\widetilde{\mathsf{C}}_{\mathcal{A}}$ the set of blocks defined as $\widetilde{C}_{\mathcal{A}} = \pi_{\mathcal{A}} \cup \{\bigcup\{\mathsf{C}_h^r\{:b_{\mathcal{A}}\} : b_{\mathcal{A}} \in \pi_{\mathcal{A}}, \exists h, r : b_{\mathcal{A}} \in \mathsf{C}_h^r\}\}$ where $\mathsf{C}_h^r$ the chain that the honest player $h$ has at round $r$.

The verifier outputs $\neg Q(\mathsf{C}_B)$. Thus it is necessary that $\pi_{\mathcal{A}} \geq_m \pi_B$. We show that $\pi_{\mathcal{A}} \geq_m \pi_B$ is a negligible event.

Let the levels of comparison decided by the verifier be $\mu_{\mathcal{A}}$ and $\mu_B$ respectively. Let $b_0 = LCA(\pi_{\mathcal{A}}, \pi_B)$. Let $\mu_B'$ be the adequate level of proof $\pi_B$ with respect to block $b_0$. Call $\alpha_{\mathcal{A}} = \pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}} \{b_0{:}\}$, $\alpha_B' = \pi_B\uparrow^{\mu_B'} \{b_0{:}\}$.

From Corollary 5.5 we have that the adversarial proof consists of a smooth interlink subchain followed by a thorny interlink subchain. We will refer to the smooth part of $\alpha_{\mathcal{A}}$ as $\alpha_{\mathcal{A}}^{\mathcal{S}}$ and to the thorny part as $\alpha_{\mathcal{A}}^{\mathcal{T}}$.

Our proof construction is based on the following intuition: we consider that $\alpha_{\mathcal{A}}$ consists of three distinct parts $\alpha_{\mathcal{A}}^1, \alpha_{\mathcal{A}}^2, \alpha_{\mathcal{A}}^3$ with the following properties.

Consider $b_0 = LCA(\pi_{\mathcal{A}}, \pi_B)$ the fork point between $\pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}$, $\pi_B\uparrow^{\mu_B}$ and $b_1 = LCA(\alpha_{\mathcal{A}}^{\mathcal{S}}, \mathsf{C}_B)$ the fork point between $\pi_{\mathcal{A}}^{\mathcal{S}}\uparrow^{\mu_{\mathcal{A}}}$, $\mathsf{C}_B$ at the zero level as the honest prover could observe. Part $\alpha_{\mathcal{A}}^1$ contains the blocks between $b_0$ exclusive and $b_1$ inclusive generated during the set of consecutive rounds $\mathcal{S}_1$ and $|\alpha_{\mathcal{A}}^1| = k_1$. Consider $b_2$ the last block in $\alpha_{\mathcal{A}}$ generated by an honest player. Part $\alpha_{\mathcal{A}}^2$ contains the blocks between $b_1$ exclusive and $b_2$ inclusive generated during the set of consecutive rounds $\mathcal{S}_2$ and $|\alpha_{\mathcal{A}}^2| = k_2$. Consider $b_3$ the next block of $b_2$ in $\alpha_{\mathcal{A}}$. Then $\alpha_{\mathcal{A}}^3 = \alpha_{\mathcal{A}}[b_3{:}]$ and $|\alpha_{\mathcal{A}}^3| = k_3$ of all adversarially generated blocks generated during the set of rounds $\mathcal{S}_3$. So, $|\alpha_{\mathcal{A}}| = k_1 + k_2 + k_3$ and we will show that $|\alpha_{\mathcal{A}}| < |\alpha_B|$ .

The above are illustrated, among other, in Parts I, II of Figure 9.

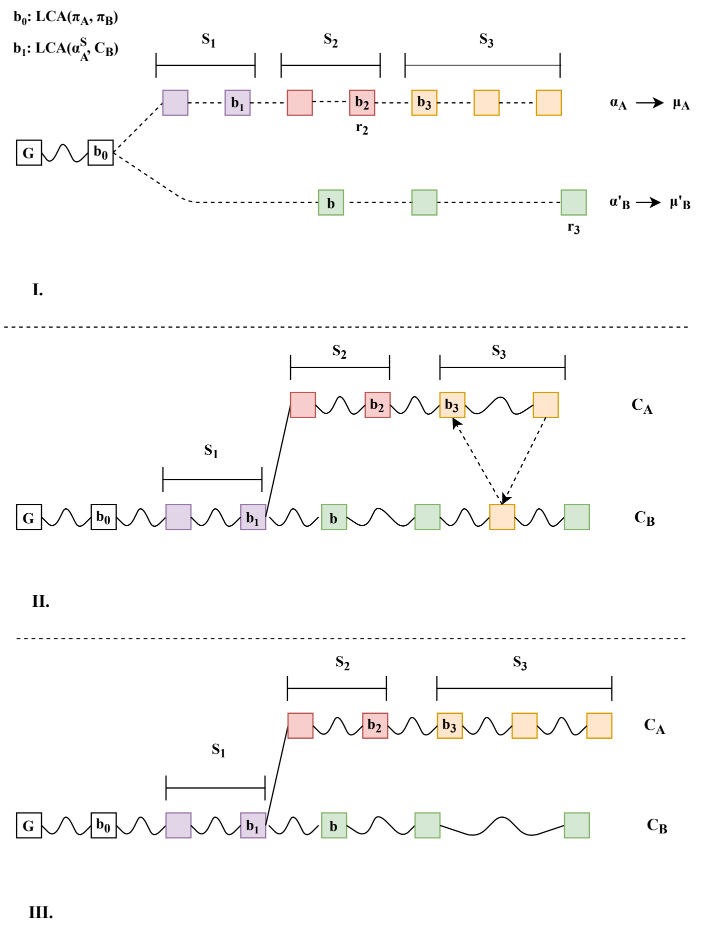


**Figure 9: *Wavy lines imply one or more blocks. Dashed lines and arrows imply interlink pointers to superblocks. I: the three round sets in two competing proofs at different levels, II: the corresponding 0-level blocks implied by the two proofs, III: blocks participating in chain $\mathsf{C}_B$ and block set $\widetilde{\mathsf{C}}_{\mathcal{A}}$ from the verifier's perspective.***

We will now show three successive claims under velvet fork conditions: First that $\alpha_{\mathcal{A}}^1$ contains only a few blocks. Second, $\alpha_{\mathcal{A}}^2$

contains only a few blocks. And third, the adversary is able to produce a winning $a_{\mathcal{A}}$ with negligible probability.

**Claim 1:** $\alpha^1_{\mathcal{A}} = \alpha_{\mathcal{A}}(b_0 : b_1]$ contains only a few blocks. We have defined $b_0 = LCA(\pi_{\mathcal{A}}, \pi_B)$ and $b_1 = LCA(\alpha^S_{\mathcal{A}}, \alpha'_B \downarrow)$. First observe that there are no thorny blocks in $\alpha^1_{\mathcal{A}}$ since $\alpha^1_{\mathcal{A}}[-1] = b_1$ is a smooth block. This means that if $b_1$ was generated at round $r_{b_1}$ and $\alpha^S_{\mathcal{A}}[-1]$ in round $r$, then $r \geq r_{b_1}$. So, $\alpha^1_{\mathcal{A}}$ consists a valid chain. We show the statement considering the two possible cases for the relation of $\mu_{\mathcal{A}}, \mu'_B$.

<u>*Claim 1a:*</u> If $\mu'_B \leq \mu_A$ then they are completely disjoint. In such a case of inequality, every block in $\alpha_A$ would also be of lower level $\mu'_B$. Because of the adequate level $\mu'_B$ we know that $C\{b:\}\uparrow^{\mu'_B} = \pi\{b:\}\uparrow^{\mu'_B}$[15]. Subsequently, any block in $\pi_A\uparrow^{\mu_A}$ $\{b:\}[1:]$ would also be included in proof $\alpha'_B$, but $b = LCA(\pi_A, \pi_B)$ so there can be no succeeding block common in $\alpha_A, \alpha'_B$.

<u>*Claim 1b*</u>: If $\mu'_B > \mu_A$ then $|\alpha_A[1:] \cap \alpha'_B \downarrow [1:]| = k_1 \leq g(2^{\mu'_B - \mu_A})$. Let's call $b$ the first block in $\alpha'_B$ after block $b_0$. Suppose for contradiction that $k_1 > g(2^{\mu'_B - \mu_A})$. Since block $b$ of level $\mu'_B$ is also of level $\mu_A$, the adversary could include it in the proof but $b$ cannot exist in both $\alpha_A, \alpha'_B$ since $\alpha_A \cap \alpha'_B = \emptyset$ by definition. In case that the adversary chooses not to include $b$ in the proof then she can include no other blocks of $C_B$ in her proof, since it would not consist a valid chain. Therefore, the adversary can include at most the $\mu_{\mathcal{A}}$ upgraded blocks between $b_0, b$, which are expected to be equal to $g(2^{\mu'_B - \mu_A})$.

We conclude that $|\alpha^S_{\mathcal{A}} \cap \alpha'_B \downarrow [1:]| = k_1 \leq g(2^{\mu'_B - \mu_{\mathcal{A}}})$, where $g$ the velvet parameter denoting the percentage of upgraded honest parties.

Consequently, there are at least $|\alpha_{\mathcal{A}}| - k_1$ blocks after block $b$ in $\alpha_{\mathcal{A}}$ which are not honestly generated blocks existing in $C_B$. In other words, there are $|\alpha_{\mathcal{A}}| - k_1$ blocks after block $b$ in $\alpha_{\mathcal{A}}$, which are either thorny blocks existing in $C_B$ either don't belong in $C_B$.

**Claim 2.** Part $\alpha^2_{\mathcal{A}} = \alpha_{\mathcal{A}}(b_1 : b_2]$ consists of only a few blocks. Let $|\alpha^2_{\mathcal{A}}| = k_2$. We have defined $b_2 = \alpha^2_{\mathcal{A}}[-1]$ to be the last block generated by an honest player in $\alpha_{\mathcal{A}}$. Consequently no thorny block exists in $\alpha^2_{\mathcal{A}}$, so all blocks in this part belong in a proper zero-level chain $C^2_{\mathcal{A}}$. Let $r_{b_1}$ be the round at which $b_1$ was generated. Since $b_1$ is the last block in $\alpha_{\mathcal{A}}$ which belongs in $C_B$, then $C^2_{\mathcal{A}}$ is a fork chain to $C_B$ at some block $b'$ generated at round $r' \geq r_{b_1}$.

Let $r_2$ be the round when $b_2$ was generated by an honest party. Because an honest party has chain $C_B$ at later round $r_3$ when the proof $\pi_B$ is constructed and because of the Common Prefix property on parameter $k_{2\downarrow} = g \cdot 2^{\mu_{\mathcal{A}}} \cdot k_2$, we conclude that $k_2 \leq k$.

**Claim 3.** The adversary may submit a suffix proof such that $|\alpha_{\mathcal{A}}| \geq |\alpha_B|$ with negligible probability. As explained earlier part $\alpha^3_{\mathcal{A}}$ consists only of adversarially generated blocks. The security parameter $m$ guarantees that $|\alpha^3_{\mathcal{A}}| > k$, as will be later presented in detail. Let $U$ be the set of consecutive rounds $r_2...r_3$. Then all $k_3$ blocks of this part of the proof are generated during $U$. Let $\alpha'^{S_3}_B$ be the last part of the honest proof containing the interlinked $\mu_B$ superblocks generated during $U$. Then from lemma 7.3 we have that $2^{\mu_{\mathcal{A}}}|\alpha^3_{\mathcal{A}}| < 2^{\mu_B}|\alpha'^U_B \uparrow^{\mu_B}|$.

From all the above Claims we have that:
For the first round set $S_1$, because of the common underlying chain:

$$2^{\mu_{\mathcal{A}}}|\alpha^{S_1}_{\mathcal{A}}| \leq 2^{\mu'_B}|\alpha'^{S_1}_B| \quad (1)$$

For the second round set $S_2$ because of the adoption by an honest party of chain $C_B$ at a later round $r_3$ we have:

$$2^{\mu_{\mathcal{A}}}|\alpha^{S_2}_{\mathcal{A}}| \leq 2^{\mu'_B}|\alpha'^{S_2}_B| \quad (2)$$

For the third round set $S_3$ we have:

$$2^{\mu_{\mathcal{A}}}|\alpha^{S_3}_{\mathcal{A}}| < 2^{\mu'_B}|\alpha'^{S_3}_B| \quad (3)$$

Consequently we have:

$$2^{\mu_{\mathcal{A}}}(|\alpha^{S_1}_{\mathcal{A}}| + |\alpha^{S_2}_{\mathcal{A}}| + |\alpha^{S_3}_{\mathcal{A}}|) < 2^{\mu'_B}(|\alpha'^{S_1}_B| + |\alpha'^{S_2}_B| + |\alpha'^{S_3}_B|) \Rightarrow$$

$$2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}| < 2^{\mu'_B}|\alpha'_B| \quad (4)$$

Therefore we have proven that $2^{\mu'_B}|\pi_B\uparrow^{\mu'_B}| > 2^{\mu_{\mathcal{A}}}|\pi^{\mu_{\mathcal{A}}}_{\mathcal{A}}|$. From the definition of $\mu_B$, we know that $2^{\mu_B}|\pi_B\uparrow^{\mu_B}| > 2^{\mu'_B}|\pi_B\uparrow^{\mu'_B}|$ because it was chosen $\mu_B$ as level of comparison by the Verifier. So we conclude that $2^{\mu_B}|\pi_B\uparrow^{\mu_B}| > 2^{\mu_{\mathcal{A}}}|\pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}|$. □

## 8 INFIX PROOFS

NIPoPoWs infix proofs are proofs that answer any predicate which depends on blocks appearing anywhere in the chain, except for the $k$ suffix for stability reasons. For example, consider the case where a client has received a transaction inclusion proof for a block $b$ and requests an infix proof so as to verify that $b$ is included in the current chain.

Because of the described protocol update for secure NIPoPoW suffix proofs, the infix proofs construction has to be altered as well. In order to construct secure infix proofs under velvet fork conditions, we suggest the following additional protocol patch: each updated miner constructs and updates an authenticated data structure for all the blocks in the chain. Our suggestion for this structure is Merkle Mountain Range *(MMR).* Now a block's header additionally includes the root of this MMR. An updated prover also keeps an updated blocks' MMR locally. Smooth blocks are now considered the blocks that contain smooth interlinks but also valid MMR root. A valid MMR root denotes the MMR that contains all the blocks in the chain of an honest full node.

Considering this addtional patch we can now define the final algorithms for the honest miner, infix and suffix prover, as well as for the infix verifier. Because of the new definition of a smooth block, Algorithm 3 needs to be updated, so as to also check for the validity of the included MMR. The complete algorithm for the computation of the smooth chain is given in Algorithm 6 . Now considering that input $C_S$ is computed with Algorithm , *Velvet updateInterlink* and *Velvet Suffix Prover* algorithms need no changes and are described in Algorithms 4, 5 repsectively. The velvet infix prover and infix verifier algorithms are given in Algorithms 7, 8 respectively.

**Algorithm 6** Smooth chain for suffix and infix proofs

```
function smoothChain(C)
    $C_S \leftarrow \{\mathcal{G}\}$
    $k \leftarrow 1$
    while $C[-k] \neq \mathcal{G}$ do
        if isSmoothBlock($C[-k]$) then
            $C_S \leftarrow C_S \cup C[-k]$
        end if
        $k \leftarrow k + 1$
    end while
    return $C_S$
end function

function isSmoothBlock($B$)
    if $B = \mathcal{G}$ then
        return true
    end if
    for $p \in B$.interlink do
        if $\neg$isSmoothPointer($B, p$) then
            return false
        end if
    end for
    return containsValidMMR($B$)
end function

function isSmoothPointer($B, p$)
    $b \leftarrow$ Block($B$.prevId)
    while $b \neq p$ do
        if level($b$) $\geq$ level($p$) $\wedge$ isSmoothBlock($b$) then
            return false
        end if
        if $b = \mathcal{G}$ then
            return false
        end if
        $b \leftarrow$ Block($b$.prevId)
    end while
    return isSmoothBlock($b$)
end function
```

**Algorithm 7** Velvet Infix Prover

```
function ProveInfixVelvet($b$)
    $(\pi, \chi) \leftarrow$ ProveVelvet($C_S$)
    tip $\leftarrow \pi[-1]$
    $\pi_b \leftarrow$ MMRinclusionProof($tip, b$)
    return $(\pi_b, (\pi, \chi))$
end function
```

**Algorithm 8** Velvet Infix Verifier

```
function VerifyInfixVelvet($b, (\pi_b, (\pi, \chi))$)
    tip $\leftarrow \pi[-1]$
    return VerifyInclProof(tip.$root_{MMR}$, $\pi_b$, $b$)
end function
```

Note that equivalent solution could be formed by using any authenticated data structure that provides inclusion proofs of size logarithmic to the size of the chain. We choose MMRs because of their efficiency while updating, since they support efficient node addition and deletion functionality.

## REFERENCES

[1] Mihir Bellare and Phillip Rogaway. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*. ACM, 62–73.
[2] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. 2020. Flyclient: Super-Light Clients for Cryptocurrencies. (2020).
[3] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* (2014).
[4] Alexander Chepurnoy, Charalampos Papamanthou, and Yupeng Zhang. 2018. Edrax: A Cryptocurrency with Stateless Transaction Validation. *IACR Cryptology ePrint Archive* 2018 (2018), 968.
[5] Cynthia Dwork and Moni Naor. 1992. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*. Springer, 139–147.
[6] Ittay Eyal and Emin Gün Sirer. 2014. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*. Springer, 436–454.
[7] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*. Springer, 186–194.
[8] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2015), 281–310.
[9] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network.. In *USENIX Security Symposium*. 129–144.
[10] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. 2019. Compact Storage of Superblocks for NIPoPoW Applications. In *The 1st International Conference on Mathematical Research for Blockchain Economy*. Springer Nature.
[11] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. 2019. Proof-of-Burn. In *International Conference on Financial Cryptography and Data Security*.
[12] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. 2020. Smart Contract Derivatives. (2020).
[13] Aggelos Kiayias, Peter Gaži, and Dionysis Zindros. 2019. Proof-of-Stake Sidechains. In *IEEE Symposium on Security and Privacy*. IEEE, IEEE.
[14] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. 2016. Proofs of Proofs of Work with Sublinear Complexity. In *International Conference on Financial Cryptography and Data Security*. Springer, 61–78.
[15] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. 2020. Non-Interactive Proofs of Proof-of-Work. In *International Conference on Financial Cryptography and Data Security*. Springer.
[16] Aggelos Kiayias and Dionysis Zindros. 2019. Proof-of-Work Sidechains. In *International Conference on Financial Cryptography and Data Security*. Springer, Springer.
[17] Jae-Yun Kim, Jun-Mo Lee, Yeon-Jae Koo, Sang-Hyeon Park, and Soo-Mook Moon. 2019. Ethanos: Lightweight Bootstrapping for Ethereum. *arXiv preprint arXiv:1911.05953* (2019).
[18] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. RFC6962: Certificate Transparency. *Request for Comments. IETF* (2013).
[19] Eric Lombrozo, Johnson Lau, and Pieter Wuille. 2015. BIP 0141: Segregated witness (consensus layer). Available at: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki.
[20] Izaak Meckler and Evan Shapiro. 2018. CODA: Decentralized Cryptocurrency at Scale. (2018).
[21] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 369–378.
[22] Satoshi Nakamoto. 2009. Bitcoin: A peer-to-peer electronic cash system. (2009). http://www.bitcoin.org/bitcoin.pdf
[23] Claus-Peter Schnorr. 1989. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*. Springer, 239–252.
[24] Peter Todd. October 2012. Merkle Mountain Ranges. (October 2012). https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md.
[25] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014), 1–32.
[26] Karl Wüst and Arthur Gervais. 2016. *Ethereum eclipse attacks*. Technical Report. ETH Zurich.
[27] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt.

2019. SoK: Communication across distributed ledgers. (2019).

[28] Alexei Zamyatin, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Edgar Weippl, William Knottenbelt, and Alexei Zamyatin. 2018. A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice. In *International Conference on Financial Cryptography and Data Security*. Springer. https://eprint.iacr.org/2017/963.pdf

[29] Dionysis Zindros. 2020. *Decentralized Blockchain Interoperability.* Ph.D. Dissertation.