The Velvet Path to Superlight Blockchain Clients

Anonymous Author(s)

ABSTRACT

Superlight blockchain clients learn facts about the blockchain state while requiring merely polylogarithmic communication in the total number of blocks. For proof-of-work blockchains, two known constructions exist: Superblock and FlyClient. Unfortunately, none of them can be deployed to existing blockchains, as they require consensus changes and at least a soft fork to implement.

In this paper, we investigate 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, a process termed a "velvet fork" in the literature. While previous work conjectured that superblock clients 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 class of attacks, called "chain-sewing", which arise in the velvet fork setting: an adversary can cut-and-paste portions of various chains from independent temporary forks, sewing them together to fool a superlight client into accepting a false claim. We show how previous velvet fork constructions can be attacked via chain-sewing. Next, we put forth the first provably secure velvet superblock client construction which we show secure against adversaries that are bounded by 1/3 of the *upgraded* honest miner population. Like non-velvet superlight clients, 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** → *Cryptography*;

KEYWORDS

blockchain, consensus, lightclients, NIPoPoW

1 INTRODUCTION

Blockchain systems such as Bitcoin [33] and Ethereum [4, 40] 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 [18]. 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 [12] of each block in order to determine which chain contains the most work. The 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 5 GB of block header data for the purpose of proof-of-work verification, even if it does not download any transactions. This has become a central problem to the usability of blockchain systems, especially for vendors who use mobile phones to accept payments and 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 number of *superlight* clients has emerged. These protocols give rise to Non-Interactive Proofs of Proof-of-Work (NIPoPoW) [25], which are short strings that "compress" the proof-of-work information of the underlying chain by sending a carefully selected sample of block headers. The necessary security property of such proofs is that 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* [19, 25] 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 and committed at random as in a Σ-protocol (e.g., Schnorr's discrete-log protocol [37]) and then using the Fiat–Shamir heuristic [14] a non-interactive proof is calculated. The number of block headers that need to be sent then grows only logarithmically with time. The NIPoPoW client, which is the proof *verifier* in this context, still relies on a connection to full nodes, who, acting as *provers*, perform the sampling of blocks from the full blockchain. 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 [16, 41]), they can arrive at the correct claim.

In both approaches, the verifier must check that the blocks sampled one way or another were generated in the same order as 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 [32] or Merkle Mountain Ranges [29, 38] whose root is stored in the block header. The inclusion of extra pointers within blocks is termed *interlinking the chain* [23].

The modified block format, which includes the extra pointers, must be respected and validated by full nodes and thus requires a hard or 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 [30]. 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* [43]. A summary of our contributions in this paper is as follows:

(1) 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.
(2) 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 and show it allows to build superlight clients for various statements regarding the blockchain state via both "suffix" and "infix" proofs.
(3) We prove our construction secure in the synchronous static difficulty model against adversaries bounded to 1/3 of the mining power of the honest upgraded nodes. As such, our protocol works even if a constant minority of miners adopts it.

Previous work. Proofs of Proof-of-Work have been proposed in the context of superlight clients [2, 25], cross-chain communication [20, 26, 42], mining [24], as well as local data consumption by smart contracts [21]. Chain interlinking has been proposed for both chains [19] and DAGs [34] has been deployed in production both since genesis [6, 10] and using hard forks [39], and relevant verifiers have been implemented [7, 8]. Deployability using soft forks has also been explored [45]. They have been conjectured to work in velvet fork conditions [25] (we show here that these conjectures are ill-informed in the light of our chain-sewing attack). Velvet forks [43] have been studied for a variety of other applications and have been deployed in practice [17]. In this work, we focus on consensus state compression. Such compression has been explored in the hard-fork setting using zk-SNARKS [31] as well as in the Proof-of-Stake setting [22]. Complementary to consensus state compression (i.e., the compression of block headers and their ancestry) is compression of application state, namely the State Trie, the UTXO, or transaction history. There is a series of works complementary and composable with ours that discusses the compression of application state [5, 27, 35].

Organization. The structure of the paper is as follows. In Section 2, we give a brief overview of the *backbone model* and its notation; as we heavily leverage the machinery of the model, the section is a necessary prerequisite to follow the analysis. The rest of the paper is structured in the form of *a proof and refutation* [28]. We believe this form is more digestible. In Section 3, we discuss the velvet model and some initial definitions, and we present a first attempt towards a velvet NIPoPoW scheme which was presented in previous work. We discuss the informal argument of why it seems to be secure, which we *refute* with an attack explored in Section 4 (we give simulation results and concrete parameters for our attack in Appendix 5). In Section 6, we patch the scheme and put forth our more elaborate and novel Velvet NIPoPoW construction. We analyze it and formally prove it secure in Section 7 (this technical section can be omitted without loss of continuity). Our scheme at this point allows verifiers to decide which blocks form a *suffix* of the longest blockchain and thus the protocol supports *suffix proofs*. We extend our scheme to allow any block of interest within the blockchain to be demonstrated to a prover in a straightforward manner in Section 8, giving a full *infix proof* protocol. The latter protocol can be used in practice and can be deployed today in real blockchains, including Bitcoin, to confirm payments achieving both decentralization and timeliness, solving a major outstanding dilemma in contemporary blockchain systems.

2 PRELIMINARIES

We consider a setting where the blockchain network consists of two types of nodes: The first kind, *full nodes*, are responsible for the verifying the chain 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 [15]. 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 modelled as a common Random Oracle [1]. To each novel query, the random oracle outputs κ 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 [11] by creating multiple messages, but not suppress messages.

Each honest full node locally maintains a *chain* C, a sequence of blocks. As we are developing an improvement on top of SPV, we use the term *block* to mean a *block header*. Each block contains

the Merkle Tree root [32] of transaction data[1] $\overline{x}$, the hash s of the previous block in the chain known as the *previd*, and a nonce ctr. Each block $b = s \parallel \overline{x} \parallel ctr$ must satisfy the proof-of-work [12] equation $H(b) \leq T$ where T is a constant *target*, a small value signifying the difficulty of proof-of-work. We assume T is constant throughout the execution[2]. $H(b)$ 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 index indicates the indexed block. A negative index indicates a block from the end, e.g., $\mathsf{C}[-1]$ is the chain *tip*. 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 use $\parallel$ to signify 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 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.

At every round, every honest party attempts to *mine* a block on top of its chain. Each party is given q queries to the random oracle which it uses in attempting to mine. 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 and broadcast it. Upon receiving a new chain C' from the network, an honest party compares its length $|\mathsf{C}'|$ against its currently adopted chain length $|\mathsf{C}|$ and adopts the new chain if it is longer. The honest parties control the majority of the computational power. This *honest majority assumption* states that there is some $0 < \delta < 1$ such that $t < (1-\delta)(n-t)$. The protocol ensures consensus among honest parties: There is a constant k, the *Common Prefix* parameter, such that, at any round, chains belonging to honest parties share a common prefix; the chains differ only up to k blocks at the end [15]. Concretely, if at some round two honest parties have C_1 and C_2 respectively, then $\mathsf{C}_1[:-k]$ is a prefix of 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 $\mu \in \mathbb{N}$ we say that b is a *μ-superblock* or a block *of level μ*. The probability of a new valid block achieving level μ is $2^{-\mu}$. The number of levels in the chain will be $\log|\mathsf{C}|$ with high probability [23]. Given chain C, we denote $\mathsf{C}{\uparrow}^{\mu}$ the subset of μ-superblocks of 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 proof-of-work samples. The prover sends superblocks to the verifier to convince them that proof-of-work has taken place without actually presenting all this 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 μ and for each such level sends a carefully chosen portion of its μ-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 C points to its previous block $\mathsf{C}[i]$, but each μ-superblock $\mathsf{C}{\uparrow}^{\mu}[i+1]$ does not point to its previous μ-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 μ-superblock points to its most recently preceding μ-superblock. The proposal is therefore to *interlink* the chain by having each μ-superblock include an extra pointer to its most recently preceding μ-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. Therefore, each block is proposed to include a pointer to the most recently preceding μ-superblock, for every μ, 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.

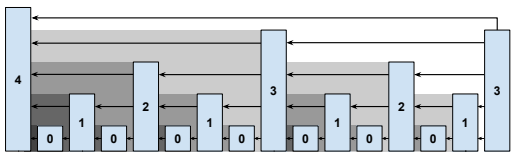

Figure 1: The interlinked blockchain. Each superblock is drawn taller according to its achieved level. Each block links to all the blocks that are not being overshadowed by their descendants. The most recent (right-most) block links to the four blocks it has direct line-of-sight to.

The exact NIPoPoW protocol works like this: The prover holds a full chain 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 π 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 μ^* 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 μ, it includes sufficient μ-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 μ^*. 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*.

[1]The compression of $\overline{x}$ is beyond the scope of our work. Vendors verifying a small number of transactions benefit exponentially from a superlight client even without compressing $\overline{x}$.

[2]A treatment of variable difficulty NIPoPoWs has been explored in the soft fork case [44], but we leave the treatment of velvet fork NIPoPoWs in the variable difficulty model for future work.

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

function $\text{Prove}_{m,k}(\mathsf{C})$
  $B \leftarrow \mathsf{C}[0]$ ▷ 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 π_1 against π_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 μ_1 for π_1 such that $|\pi_1\{b:\}\uparrow^{\mu_1}| \geq m$ (i.e., π_1 has at least m superblocks of level μ_1 following block b) and the value $2^{\mu_1}|\pi_1\{b:\}\uparrow^{\mu_1}|$ is maximized. It chooses a level μ_2 for π_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 party A is deemed longer than the underlying chain of party B.

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

Blockchain protocols can be upgraded using hard or soft forks [3]. 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 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 μ-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 must additionally verify these Merkle proofs.

In a *soft fork*, blocks created by unupgraded miners are not accepted by upgraded miners, but blocks created by upgraded miners are accepted by unupgraded miners. 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 and validate the Merkle Tree root of incoming blocks. This root is easily validated because it is calculated deterministically from the previous blocks in the chain. Unupgraded miners ignore this data and accept the block regardless. The fork is successful if the majority of miners upgrade. Whenever the prover wishes to show that a block b in the proof contains a pointer to its most recently preceding μ-superblock b', it must accompany the block header of $b = s \parallel \overline{x} \parallel ctr$ with the coinbase transaction tx_{cb} of b as well as two Merkle Tree proofs: One proving tx_{cb} is in $\overline{x}$, and one proving $H(b')$ is in the interlink Merkle Tree whose root is committed in tx_{cb}.

3 VELVET INTERLINKS

Velvet forks were recently introduced [43]. 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 upgrade. To maintain backwards compatibility and to avoid causing a permanent fork, 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 interlink the chain with a velvet fork is to have upgraded miners include the interlink pointer in the coinbase of 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 surgical change in the way velvet blocks are produced is necessary to achieve security.

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 $n_h = g(n - t)$.

Unupgraded honest nodes will produce blocks containing no interlink, while upgraded honest nodes will produce blocks containing truthful interlinks. Therefore, any block with invalid interlinks is adversarial. However, such blocks cannot be rejected by the upgraded nodes, as this gives the adversary an opportunity to cause a permanent fork. A block generated by the adversary can thus

contain arbitrary interlinks and yet become honestly adopted. Because the honest prover is an upgraded full node, it determines what the correct interlink pointers are by examining the whole previous chain, and can deduce whether a block contains invalid interlinks. In that case, the prover can simply treat such blocks as unupgraded. In the context of the attack presented in the following section, we examine the case where the adversary includes false interlink pointers. We distinguish blocks based on whether they follow the velvet protocol rules or they deviate from them.

Definition 3.2 (Smooth and Thorny blocks). A block in a velvet upgrade is called *smooth* if it contains auxiliary data corresponding to the honest upgraded protocol. A block is called *thorny* if it contains auxiliary data, but the data differs from the honest upgraded protocol. A block is neither smooth nor thorny if it contains no auxiliary data.

In the case of velvet forks for interlinking, the auxiliary data consists of the interlink Merkle Tree root.

A naïve velvet scheme. In previous work [25], it was conjectured that superblock NIPoPoWs remain secure under a velvet fork. We call this scheme the *Naïve Velvet NIPoPoW* protocol. It is not dissimilar from the NIPoPoW protocol in the soft fork case. The naïve velvet NIPoPoW protocol works as follows. Each upgraded honest miner attempts to mine a block b that includes interlink pointers in the form of a Merkle Tree included in its coinbase transaction. For each level μ, the interlink contains a pointer to the most recent among all the ancestors of b that have achieved at least level μ, regardless of whether the referenced block is upgraded or not and regardless of whether its interlinks are valid. Unupgraded honest nodes will keep mining blocks on the chain as usual; because the status of a block as superblock does not require it to be mined by an upgraded miner, the unupgraded miners contribute mining power to the creation of superblocks.

The prover in the naïve velvet NIPoPoWs works as follows. The honest prover constructs the proof $\pi\chi$ as in Algorithm 2. The outstanding issue is that π does not form a chain because some of its blocks may not be upgraded and they may not contain any pointers (or may contain invalid pointers). Suppose $\pi[i]$ is the most recent μ-superblock preceding $\pi[i+1]$. The prover must provide a connection between $\pi[i+1]$ and $\pi[i]$. The block $\pi[i+1]$ is a superblock and exists at some position j in the underlying chain C of the prover, i.e., $\pi[i+1] = \mathsf{C}[j]$. If $\mathsf{C}[j]$ is smooth, then the interlink pointer at level μ within it can be used. Otherwise, the prover uses the *previd* pointer of $\pi[i+1] = \mathsf{C}[j]$ to repeatedly reach the parents of $\mathsf{C}[j]$, namely $\mathsf{C}[j-1], \mathsf{C}[j-2], \cdots$ until a smooth block b between $\pi[i]$ and $\pi[i+1]$ is found in C, or until $\pi[i]$ is reached. The block b contains a pointer to $\pi[i]$, as $\pi[i]$ is also the most recent μ-superblock ancestor of b. The blocks $\mathsf{C}[j-1], \mathsf{C}[j-2], \cdots, b$ are then included in the proof to illustrate that $\pi[i]$ is an ancestor of $\pi[i+1]$.

The argument for why the above scheme work is as follows. First of all, the scheme does not add many new blocks to the proof. In expectation, if a fully honestly generated chain is processed, after in expectation $\frac{1}{g}$ blocks have been traversed, a smooth block will be found and the connection to $\pi[i]$ will be made. Thus, the number of blocks needed in the proof increases by a factor of $\frac{1}{g}$. Security was argued as follows: An honest party includes in their proof as many blocks as in a soft forked NIPoPoW, albeit by using

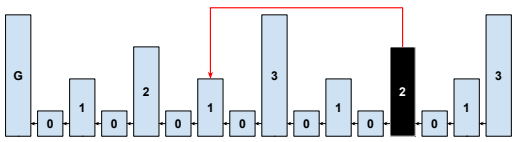

Figure 2: A thorny pointer of an adversarial block, colored black, in an honest party's chain. The thorny block points to a 1-superblock which is an ancestor 1-superblock, but not the *most recent* ancestor 1-superblock.

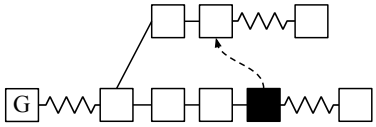

Figure 3: A thorny block, colored black, in an honest party's chain, uses its interlink to point to a fork chain.

an indirect connection. The crucial feature is that it is not missing any superblocks. Even if the adversary creates interlinks that skip over some honest superblocks, the honest prover will not utilize these interlinks, but will use the "slow route" of level 0 instead. The adversarial prover, on the other hand, can only use honest interlinks as before, but may also use false interlinks in blocks mined by the adversary. However, these false interlinks cannot point to blocks of incorrect level. The reason is that the verifier looks at each block hash to verify its level and therefore cannot be cheated. The only problem a fake interlink can cause is that it can point to a μ-superblock which is not the *most recent* ancestor, but some older μ-superblock ancestor in the same chain, as illustrated in Figure 2. However, the adversarial prover can only harm herself by using such pointers, as the result will be a shorter superchain.

We conclude that the honest verifier comparing the honest superchain against the adversarial superchain will reach the same conclusion in a velvet fork as he would have reached in a soft fork: Because the honest superchain in the velvet case contains the same amount of blocks as the honest superchain in the soft fork case, but the adversarial superchain in the velvet case contains fewer blocks than in the soft fork case, the comparison will remain in favor of the honest party. As we will see in the next section, this conclusion is incorrect.

4 THE CHAINSEWING ATTACK

We now make the critical observation that a thorny block can include interlink pointers to blocks that are not its own ancestors in the 0-level chain. Because it must contain a pointer to the hash of the block it points to, they must be older blocks, but they may belong to a different 0-level chain. This is shown in Figure 3.

In fact, as the interlink vector contains multiple pointers, each pointer may belong to a different fork. This is illustrated in Figure 4. The interlink pointing to arbitrary directions resembles a thorny bush.

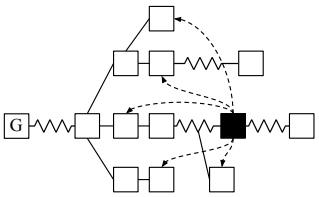

Figure 4: A thorny block appended to an honest party's chain. The dashed arrows are interlink pointers.

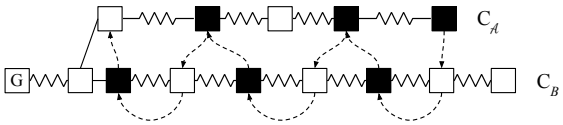

Figure 5: Generic Chainsewing Attack. C_B is the chain of an honest party and $C_{\mathcal{A}}$ is the adversary's chain. Thorny blocks are colored black. Dashed arrows represent interlink pointers included in the adversary's suffix proof. Wavy lines imply one or more blocks.

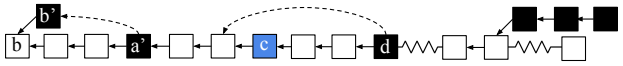

Figure 6: A portion of the concrete Chainsewing Attack. The adversary's blocks are shown in black, while the honestly generated blocks are shown in white. Block b' contains a double spend, while block a' sews it in place. The blue block c is a block included in the honest NIPoPoW, but it is bypassed by the adversary by introducing block d which, while part of the honest chain, points to c's parent. After a point, the adversary forks off and creates $k = 3$ of their own blocks.

We now present the *chainsewing attack* against the naïve velvet NIPoPoW protocol. The attack leverages thorny blocks in order to enable the adversary to *usurp* blocks belonging to a different chain and claim them as her own. Taking advantage of thorny blocks, the adversary produces suffix proofs containing an arbitrary number of blocks belonging to several fork chains. The attack works as follows.

Let C_B be a chain adopted by an honest party B and $C_{\mathcal{A}}$, a fork of C_B at some point, maintained by the adversary. After the fork point $b = (C_B \cap C_{\mathcal{A}})[-1]$, the honest party produces a block extending b in C_B containing a transaction tx. The adversary includes a conflicting (double spending) transaction tx$'$ in a block extending b in $C_{\mathcal{A}}$. The adversary produces a suffix proof to convince the verifier that $C_{\mathcal{A}}$ is longer. 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 party's proof, π_B, so as to achieve $\pi_{\mathcal{A}} \geq_m \pi_B$. Towards this purpose, she mines intermittently on both C_B and $C_{\mathcal{A}}$. She produces some thorny blocks in both chains $C_{\mathcal{A}}$ and C_B which will allow her to usurp selected blocks of C_B and present them to the light client as if they belonged to $C_{\mathcal{A}}$ in her suffix proof.

The general form of this attack for an adversary sewing blocks to one forked chain is illustrated in Figure 5. 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, jumping between $C_{\mathcal{A}}$ and C_B, a chain of blocks crossing forks is formed, which the adversary claims as part of her suffix proof. Blocks of both chains are included in this proof and a verifier cannot distinguish the non-smooth pointers participating in this proof chain and, as a result, considers it a valid proof. Importantly, the adversary must ensure that any blocks usurped from the honest chain are not included in the honest NIPoPoW to force the NIPoPoW verifier to consider an earlier LCA block b; otherwise, the adversary will compete after a later fork point, negating any sewing benefits.

This generic attack is made concrete as follows. The adversary chooses to attack at some level $\mu_{\mathcal{A}} \in \mathbb{N}$. Ideally, the adversary chooses μ_A to be as low as possible. As shown in Figure 6, she first generates a block b' in her forked chain $C_{\mathcal{A}}$ containing the double spend, and a block a' in the honest chain C_B which thorny-points to b'. Block a' will be accepted as valid in the honest chain C_B despite the invalid interlink pointers. The adversary also chooses a desired superblock level $\mu_B \in \mathbb{N}$ that she wishes the honest party to attain. Subsequently, the adversary waits for the honest party to mine and sews any blocks mined on the honest chain that are of level below μ_B. However, she must bypass blocks that she thinks the honest party will include in their final NIPoPoW, which are of level μ_B (the blue block designated c in Figure 6). To bypass a block, the adversary mines her own thorny block d on top of the current honest tip (which could be equal to the block to be bypassed, or have progressed further), containing a thorny pointer to the block preceding the block to be bypassed and hoping d will not exceed level μ_B (if it exceeds that level, she discards her d block). Once m blocks of level μ_B have been bypassed in this manner, the adversary starts bypassing blocks of level $\mu_B - 1$, because the honest NIPoPoW will start including lower-level blocks. The adversary continues descending in levels until a sufficiently low level $\min \mu_B$ has been reached at which point it becomes uneconomical for the adversary to continue bypassing blocks (typically for a $1/4$ adversary, $\min \mu_B = 2$). At this point, the adversary forks off of the last sewed honest block. This last honest block will be used as the last block of the adversarial π part of the NIPoPoW proof. She then independently mines a k-long suffix for the χ portion and creates her NIPoPoW $\pi\chi$. Lastly, she waits for enough time to pass so that the honest party's chain progresses sufficiently to make the previous bypassing guesses correct and so that no blocks in the honest NIPoPoWs coincide with blocks that have not been bypassed. This requires to wait for the following blocks to appear in the honest chain: $2m$ blocks of level μ_B; after the m^{th} μ_B-level block, a further $2m$ blocks of level $\mu_B - 1$; after the m^{th} such block, a further $2m$ blocks of the level $\mu_B - 2$, and so on until level 0 is reached.

In this attack the adversary uses thorny blocks to "sew" portions of the honestly adopted chain to her own forked chain. This justifies the name given to the attack. In order to make this attack successful,

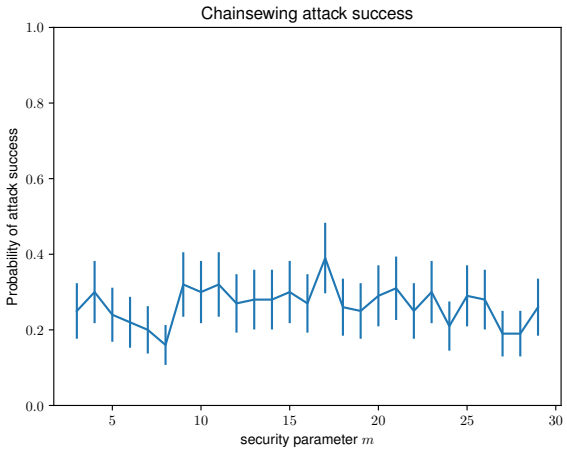

Figure 7: The measured probability of success of the Chainsewing attack mounted under our parameters for varying values of the security parameter m. Confidence intervals at 95%.

the adversary needs only to produce few superblocks, while she can arrogate a large number of honestly produced blocks.

The attack described against superblock NIPoPoWs generalizes to other superlight protocols. We give an overview of the chainsewing attack against velvet FlyClient [2] in Appendix A.

5 CHAINSEWING ATTACK SIMULATION

To measure the success rate of the chainsewing attack against the naïve NIPoPoW construction described in Section 4, we implemented a simulation to estimate the probability of the adversary generating a winning NIPoPoW against the honest party[3]. Our experimental setting is as follows. We fix $\mu_A = 0$ (in case the verifier checks previd pointer consistency, we can set $\mu_A = 1$) and $\mu_B = 10$ as well as the required length of the suffix $k = 15$. We fix the adversarial mining power to $t = 1$ and $n = 5$ which gives a 20% adversary. We then vary the NIPoPoW security parameter for the π portion from $m = 3$ to $m = 30$. We then run 100 Monte Carlo simulations and measure whether the adversary was successful in generating a competing NIPoPoW which compares favourably against the adversarial NIPoPoW.

For performance reasons, our model for the simulation slightly deviates from the Backbone model on which the theoretical analysis of Section 7 is based and instead follows the simpler model of Ren [36]. This model favours the honest parties, and so provides a lower bound for probability of adversarial success, which implies that our attack efficacy is in reality better than estimated here. In this model, block arrival is modelled as a Poisson process and blocks are deemed to belong to the adversary with probability t/n, while they are deemed to belong to the honest parties with probability $(n-t)/n$. Block propagation is assumed instant and every party learns about a block as soon as it is mined. As such, the honest parties are assumed to work on one common chain and the problem of non-uniquely successful rounds does not occur.

We consistently find a success rate of approximately 0.26 which remains more or less constant independent of the security parameter, as expected. We plot our results with 95% confidence intervals in Figure 7. This is in contrast with the best previously known attack in which, for all examined values of the security parameter, the probability of success remains below 1%.

[3]A link to the open source implementation of the attack simulation has been removed for anonymization purposes

6 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 was able to claim not only blocks of shorter forked chains, but also arbitrarily long parts of the chain generated by an honest party. 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 from a thorny block.

The idea for a secure protocol is to distinguish the smooth from the thorny blocks, so that smooth blocks can never point to thorny blocks. In this way we can make sure that thorny blocks acting as passing points to fork chains, as block a' does in Figure 6, cannot be pointed to by honestly generated blocks. Therefore, the adversary cannot utilize honest mining power to construct a stronger suffix proof for her fork chain. Our velvet construction mandates that honest miners create blocks that contain interlink pointers pointing only to previous smooth blocks. As such, newly created smooth blocks can only point to previously created smooth blocks and not thorny blocks. Following the terminology of Section 3, the smoothness of a block in this new construction is a stricter notion than smoothness in the naïve construction.

In order to formally describe the suggested protocol patch, we define smooth blocks in our patched protocol recursively by introducing the notion of a smooth interlink pointer.

Definition 6.1 (Smooth Pointer). A *smooth pointer* of a block b for a specific level μ is the interlink pointer to the most recent μ-level smooth ancestor of b.

We describe a protocol patch that operates as follows. The superblock NIPoPoW protocol works as usual but each honest miner constructs smooth blocks whose interlink contains only smooth pointers; thus it 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 a passage 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 (χ). Consequently, as far as the blocks included in a suffix proof are concerned, we can think of thorny blocks as belonging in the adversary's fork chain for the π part of the proof, which is the critical part for proof comparison. Figure 8 illustrates this remark. The velvet NIPoPoW verifier is also modified to only follow interlink pointers, and never previd

pointers (which could be pointing to thorny blocks, even if honestly generated).

Algorithm 3 Smooth chain for suffix proofs

```
function smoothChain(C)
    C_S ← {G}
    k ← 1
    while C[−k] ≠ G do
        if isSmoothBlock(C[−k]) then
            C_S ← C_S ∪ C[−k]
        end if
        k ← k + 1
    end while
    return C_S
end function
function isSmoothBlock(B)
    if B = G then
        return true
    end if
    for p ∈ B.interlink do
        if ¬isSmoothPointer(B, p) then
            return false
        end if
    end for
    return true
end function
function isSmoothPointer(B, p)
    b ← Block(B.prevId)
    while b ≠ p do
        if level(b) ≥ level(p) ∧ isSmoothBlock(b) then
            return false
        end if
        if b = G then
            return false
        end if
        b ← Block(b.prevId)
    end while
    return isSmoothBlock(b)
end function
```

With this protocol patch we conclude that the adversary cannot usurp honest mining power for use in her fork chain. This change has an undesired side effect: the honest prover cannot utilize thorny blocks belonging in the honest chain. Thus, contrary to the naïve protocol, the honest prover can only depend on *honestly* mined blocks in the honestly adopted chain. Due to this fact, to ensure security in the velvet model, we introduce the assumption that the adversary is bound by 1/3 of the honest *upgraded* mining power.

Definition 6.2 (Velvet Honest Majority). Let n_h be the number of upgraded honest miners. Then t out of total n parties are corrupted such that $\frac{t}{n_h} < \frac{1-\delta_v}{3}$, for some $\delta_v > 0$.

Algorithm 4 Velvet updateInterlink

```
function updateInterlinkVelvet(C_S)
    B′ ← C_S[−1]
    interlink ← B′.interlink
    for μ = 0 to level(B′) do
        interlink[μ] ← id(B′)
    end for
    return interlink
end function
```

Algorithm 5 Velvet Suffix Prover

```
function ProveVelvet_{m,k}(C_S)
    B ← C_S[0]
    for μ = |C_S[−k].interlink| down to 0 do
        α ← C_S[: − k]{B:}↑^μ
        π ← π ∪ α
        B ← α[−m]
    end for
    χ ← C_S[−k:]
    return πχ
end function
```

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

Lemma 6.3. *A velvet suffix proof constructed by an honest party cannot contain any thorny block.*

The following lemma discusses the structure of valid adversarial proofs, i.e., adversarial proofs that pass the honest verifier validation process. The structure is illustrated in Figure 9.

Lemma 6.4. *Any valid 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.*

Proof. Suppose for contradiction that there was a thorny block immediately preceding a smooth block. Then the smooth block would contain a pointer to a thorny block, contradicting the definition of smoothness. □

We now describe the algorithms needed by the upgraded miner, prover and verifier. 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 in his adopted 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 smooth 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, i.e., a pointer to the most recent smooth block for the level denoted by the pointer itself. The *updateInterlink* algorithm is given in Algorithm 4. It is the same as in the case of a soft fork, but works on the smooth chain C_S instead of C.

The construction of the velvet suffix prover is given in Algorithm 5. Again it deviates from the soft fork case by working on the smooth chain C_S instead of C. Lastly, the Verify algorithm for the NIPoPoW suffix protocol remains the same as in the case of

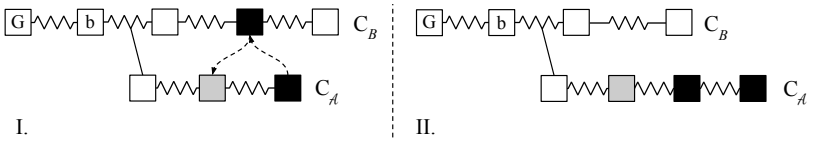

Figure 8: Adversarial fork chain $C_{\mathcal{A}}$ and chain C_B of an honest party. Thorny blocks are colored black. Dashed arrows represent interlink pointers. After the protocol update when an adversarially generated block is sewed from C_B into the adversary's suffix proof the verifier perceives $C_{\mathcal{A}}$ as longer and C_B as shorter. I: The real picture of the chains. II: Equivalent picture from the verifier's perspective considering the suffix proof for each chain.

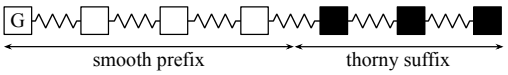

Figure 9: After the protocol update the adversarial velvet suffix proof consists of an initial part of smooth blocks possibly followed by thorny blocks.

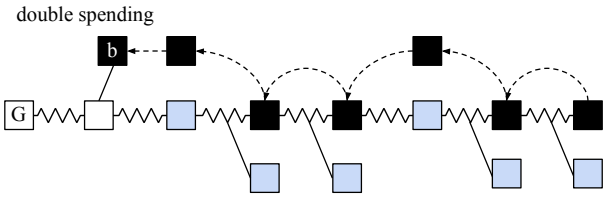

Figure 10: The adversary suppresses honestly generated blocks and chainsews thorny blocks in C_B. Blue blocks are honestly generated blocks of some level of attack. The adversary tries to suppress them. If the suppression is not successful, the adversary can still use the block she mined in her proof.

a hard or soft fork, keeping in mind that no *previd* links can be followed when verifying the ancestry of the chain to avoid hitting any thorny blocks.

7 ANALYSIS

In this section, we prove the security of our scheme. Before we delve in detail into the formal details of the proof, let us first observe why the 1/4 bound is necessary through a combined attack on our construction.

After the suggested protocol update the honest prover cannot include any thorny blocks in his suffix NIPoPoW even if these blocks are part of his chain C_B. The adversary may exploit this fact as follows. She tries to suppress high-level honestly generated blocks in C_B, in order to reduce the blocks that can represent the honest chain in a proof. This can be done by mining a *suppressive block* on the parent of an honest superblock on the honest chain and hoping that she will be faster than the honest parties. In parallel, while she mines suppressive thorny blocks on 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 a second honestly generated block is published soon enough, she does not lose the mining power spent but can still utilize it by including the block in her proof.

In more detail, consider the adversary who wishes to attack a specific block level μ_B and generates a NIPoPoW proof containing a block b of a fork chain which contains a double spending transaction. Then she acts as follows. She mines on her fork chain $C_{\mathcal{A}}$ but when she observes a μ_B-level block in C_B she tries to mine a thorny block on C_B in order to suppress this μ_B block. This thorny block contains an interlink pointer which jumps onto her fork chain, but a previd pointer to the honest chain. If the suppression succeeds she has managed to damage the distribution of μ_B-superblocks within the honest chain, at the same time, to mine a block that she can afterwards use in her proof. If the suppression does not succeed she can still use the thorny block in her proof. The above are illustrated in Figure 10.

The described attack is a combined attack which combines both superblock suppression (initially described in [25], a variant of selfish mining [13]) and chainsewing (introduced in this work). This combined attack forces us to consider the Velvet Honest Majority Assumption of (1/4)-bounded adversary, so as to guarantee that the unsuppressed blocks in C_B suffice for constructing winning NIPoPoW proofs against the adversarial ones.

For the analysis, we use the techniques developed in the Backbone line of work [15]. Towards that end, we follow their definitions and call a round *successful* if at least one honest party made a successful random oracle query during the round, i.e., a query b such that $H(b) \leq T$. A round in which exactly one honest party made a successful query is called *uniquely successful* (the adversary could have also made successful queries during a uniquely successful round). Let $X_r \in \{0,1\}$ and $Y_r \in \{0,1\}$ denote the indicator random variables signifying that r was a successful or uniquely successful round respectively, and let $Z_r \in \mathbb{N}$ be the random variable counting the number of successful queries of the adversary during round r. For a set of consecutive rounds U, we define $Y(U) = \sum_{r \in U} Y_r$ and similarly define X and Z. We denote $f = \mathbb{E}[X_r] < 0.3$ the probability that a round is successful.

Let λ denote the security parameter (the output size κ of the random oracle is taken to be some polynomial of λ). We make use of the following known [15] results. It holds that $pq(n-t) < \frac{f}{1-f}$. For the Common Prefix parameter, it holds that $k \geq 2\lambda f$. Additionally, for any set of consecutive rounds U, it holds that

$\mathbb{E}[Z(U)] < \frac{t}{n-t} \cdot \frac{f}{1-f}|U|$, $\mathbb{E}[X(U)] < pq(n-t)|U|$, $\mathbb{E}[Y(U)] > f(1-f)|U|$. An execution is called *typical* if the random variables X, Y, Z do not deviate significantly (more than some error term $\epsilon < 0.3$) from their expectations. It is known that executions are typical with overwhelming probability in λ. Typicality ensures that for any set of consecutive rounds U with $|U| > \lambda$ it holds that $Z(U) < \mathbb{E}[Z(U)] - \epsilon\,\mathbb{E}[X(U)]$ and $Y(U) > (1-\epsilon)\,\mathbb{E}[Y(U)]$. From the above we can conclude to $Y(U) > (1-\epsilon)f(1-f)|U|$ and $Z(U) < \frac{t}{n-t} \cdot \frac{f}{1-f}|U| + \epsilon f|U|$ which will be used in our proofs. We consider $f < \frac{1}{20}$ a typical bound for parameter f. This is because in our (1/4)-bounded adversary assumption we need to reach about 75% of the network, which requires about 20 seconds [9]. Considering also that in Bitcoin the block generation time is in expectation 600 seconds, we conclude to an estimate $f = \frac{18}{600}$ or $f = 0.03$.

The following definition and lemma are known [44] results and will allow us to argue that some smooth superblocks will survive in all honestly adopted chains. With foresight, we remark that we will take Q to be the property of a block being both smooth and having attained some superblock level $\mu \in \mathbb{N}$.

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

Lemma 7.2 (Unsuppressibility). *Consider a collection of polynomially many block properties $\boldsymbol{Q}$. 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 $Q \in \boldsymbol{Q}$, 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.*

We now apply the above lemma to our construction. The following result lies at the heart of our security proof and allows us to argue that an honestly adopted chain will have a better superblock score than an adversarially generated chain.

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 $\mu, \mu' \in \mathbb{N}$. Let C' be a μ' superchain containing only adversarial blocks generated during U and suppose $|\mathsf{C}_U^S\uparrow^\mu| > k$. Then for any $\delta_3 \leq \frac{3\lambda f}{5}$ it holds that $2^{\mu'}|\mathsf{C}'| < 2^\mu(|\mathsf{C}_U^S\uparrow^\mu| + \delta_3)$.*

Proof. From the Unsuppressibility Lemma we have that there is a set of uniquely successful rounds $S \subseteq U$, 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 generated during S are distributed as in an unsuppressed chain. Therefore considering the property Q for blocks of level μ that contain smooth interlinks we have that $|\mathsf{C}_U^S\uparrow^\mu| \geq (1-\epsilon)g2^{-\mu}|S|$. We also know that for the total number of μ'-blocks the adversary generated during U 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 $((1+\epsilon) + 2g(1-\epsilon))Z(U) < g(1-\epsilon)(Y(U) + \delta')$. But it holds that $(1+\epsilon) + 2g(1-\epsilon) < 3$, therefore it suffices to show that $3Z(U) < g(1-\epsilon)(Y(U) + \delta') - 2^\mu\delta_3$.

Substituting the bounds of X, Y, Z discussed above, it suffices to show that

$$3[\frac{t}{n-t} \cdot \frac{f}{1-f}|U| + \epsilon f|U|] < (1-\epsilon)g[(1-\epsilon)f(1-f)|U| - \delta'] - 2^\mu\delta_3$$

$$\text{or } \frac{t}{n-t} < \frac{(1-\epsilon)g[(1-\epsilon)f(1-f) - \frac{\delta'}{|U|}] - 3\epsilon f - \frac{2^\mu\delta_3}{|U|}}{3\frac{f}{1-f}}.$$

But $\epsilon(1-f) \ll 1$ thus we have to show that

$$\frac{t}{n-t} < \frac{g}{3} \cdot \frac{(1-\epsilon)^2 f(1-f) - \frac{(1-\epsilon)\delta'}{|U|} - \frac{2^\mu\delta_3}{|U|}}{\frac{f}{1-f}} - \epsilon' \quad (1)$$

In order to show Equation 1 we use $f \leq \frac{1}{20}$ which is a typical bound for our setting as discussed above. Because all blocks in C were generated during U and $|\mathsf{C}| > k$, $|U|$ follows negative binomial distribution with probability $2^{-\mu}pq(n-t)$ and number of successes k. Applying a Chernoff bound we have that $|U| > (1-\epsilon)\frac{k}{2^{-\mu}pq(n-t)}$. Using the inequalities $k \geq 2\lambda f$ and $pq(n-t) < \frac{f}{1-f}$, we deduce that $|U| > (1-\epsilon)2^\mu 2\lambda(1-f)$. So we have that $\frac{\delta'}{|U|} < \frac{2\lambda f(\frac{t}{n-t}\frac{1}{1-f} + \epsilon)}{(1-\epsilon)2^\mu 2\lambda(1-f)}$ or $\frac{\delta'}{|U|} < \frac{t}{n-t} \cdot \frac{f}{(1-\epsilon)(1-f)^2} + \epsilon < 0.01 + \epsilon$. We also know that $\delta_3 \leq \frac{3\lambda f}{5}$, so $\frac{2^\mu\delta_3}{|U|} < \frac{2^\mu\frac{3\lambda f}{5}}{2^\mu 2\lambda(1-f)}$ or $\frac{2^\mu\delta_3}{|U|} < \frac{3f}{10(1-f)} < 0.01 + \epsilon$. By substituting the above and the typical f parameter bound in Equation (1) we conclude that it suffices to show that $\frac{t}{n-t} < \frac{1-\epsilon''}{3}g$ which is equivalent to $\frac{t}{n-t} < \frac{1-\delta_v}{3}g$ for $\epsilon'' = \delta_v$, which is the (1/4) velvet honest majority assumption, so the claim is proven. □

Lemma 7.4. *Consider Algorithm 4 under velvet fork with parameter g and (1/4)-bounded velvet honest majority. Consider the property Q for blocks of level μ. Let U be a set of consecutive rounds and C the chain of an honest party at the end of U of a typical execution and $\mathsf{C}_U = \{b \in \mathsf{C} : b \text{ was generated during } U\}$. Suppose that no block in C_U is of level μ. Then $|U| \leq \delta_1$ where $\delta_1 = \frac{(2+\epsilon)2^\mu + \delta'}{(1-\epsilon)f(1-f) - 2\frac{t}{n-t}\frac{f}{1-f} - 3\epsilon f}$.*

Proof. The statement results immediately form the Unsuppressibility Lemma. Suppose for contradiciton that $|U| > \delta_1$. Then

from the Unsuppressibility Lemma we have that there is a subset of consecutive rounds S of U for which it holds that $|S| \geq Y(U) - 2Z(U) - \delta'$ where $\delta' = 2\lambda f(\frac{t}{n-t} \cdot \frac{1}{1-f} + \epsilon)$. By substituting $Y(U) > (1-\epsilon)f(1-f)|U|$ and $Z(U) < \frac{t}{n-t}\frac{f}{1-f} + \epsilon f|U|$ we have that $|S| > (2+\epsilon)2^{\mu}$ but Q-blocks generated during S follow the distribution as in a chain where no suppression attacks occur. Therefore at least one block of level μ would appear in C_U, thus we have reached a contradiction and the statement is proven. □ □

Theorem 7.5 (Suffix Proofs Security under velvet fork). *Assuming honest majority under velvet fork conditions (6.2) such that $t \leq (1-\delta_v)\frac{n_h}{3}$ where n_h the number of upgraded honest parties, 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, π_B is a proof constructed based on underlying blockchain C_B (with $\pi_B \subseteq \mathsf{C}_B$), which B has adopted during round r_3 at which π_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 C_h^r the chain that the honest party h has at round r. Consider also C_B^S the set of smooth blocks of honest chain C_B. We apply security parameter

$$m = 2k + \frac{2+\epsilon+\delta'}{\frac{t}{n-t}\frac{f}{1-f}[f(1-f) - \frac{2}{3}\frac{f}{1-f}]}$$

Suppose for contradiction that 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 μ_B respectively. Let $b_0 = LCA(\pi_{\mathcal{A}}, \pi_B)$. Call $\alpha_{\mathcal{A}} = \pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}\{b_0:\}$, $\alpha_B = \pi_B\uparrow^{\mu_B}\{b_0:\}$.

From Lemma 6.4 we have that the adversarial proof consists of a smooth interlink subchain followed by a thorny interlink subchain. We 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. Block $b_0 = LCA(\pi_{\mathcal{A}}, \pi_B)$ is the fork point between $\pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}, \pi_B\uparrow^{\mu_B}$. Let block $b_1 = LCA(\alpha_{\mathcal{A}}^{\mathcal{S}}, \mathsf{C}_B^S)$ be the fork point between $\pi_{\mathcal{A}}^{\mathcal{S}}\uparrow^{\mu_{\mathcal{A}}}, \mathsf{C}_B$ as an 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 party. 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 $|\alpha_{\mathcal{A}}^3| = k_3$ consisting of adversarial blocks generated during the set of consecutive rounds $\mathcal{S}_3$. Therefore $|\alpha_{\mathcal{A}}| = k_1 + k_2 + k_3$ and we will show that $|\alpha_{\mathcal{A}}| < |\alpha_B|$.

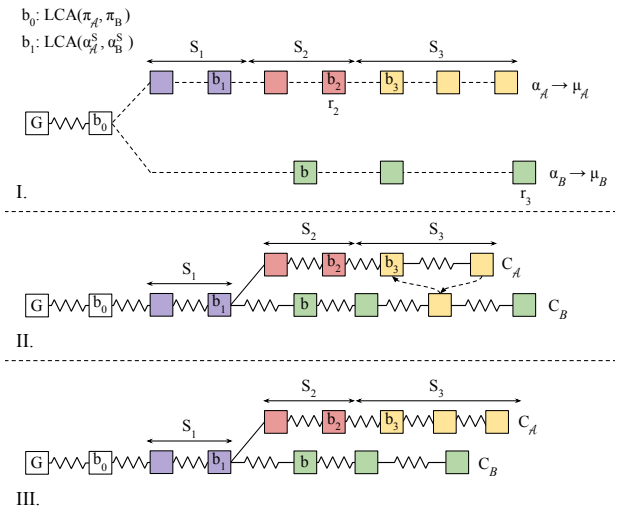

Figure 11: I. the three round sets in two competing proofs at different levels, II. the corresponding 0-level blocks implied by the two proofs, III: blocks in C_B and block set $\widetilde{\mathsf{C}}_{\mathcal{A}}$ from the verifier's perspective.

The above are illustrated, among other, in Parts I, II of Figure 11.

We now show three successive claims: First that $\alpha_{\mathcal{A}}^1$ contains few blocks. Second, $\alpha_{\mathcal{A}}^2$ contains few blocks. And third, the adversary can produce a winning $a_{\mathcal{A}}$ with negligible probability.

Claim 1: $\alpha_{\mathcal{A}}^1 = (\alpha_{\mathcal{A}}\{b_0 : b_1\} \cup b_1)$ *contains only a few blocks.* Let $|\alpha_{\mathcal{A}}^1| = k_1$. We have defined the blocks $b_0 = LCA(\pi_{\mathcal{A}}, \pi_B)$ and $b_1 = LCA(\alpha_{\mathcal{A}}^{\mathcal{S}}, \mathsf{C}_B^S)$. First observe that because of the Lemma 6.4 there are no thorny blocks in $\alpha_{\mathcal{A}}^1$ since $\alpha_{\mathcal{A}}^1[-1] = b_1$ is a smooth block. This means that if b_1 was generated at round r_{b_1} and $\alpha_{\mathcal{A}}^{\mathcal{S}}[-1]$ in round r then $r \geq r_{b_1}$. Therefore, $\alpha_{\mathcal{A}}^1$ contains smooth blocks of C_B. We show the claim by considering the two possible cases for the relation of $\mu_{\mathcal{A}}, \mu_B$.

Claim 1a: If $\mu_B \leq \mu_{\mathcal{A}}$ then $k_1 = 0$. In order to see this, first observe that every block in $\alpha_{\mathcal{A}}$ would also be of lower level μ_B. Subsequently, any block in $\alpha_{\mathcal{A}}\{b_0:\}$ would also be included in proof α_B but this contradicts the minimality of block b_0.

Claim 1b: If $\mu_B > \mu_{\mathcal{A}}$ then $k_1 \leq \frac{\delta_1 2^{-\mu_{\mathcal{A}}}}{(1+\epsilon)\frac{t}{n-t}\frac{f}{1-f}}$. In order to show this we consider block b the first block in α_B. Now suppose for contradiction that $k_1 > \frac{\delta_1 2^{-\mu_{\mathcal{A}}}}{(1+\epsilon)\frac{t}{n-t}\frac{f}{1-f}}$. Then from lemma 7.4 we have that block b is generated during S_1. But b is of lower level $\mu_{\mathcal{A}}$ and $\alpha_{\mathcal{A}}^1$ contains smooth blocks of C_B. Therefore b is also included in $\alpha_{\mathcal{A}}^1$, which contradicts the minimality of block b_0.

Consequently, there are at least $|\alpha_{\mathcal{A}}| - k_1$ blocks in $\alpha_{\mathcal{A}}$ which are not honestly generated blocks existing in C_B. In other words, these

are blocks which are either thorny blocks existing in C_B either don't belong in C_B.

Claim 2. *Part $\alpha^2_{\mathcal{A}} = (\alpha_{\mathcal{A}}\{b_1 : b_2\} \cup b_2)$ 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 party 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 adopted chain C_B at a later round r_3 when the proof π_B is constructed and because of the Common Prefix property on parameter k_2, we conclude that $k_2 \leq 2^{-\mu_{\mathcal{A}}}k$.

Claim 3. *The adversary may submit a suffix proof such that $|\alpha_{\mathcal{A}}| \geq |\alpha_B|$ with negligible probability.* Let $|\alpha^3_{\mathcal{A}}| = k_3$. As explained earlier part $\alpha^3_{\mathcal{A}}$ consists only of adversarially generated blocks. Let S_3 be the set of consecutive rounds $r_2 \ldots r_3$. Then all k_3 blocks of this part of the proof are generated during S_3. Let α^3_B be the last part of the honest proof containing the interlinked μ_B superblocks generated during S_3. Then by applying lemma 7.3 $\frac{m}{k}$ times we have that $2^{\mu_{\mathcal{A}}}|\alpha^3_{\mathcal{A}}| < 2^{\mu_B}(|\alpha_B^{S_3}\uparrow^{\mu_B}| + \frac{m\delta_3}{k})$. By substituting the values from all the above Claims and because every block of level μ_B in $a\alpha_B$ is of equal hashing power to $2^{\mu_B - \mu_{\mathcal{A}}}$ blocks of level $\mu_{\mathcal{A}}$ in the adversary's proof we have that: $2^{\mu_B}|\alpha^3_B| - 2^{\mu_{\mathcal{A}}}|\alpha^3_{\mathcal{A}}| > 2^{\mu_{\mathcal{A}}}(k_1 + k_2)$ or $2^{\mu_B}|\alpha^3_B| > 2^{\mu_{\mathcal{A}}}|\alpha^1_{\mathcal{A}} + \alpha^2_{\mathcal{A}} + \alpha^3_{\mathcal{A}}|$ or $2^{\mu_B}|\alpha_B| > 2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}|$ Therefore we have proven that $2^{\mu_B}|\pi_B\uparrow^{\mu_B}| > 2^{\mu_{\mathcal{A}}}|\pi^{\mu_{\mathcal{A}}}_{\mathcal{A}}|$. □ □

8 INFIX PROOFS

NIPoPoW infix proofs 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 upgraded miner constructs and updates an authenticated data structure for all the blocks in the chain. We suggest Merkle Mountain Ranges *(MMR)* for this structure. Now a velvet block's header additionally includes the root of this MMR.

After this additional protocol change the notion of a smooth block changes as well. Smooth blocks are now considered the blocks that contain truthful interlinks and valid MMR root too. A valid MMR root denotes the MMR that contains all the blocks in the chain of an honest full node. Note that a valid MMR contains all the blocks of the longest valid chain, meaning both smooth and thorny. An invalid MMR constructed by the adversary may contain a block of a fork chain. Consequently an upgraded prover has to maintain a local copy of this MMR locally, in order to construct correct proofs. This is crucial for the security of infix proofs, since keeping the notion of a smooth block as before would allow an adversary to produce a block b in an honest party's chain, with b containing a smooth interlink but invalid MMR, so she could succeed in providing an infix proof about a block of a fork chain.

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 notion of smooth block, the function *isSmoothBlock()* of Algorithm 3 needs to be updated, so that the validity of the included MMR root is also checked. The updated function is given in Algorithm 6. Considering that input C_S is computed using Algorithm 3 with the updated *isSmoothBlock'()* function, *Velvet updateInterlink* and *Velvet Suffix Prover* algorithms remain the same as described in Algorithms 4, 5 repsectively. The velvet infix prover and infix verifier algorithms are given in Algorithms 7, 8 respectively. Details about the construction and verification of an MMR and the respective inclusion proofs can be found in [29]. Note that equivalent solution could be formed by using any authenticated data structure that provides inclusion proofs of size logarithmic to the length of the chain. We suggest MMRs because of they come with efficient update operations.

Algorithm 6 Function isSmoothBlock'() for infix proof support

```
function isSmoothBlock'(B)
    if B = G then
        return true
    end if
    for p ∈ B.interlink do
        if ¬isSmoothPointer(B, p) then
            return false
        end if
    end for
    return containsValidMMR(B)
end function
```

Algorithm 7 Velvet Infix Prover

```
function ProveInfixVelvet(C_S, b)
    (π, χ) ← ProveVelvet(C_S)
    tip ← π[−1]
    π_b ← MMRinclusionProof(tip, b)
    return (π_b, (π, χ))
end function
```

Algorithm 8 Velvet Infix Verifier

```
function VerifyInfixVelvet(b, (π_b, (π, χ)))
    tip ← π[−1]
    return VerifyInclProof(tip.root_MMR, π_b, b)
end function
```

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.. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE.

[3] Vitalik Buterin. 2017. *Hard Forks, Soft Forks, Defaults and Coercion.* https://vitalik.ca/general/2017/03/14/forks_and_markets.html

[4] Vitalik Buterin et al. 2014. *A next-generation smart contract and decentralized application platform.* https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf

[5] Alexander Chepurnoy, Charalampos Papamanthou, and Yupeng Zhang. 2018. Edrax: A Cryptocurrency with Stateless Transaction Validation. *IACR Cryptology ePrint Archive* 2018 (2018), 968.

[6] Elion Chin, Philipp von Styp-Rekowsky, and Robin Linus. 2018. Nimiq. Available at: https://nimiq.com. https://nimiq.com/

[7] Georgios Christoglou. 2018. *Enabling crosschain transactions using NIPoPoWs.* Master's thesis. Imperial College London.

[8] Stelios Daveas, Kostis Karantias, Aggelos Kiayias, and Zindros Dionysis. 2020. A Gas-Efficient Superlight Bitcoin Client in Solidity. *IACR Cryptology ePrint Archive* 2020 (2020), 927.

[9] C. Decker and R. Wattenhofer. 2013. Information propagation in the Bitcoin network. In *IEEE P2P 2013 Proceedings*. 1–10.

[10] ERGO Developers. 2019. Ergo: A Resilient Platform For Contractual Money. https://ergoplatform.org/docs/whitepaper.pdf.

[11] John R Douceur. 2002. The sybil attack. In *International Workshop on Peer-to-Peer Systems*. Springer, 251–260.

[12] Cynthia Dwork and Moni Naor. 1992. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*. Springer, 139–147.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] Kostis Karantias. 2019. *Enabling NIPoPoW Applications on Bitcoin Cash.* Master's thesis. University of Ioannina, Ioannina, Greece.

[18] Kostis Karantias. 2020. SoK: A Taxonomy of Cryptocurrency Wallets. *IACR Cryptology ePrint Archive* 2020 (2020), 868.

[19] 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.

[20] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. 2019. Proof-of-Burn. In *International Conference on Financial Cryptography and Data Security*.

[21] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. 2020. Smart Contract Derivatives. In *International Conference on Mathematical Research for Blockchain Economy*. Imperial College London, Springer. https://eprint.iacr.org/2020/138.pdf

[22] Aggelos Kiayias, Peter Gaži, and Dionysis Zindros. 2019. Proof-of-Stake Sidechains. In *IEEE Symposium on Security and Privacy*. IEEE, IEEE.

[23] 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.

[24] Aggelos Kiayias, Nikos Leonardos, and Dionysis Zindros. 2020. Mining in Logarithmic Space. (2020). Unpublished Manuscript.

[25] 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.

[26] Aggelos Kiayias and Dionysis Zindros. 2019. Proof-of-Work Sidechains. In *International Conference on Financial Cryptography and Data Security*. Springer, Springer.

[27] 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).

[28] Imre Lakatos. 2015. *Proofs and refutations: The logic of mathematical discovery.* Cambridge University Press.

[29] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. RFC6962: Certificate Transparency. *Request for Comments. IETF* (2013).

[30] 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.

[31] Izaak Meckler and Evan Shapiro. 2018. CODA: Decentralized Cryptocurrency at Scale. (2018).

[32] 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.

[33] Satoshi Nakamoto. 2009. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf

[34] Ori Newman and Yonatan Sompolinsky. May 2021. FlyDAG. https://github.com/kaspanet/research/issues/2.

[35] Andrew Poelstra. 2016. Mimblewimble. (2016).

[36] Ling Ren. 2019. Analysis of Nakamoto consensus. *IACR Cryptology ePrint Archive* 2019 (2019), 943.

[37] Claus-Peter Schnorr. 1989. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*. Springer, 239–252.

[38] Peter Todd. October 2012. Merkle Mountain Ranges. https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md.

[39] Ying Tong Lai, James Prestwich, and Georgios Konstantopoulos. 2019. FlyClient - Consensus-Layer Changes. Available at: https://zips.z.cash/zip-0221.

[40] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014), 1–32.

[41] Karl Wüst and Arthur Gervais. 2016. *Ethereum eclipse attacks.* Technical Report. ETH Zurich.

[42] 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.

[43] 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

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

[45] Dionysis Zindros. 2021. Soft Power: Upgrading Chain Macroeconomic Policy Through Soft Forks. In *International Conference on Financial Cryptography and Data Security*. Springer, Springer.

APPENDIX

A FLYCLIENT UNDER VELVET FORK

In the FlyClient paper [2] a velvet fork is suggested for the deployment of the protocol as-is. We now describe an explicit attack against the FlyClient protocol under velvet fork deployment. This is a variation of our Chainsewing Attack of Section 4.

A.1 The FlyClient Protocol

The FlyClient protocol suggests that block headers additionaly include an MMR root of all the blocks in the chain. The protocol uses this root hash in multiple ways, both for chain synchronization and specific block queries. Consider a block b which is appended to the chain C at height h_b:

- the prover generates a merkle inclusion proof Π_b for the existence of b at height h_b in C with respect to the MMR root included in the *tip* of the stable chain $\mathsf{C}[-1]$
- the verifier receives the merkle root of the chain from a prover and an inclusion proof Π_b for block b. He also generates from Π_b the root of the MMR subtree of all blocks in C from genesis up to $\mathsf{C}[h_b - 1]$ and verifies that it is equal to the merkle root included in the header of block b.

The above proofs are produced with respect to the MMR root included in $\mathsf{C}[-1]$.

A high level description of the FlyClient is as follows. Consider a verifier, a superlight client, that asks to synchronize to the current longest valid chain. Suppose that he receives different proofs from two provers. Each prover sends (the header) of the last block in the chain, $\mathsf{C}[-1]$, and a claim for the number of blocks in his chain, $|\mathsf{C}|$ (which he will later be interrogated upon). If both proofs are valid, then the one claiming the greater block count is selected. The validity check of a proof goes as follows. The verifier has received $\mathsf{C}[-1]$, $|\mathsf{C}|$ and queries k random block headers from each prover based on a specific probabilistic sampling algorithm. For each queried block B_i the prover sends the header of B_i along with an MMR subtree inclusion proof Π_{B_i} that B_i is the i_{th} block in the chain. The verifier also checks that B_i is normally mined on the same chain as $\mathsf{C}[-1]$ by verifying that the root included in B_i is the MMR root of the first $(|\mathsf{C}| - 1)$ blocks' subtree. If the k random sampled blocks successfully pass through this verification procedure then the proof is considered valid, otherwise the proof is rejected by the verifier. The chain synchronization protocol is given in Algorithm 9.

Once the longest chain has been chosen, the infix validation is similar to the previous algorithm. The verifier is synchronized to a chain C and already has the tip of the chain $\mathsf{C}[-1]$. He then queries the prover and receives the header of the specific block of interest B in C and the inclusion proof $\Pi_{B\in\mathsf{C}}$. Then the verifier checks the validity of B in the same way as already described for the random sampled blocks in the synchronization protocol. The prover/verifier single-query protocol is given in Algorithm 10.

A.2 Velvet MMRs

In a velvet fork deployment of FlyClient, upgraded miners additionally include an MMR root in each block's header. The claim made in the paper is that considering a constant fraction α of upgraded blocks in the chain, an honest prover could produce proofs by utilizing only these blocks and by joining the intermediary blocks together. The claim is that the velvet proofs remain secure. The velvet protocol is susceptible to a chainsewing attack, succeeding with non-negligible probability. We now give a high-level overview of this attack.

Velvet fork requires any block to be accepted in the chain regardless the validity of the auxiliary data coming with the protocol update. In the case of FlyClient, an adversary may produce blocks which are compatible to the basic consensus rules but contain invalid MMR information. As an example, consider an invalid MMR that omits blocks existing in C or contain blocks which belong in temporary forks of C. We call adversarially generated blocks containing invalid MMRs *thorny* blocks.

Algorithm 9 FlyClient suffix protocol [2]

A verifier performs the following steps speaking with two provers who want to convince him that they hold a valid chain of length $n + 1$. At least one of the provers is honest. If the provers claim different lengths for their claims then the longer chain is checked first.

(1) The provers send to the verifier the last block header in their chain. Each header includes the root of an MMR created over the first n blocks of the corresponding chain.
(2) The verifier queries k random block headers from each prover based on a probabilistic sampling algorithm [2].
(3) For each queried block, B_i, the prover sends the header of B_i along with an MMR proof $\Pi_{B_i\in\mathsf{C}}$ that B_i is the $i-$th block in the chain, and that B's MMR contents form a prefix of the MMR included in C.
(4) The client performs the above checks for each block B_i according to the previous step.
(5) The client rejects the proof if any checks fail.
(6) Otherwise, the client accepts C as the valid chain.

Algorithm 10 FlyClient infix protocol [2]

The verifier queries the prover for the header and MMR proof for a single block b in the prover's chain of $n + 1$ blocks.

Verifier

(1) Has the root of the MMR of n blocks stored in the header of $C[-1]$
(2) Queries prover for the header of block b and for $\Pi_{b\in C}$
(3) Checks the validity of the MMR inclusion proof $\Pi_{b\in C}$
(4) Checks that the MMR of b has contents that are a prefix of the MMR of $C[-1]$
(5) If everything checks out, accepts the block proof

Prover

(1) Has chain of $n+1$ blocks and the MMR of the first n blocks
(2) Receives query for block b from verifier
(3) Calculates $\Pi_{b\in C}$ from MMR_C
(4) Sends header of b and $\Pi_{b\in C}$ to verifier

A.3 The Attack

Consider the security impact of thorny blocks in the FlyClient protocol. Due to the velvet FlyClient description being only partial, we make make a couple of assumptions considering how thorny blocks are treated by the protocol.

When an upgraded *miner* creates a new block, he must include an MMR in this block pointing to all previous blocks (thorny or not).

When the verifier receives two FlyClient client proofs, he will see two chain tips $C_{\mathcal{A}}[-1]$ (by the adversary) and $C_B[-1]$ (by the honest prover). The honest tip $C_B[-1]$ will be a non-thorny block, while the block $C_{\mathcal{A}}[-1]$ may be thorny or non-thorny. He then interrogates each of the provers. During the interrogation, each block b sampled by the verifier may contain their own MMR. In case b does not contain an MMR (b is an unupgraded block), the verifier cannot check consistency between b and $C[-1]$. In the case of the honest prover, it is possible that the sampling yields a block b which is upgraded but thorny. In this case, the consistency check between b and $C_B[-1]$ will fail. Similarly, in the case of the adversarial consistency check, the consistency check may succeed in case both b and $C_{\mathcal{A}}[-1]$ even if both of these are thorny.

In case the honest verifier readily rejects a proof with any inconsistency, as in the soft fork case, the adversary can easily make any honest proof fail by simply introducing thorny blocks in the chain. Therefore, the verifier must accept that some consistency check between b and $C[-1]$ will fail. On the other hand, the verifier cannot allow any number of inconsinstencies freely. To see why, consider the following attack. Let the honest chain have three consecutive blocks b_i, b_{i+1}, b_{i+2} at heights $i, i+1, i+2$ respectively. The attacker produces a thorny block b'_{i+1} on top of b_i containing a double spend transaction in conflict with b_{i+1}. Whenever she wants to convince the verifier that the honest chain contains b'_{i+1} she produces another thorny block b_j on top of the current chain's tip and sends her proof (setting $C_{\mathcal{A}}[-1] = b_j$). Note that b_j is a thorny block that contains an MMR that includes b'_{i+1}. The adversary claims that b'_{i+1} and b_j are the only non-thorny blocks in the chain, contrary to reality, where all *other* upgraded blocks are non-thorny. More specifically b_j will appear in the proof as it is the tip of the chain, while b_{i+1} will be in the proof only if it is selected by the random sampler. Her proof will only be found invalid if the random sampler queries both blocks at heights $i+1, i+2$, so that the invalid prevId pointer shows up. The attacker can decrease this probability by letting the chain grow enough before attempting to construct a proof, as the random sampler chooses older blocks with lower probability (the probability of this sampling occuring is negligible if the proof size is logarithmic). When the verifier receives two valid proofs of the same length, one from the attacker and one from an honest node, the verifier cannot tell which proof is the correct one. One mechanism to avoid this attack is to have the verifier perform consistency verification and count how many of these verifications are successful. In the case of the above attack, most upgraded blocks will appear inconsistent with the claimed (thorny) tip. Intuitively, under an appropriate velvet honest majority assumption, and for sufficiently long underlying chains, the honest proof (ending in a non-thorny block) will pass more consistency checks among the samples performed by the verifier, as compared to the adversarial proof (ending in a thorny block).

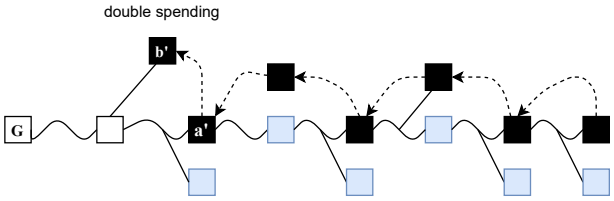

Figure 12: The chainsewing attack. Here, dashed arrows are thorny MMR commitments.

We conclude that an appropriately designed velvet variant of the FlyClient protocol must allow for inconsistencies, but must compare proofs depending on how many consistency checks pass. Despite this favourable version of the FlyClient protocol, we give a sketch of a chainsewing attack which works with an adversary that has more than 1/3 mining power and a constant proportion of the honest upgraded parties (despite the claimed 1/2 adversarial bound in the FlyClient work) .

The adversary utilizes more than one thorny blocks, in order to cut-and-paste portions from the chain adopted by honest parties to her fork chain. The adversary acts as follows. She initially creates a transaction in an honest block b. She subsequently forks from the parent of b to create a block b' containing a double spend. Afterwards she mines block a' in the honest chain C_B, which includes an MMR root containing b'. Next she keeps mining blocks on top of the honest chain, ensuring that they contain MMR commitments to a', b' and the whole other honest chain, excluding b. Additionally, during this period when she mines on C_B she tries to suppress any honest upgraded block in C_B by mining selfishly. When an honest upgraded block $\mathsf{C}[i]$ is appended she mines on top of block $\mathsf{C}[i-1]$. Even if she mines a block and the suppression fails she can still use her fresh block in her proof by continuing to construct consistent MMRs in the coming blocks as described before. Figure 12 illustrates an example of the underlying suppression attack. At some point, the adversary produces her proof. From the verifier's perspective the black colored blocks form a valid chain, since the tip contains consistent MMR commitments with all these blocks. In addition, the random sampling performed by the FlyClient protocol will favour the adversary (with the exception of the case where a consecutive thorny and honest blocks are sampled). We conclude that the FlyClient protocol may be possible to work in velvet conditions through this construction, but similar assumptions to our construction in Section 6 will be necessary to avoid such suppression attacks. Therefore, a stronger velvet honest majority assumption is required.