The Velvet Path to Superlight Blockchain Clients

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. 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.

1 Introduction

Blockchains such as Bitcoin [28] and Ethereum [3,33] maintain chains of blocks that grow linearly with time. A node synchronizing with the rest for the first time must download and validate the whole chain if it does not rely on a trusted third party [15]. While a lightweight node (SPV) can avoid downloading transactions beyond their interest, it must still download block headers containing the proof-of-work [10] (PoW) of each block to determine which chain contains the most work. Block headers, while smaller by a significant constant factor, still grow linearly with time. An Ethereum node synchronizing for the first time must download 4 GB of block headers for proof-of-work verification, even if it does not download transactions. This has become a central problem to the usability of blockchain systems 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) [21], short strings that "compress" the proof-of-work information of the chain by sending a 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 she can convince an SPV client, too.

There are two general directions for superlight clients: In the *superblock* [21,16] approach, the client relies on *superblocks*, blocks achieving much better proof-of-work than required. In the *FlyClient* [1] approach, blocks are randomly sampled and committed as in a Σ-protocol [30] and a non-interactive proof is calculated using Fiat–Shamir [11]. The number of block headers that must be sent then grows only logarithmically with time. The NIPoPoW client, which is the proof *verifier* here, relies on a connection to full nodes, who, acting as *provers*, perform block sampling from the full chain. 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 [13,34]), they arrive at the correct chain.

In both approaches, the verifier must check that blocks sampled one way or another were generated in the same order as presented by the prover. As such, each block in the proof contains 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* of block headers does not suffice. Both approaches require modifications to the consensus layer. For superblock NIPoPoWs, the block header is modified to include, in addition to a pointer to the previous block, pointers to a small amount of recent high-proof-of-work blocks. For FlyClient, each block additionally contains pointers to all previous blocks in the chain. Both these modifications can be made efficiently by organizing these pointers into Merkle Trees [27] or Mountain Ranges [24,31] whose root is stored in the header. Including such extra pointers in blocks is termed *interlinking the chain* [20].

The modified block format must be respected and validated by full nodes and thus requires a hard or soft fork. However, even soft forks require a super-majority approval, and features considered non-essential by the community have taken years to receive it [25]. Towards the goal of implementing superlight clients sooner, we study the question of whether superlight clients can be deployed without soft forks. We propose some *helpful but untrusted* modifications to blocks. These mandate that some extra data is included in each block. The data is placed inside the block by upgraded miners only, while the rest of the miners do not include this data into their blocks and do not verify its inclusion, treating them merely as comments. To maintain backwards compatibility, contrary to a soft fork, upgraded miners accept blocks produced by unupgraded miners that do not contain the extra data, or even blocks containing invalid or malicious such data produced by a mining adversary. This acceptance is necessary to avoid causing a chain split with the unupgraded nodes. Such a modification to consensus is termed a *velvet fork* [36]. A summary of our contributions is as follows:

1. We illustrate that, contrary to previous claims, superlight clients designed to work in a soft fork cannot be readily deployed in a velvet fork. We present the novel and insidious *chain-sewing* attack which thwarts defenses of previous proposals and allows 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 construct a superblock NIPoPoW protocol on top of it and build clients for statements about the blockchain state via "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. Our protocol works even if a minority adopts it.

Previous work. Proofs of Proof-of-Work were proposed in the context of superlight clients [21,1], cross-chain communication [22,17,35], and local data consumption by smart contracts [18]. Interlinking has been deployed in production both since genesis [9,5] and using hard forks [32], and relevant verifiers have been implemented [6,7]. They have been conjectured to work in velvet fork conditions [21] (we show here that these conjectures are ill-informed in light of our attack). Velvet forks have seen many other applications [36] and have been deployed in practice [14]. In this work, we focus on consensus state compression. Alternative constructions in the hard-fork setting use zk-SNARKS [26] or concern Proof-of-Stake [19]. Complementary to consensus state compression (i.e., the compression of block headers) is the compression of application state, namely the State Trie, UTXO, or transaction history. A series of works complementary and composable with ours discusses this [4,23].

Organization. In Section 2, we describe the known superblock NIPoPoW protocol designed for a soft fork (for a detailed description refer to Appendix A and [21]). In Section 3, we discuss the velvet model and some initial definitions, and present a first attempt towards a velvet NIPoPoW scheme which appeared in previous work. A concrete attack is explored in Section 4 (we give simulation results and concrete parameters for our attack in Appendix D). In Section 5, we patch the scheme and put forth our more elaborate and novel Velvet NIPoPoW construction. We analyze it and formally prove it secure in Appendix C. 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 chain to be demonstrated to a prover in a straightforward manner in Section 6, giving a full *infix proof* protocol. The latter protocol 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

The network consists of two node types: *Full nodes* are responsible for verifying the chain and mining new blocks. *Verifiers* connect to full nodes to learn facts about the chain without downloading it, such as whether a transaction is

confirmed. A verifier connects to multiple full nodes, which function as *provers* for the verifier, at least one of which is honest. We model full nodes according to Backbone [12]. There are n full nodes, of which t are adversarial and $n-t$ honest. All t parties are controlled by one colluding adversary $\mathcal{A}$.

Each honest full node locally maintains a *chain* C, a sequence of blocks. From now on we will use the term *block* to mean *block header*. Each block contains a Merkle Tree root [27] of transaction data $\overline{x}$, the hash s of the previous block in the chain (*previd*), and a nonce *ctr*. Each block $b = s \,\|\, \overline{x} \,\|\, ctr$ satisfies the PoW [10] equation $H(b) \leq T$ where T is a constant *target*, a small value signifying the difficulty. We assume T is constant[1]. $H(b)$ is known as the *block id*.

Blockchains are finite block sequences obeying the *blockchain property*: every block in the chain contains a pointer to its previous one. A full valid chain begins with the *genesis* block $\mathcal{G}$, a special block known to all. The verifier only knows about $\mathcal{G}$ when it boots. 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 *tip*. A range $\mathsf{C}[i{:}j]$ is a subarray 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 $\mathsf{C}_1 A$ (we also use $\|$ for concatenation). Here, $\mathsf{C}_2[0]$ must point to $\mathsf{C}_1[-1]$ and A must point to $\mathsf{C}_1[-1]$. We denote $\mathsf{C}\{A{:}Z\}$ the subarray from block A (inclusive) to block Z (exclusive). We omit blocks or indices from either side of the range to take the chain to the beginning or end respectively. If the blockchain property is maintained, we use set operators $\cup$, $\cap$ and $\subseteq$ between chains, implying that blocks are selected and then ordered chronologically.

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 to mine. The adversary has tq queries per round while each honest party has $(n-t)q$ queries per round. When an honest party discovers a new block, they extend their chain and broadcast it. Upon receiving a chain C' from the network, an honest party compares its length $|\mathsf{C}'|$ against its currently adopted length $|\mathsf{C}|$ and adopts the new chain if longer. The *honest majority assumption* states that honest parties control the majority of compute power: $t < (1-\delta)(n-t)$ for some $0 < \delta < 1$. The protocol ensures consensus: There is a 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 [12].

Some valid blocks satisfy the PoW equation better than required. If 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 is $\log|\mathsf{C}|$ with high probability [20]. Given chain C, we denote $\mathsf{C}{\uparrow^{\mu}}$ the subset of μ-superblocks of C.

NIPoPoW protocols allow verifiers to learn the most recent k blocks of the chain adopted by an honest full node. 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,

[1] Variable difficulty NIPoPoWs have been explored in the soft fork case [37]. We leave the treatment of velvet NIPoPoWs in variable difficulty for future work.

the *superblock* approach uses superblocks as PoW samples. The prover sends superblocks to the verifier to convince them that PoW has taken place without actually presenting all this work. The protocol is parametrized by a security parameter m. It determines how many superblocks the prover will send to the verifier. 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 protocols such as Bitcoin and Ethereum, each block $\mathsf{C}[i+1]$ 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 $\mathsf{C}{\uparrow^{\mu}}\ [i]$. An adversarial prover should not be able to reorder the blocks within a superchain. To allow the verifier to check this, each μ-superblock points to its most recent preceding μ-superblock. The proposal is therefore to *interlink* the chain by having each μ-superblock include a pointer to its most recently preceding μ-superblock. To ensure integrity, this pointer must be included in the block header and verified by PoW. 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. This only adds $\log |\mathsf{C}|$ pointers to each block header.

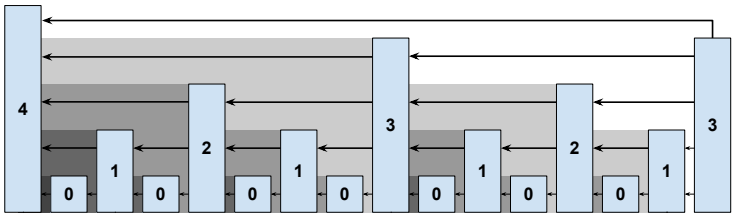


Fig. 1: An interlinked chain. Each superblock is drawn taller according to its level. A block links to all blocks that are not overshadowed by descendants. The most recent (right-most) block links to the four blocks it has direct line-of-sight to.

The NIPoPoW protocol can be in short described 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 prefix $\mathsf{C}[:-k]$, the prover constructs a proof π by selecting certain superblocks as representative samples of the PoW. For each superblock level he adds at least m superblocks.

Upon receiving two proofs $\pi_1\chi_1, \pi_2\chi_2$ of this form, the 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 this, he ensures every block in the proof contains a pointer to its previous block in the proof through either *previd* or an interlink pointer. It then compares π_1 against π_2 as follows: for each proof, he chooses the superblock level containing the most PoW in comparison to all other levels in the proof. Finally, he compares these two selected superblock levels in terms of the aggregate PoW included and the one with more PoW is accepted. For a detailed description, see Appendix A.

Blockchains can be upgraded using hard or soft forks [2]. 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 mandating that interlink pointers are included in the header. To ensure the header is of constant size, instead of including all these superblock pointers in the header individually, they are organized into a Merkle Tree of interlink pointers and only the root is included in the header. In this case, the prover wishing 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 $H(b')$ is a leaf in the interlink Merkle Tree included in the header of b. The verifier verifies 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. Additional data introduced by the upgrade is included in a field treated as a comment by unupgraded miners. To soft fork interlink the chain, the interlink Merkle Tree root is placed in the *coinbase* transaction. Upgraded miners include the correct interlink tree root in their coinbase and validate the tree root of blocks. Unupgraded miners ignore this data and accept the block regardless. When a prover wishes to show that a block b in the proof contains a pointer to its most recently preceding μ-superblock b', it accompanies the header of $b = s \parallel \overline{x} \parallel ctr$ with the coinbase transaction $\mathsf{tx}_{\mathsf{cb}}$ of b and two Merkle Tree proofs: One proving $\mathsf{tx}_{\mathsf{cb}}$ is in $\overline{x}$, and one proving $H(b')$ is in the interlink Merkle Tree whose root is in $\mathsf{tx}_{\mathsf{cb}}$.

3 Velvet Interlinks

Velvet forks were recently introduced [36]. 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 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 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.

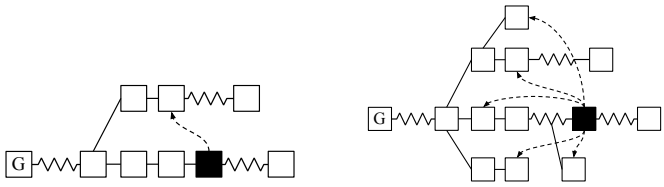
A naïve velvet scheme. In previous work [21], it was conjectured that superblock NIPoPoWs remain secure under a velvet fork. We call this scheme the *Naïve Velvet NIPoPoW* protocol. It is not dissimilar from the NIPoPoW protocol in the soft fork case. The naïve velvet NIPoPoW protocol works as follows. 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 8. 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 this scheme works can be found in Appendix B.

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. In fact, as the interlink vector contains multiple pointers, each pointer may belong to a different fork. The interlink pointing to arbitrary directions resembles a thorny bush. These are shown in Figure 2.



(a) (b)

Fig. 2: (a) A thorny block, colored black, in an honest party's chain, uses its interlink to point to a fork chain. (b) A thorny block appended to an honest party's chain. The dashed arrows are interlink pointers.

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 it 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 $\mathsf{C}_{\mathcal{A}}$, a fork of C_B at some point, be maintained by the adversary. After the fork point $b = (\mathsf{C}_B \cap \mathsf{C}_{\mathcal{A}})[-1]$, 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 $\mathsf{C}_{\mathcal{A}}$. The adversary produces a suffix proof to convince the verifier that $\mathsf{C}_{\mathcal{A}}$ 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 miners intermittently on both C_B and $\mathsf{C}_{\mathcal{A}}$. She produces some thorny blocks in both chains $\mathsf{C}_{\mathcal{A}}$ 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 $\mathsf{C}_\mathcal{A}$ in her suffix proof.

The general form of this attack for an adversary sewing blocks to one forked chain is illustrated in Figure 3. 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 $\mathsf{C}_\mathcal{A}$ 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.

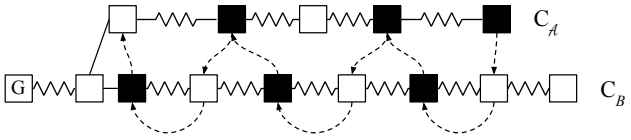


Fig. 3: Generic Chainsewing Attack. C_B is the chain of an honest party and $\mathsf{C}_\mathcal{A}$ the adversary's chain. Thorny blocks are colored black. Dashed arrows represent interlink pointers included in the adversary's suffix proof. Wavy lines imply one or more blocks.

This generic attack is made concrete as follows. The adversary chooses to attack at some level $\mu_\mathcal{A} \in \mathbb{N}$ (ideally, if the honest verifier does not impose any succinctness limits, the adversary sets $\mu_A = 0$). As shown in Figure 4, she first generates a block b' in her forked chain $\mathsf{C}_\mathcal{A}$ 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 4). 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.

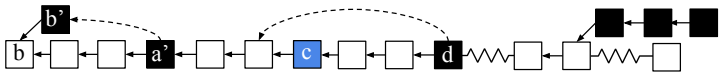


Fig. 4: A portion of the concrete Chainsewing Attack. The adversary's blocks are shown in black, while the honestly generated blocks are shown in white. Block b' contains a double spend, while block a' sews it in place. The blue block c is a block included in the honest NIPoPoW, but it is bypassed by 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.

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, the adversary needs only produce few superblocks, but she can arrogate a large number of honestly produced blocks. Thus the attack succeeds with non-negligible probability.

We illustrate simulation results[2] for the success rate of our attack in Appendix D. Our experiments find the attack with parameters $\mu_B = 10, \mu_A = 0, t = 1, n = 5, k = 15$ succeeds with a constant rate of success of approximately 0.26, when the security parameter m ranges from 3 to 15. This is in contrast to the best previously known attack (which does not make use of thorny blocks), which succeeds with probability less than 0.01. Previous work recommends $m = 15$ for a 1/3 adversary for a probability of failure bounded by 0.001.

5 Velvet NIPoPoWs

In order to eliminate the Chainsewing Attack we propose an update to the velvet NIPoPoW protocol. The core problem is that, in her suffix proof, the adversary 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

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

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 4, 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 3 (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 5 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).

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 4 (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 $\delta_v > 0$.*

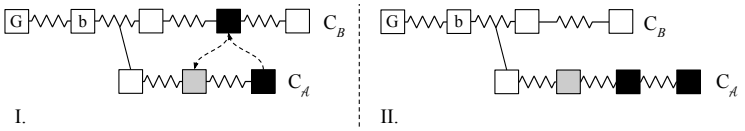


Fig. 5: The adversarial fork chain $\mathsf{C}_{\mathcal{A}}$ and chain C_B of an honest party. Thorny blocks are colored black. Dashed arrows represent interlink pointers. Wavy lines imply one or more blocks. After the protocol update, when an adversarially generated block is sewed from C_B into the adversary's suffix proof the verifier perceives $\mathsf{C}_{\mathcal{A}}$ as longer and C_B as shorter. **I:** The real picture of the chains. **II:** Equivalent picture from the verifier's perspective considering the blocks included in the corresponding suffix proof for each chain.

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

Lemma 1. *A velvet suffix proof constructed by an honest party cannot contain any thorny block.*

Lemma 2. *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. □

Because of Lemma 2 any adversarial proof that successfully passes the honest verifier validation process is of the form illustrated in Figure 6.

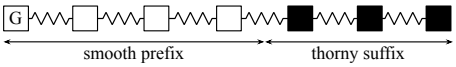


Fig. 6: General case of the adversarial velvet suffix proof $\mathcal{P}_{\mathcal{A}} = (\pi_{\mathcal{A}}, \chi_{\mathcal{A}})$ consisting of an initial part of smooth blocks followed by thorny blocks.

We now describe the algorithms for 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 1. 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 2. It is of the same form as in the case of a soft fork, but works on the smooth chain C_S instead of C.

Algorithm 1 Smooth chain for suffix proofs

```
1: function smoothChain(C)
2:     C_S ← {G}
3:     k ← 1
4:     while C[−k] ≠ G do
5:         if isSmoothBlock(C[−k]) then
6:             C_S ← C_S ∪ C[−k]
7:         end if
8:         k ← k + 1
9:     end while
10:    return C_S
11: end function
12: function isSmoothBlock(B)
13:    if B = G then
14:        return true
15:    end if
16:    for p ∈ B.interlink do
17:        if ¬isSmoothPointer(B, p) then
18:            return false
19:        end if
20:    end for
21:    return true
22: end function
23: function isSmoothPointer(B, p)
24:    b ← Block(B.prevId)
25:    while b ≠ p do
26:        if level(b) ≥ level(p) ∧ isSmoothBlock(b) then
27:            return false
28:        end if
29:        if b = G then
30:            return false
31:        end if
32:        b ← Block(b.prevId)
33:    end while
34:    return isSmoothBlock(b)
35: end function
```

Algorithm 2 Velvet updateInterlink

```
1: function updateInterlinkVelvet(C_S)
2:     B′ ← C_S[−1]
3:     interlink ← B′.interlink
4:     for μ = 0 to level(B′) do
5:         interlink[μ] ← id(B′)
6:     end for
7:     return interlink
8: end function
```

Algorithm 3 Velvet Suffix Prover

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

The construction of the velvet suffix prover is given in Algorithm 3. 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 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. We provide an in-depth analysis and formal proof of our construction's security in Appendix C.

6 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 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 keeping the ids of all the blocks in the chain in the correct order. 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.

Considering this additional patch we can now define the final algorithms for the honest miner, suffix prover, infix prover and infix verifier. Because of the new notion of smooth block, the function *isSmoothBlock()* of Algorithm 1 needs to be altered, so that the validity of the included MMR root is also

checked. The updated function is given in Algorithm 4. Considering the updated *isSmoothBlock'()* function in Algorithm 1, algorithms *Velvet updateInterlink* and *Velvet Suffix Prover* remain the same as described in Algorithms 2, 3 repsectively. The velvet infix prover and infix verifier algorithms are given in Algorithms 5, 6 respectively. Details about MMR construction, verification and inclusion proofs can be found in [24]. 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 they come with efficient update operations.

Algorithm 4 Function isSmoothBlock'() for infix proof support

```
1: function isSmoothBlock'(B)
2:     if B = G then
3:         return true
4:     end if
5:     for p ∈ B.interlink do
6:         if ¬isSmoothPointer(B, p) then
7:             return false
8:         end if
9:     end for
10:    return containsValidMMR(B)
11: end function
```

Algorithm 5 Velvet Infix Prover

```
1: function ProveInfixVelvet(C_S, b)
2:     (π, χ) ← ProveVelvet(C_S)
3:     tip ← π[−1]
4:     π_b ← MMRinclusionProof(tip, b)
5:     return (π_b, (π, χ))
6: end function
```

Algorithm 6 Velvet Infix Verifier

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

Appendix

A NIPoPoW protocol

The exact NIPoPoW protocol under a soft fork 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 7. 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*.

Algorithm 7 The Prove algorithm for the NIPoPoW protocol in a soft fork

function $\mathsf{Prove}_{m,k}(\mathsf{C})$
  $B \leftarrow \mathsf{C}[0]$ ▷ Genesis
  for $\mu = |\mathsf{C}[-k-1].\mathsf{interlink}|$ 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 8.

Algorithm 8 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 $\mathsf{best\text{-}arg}_m(\pi, b)$
  $M \leftarrow \{\mu : |\pi\uparrow^{\mu} \{b:\}| \geq m\} \cup \{0\}$ ▷ 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 $\mathsf{best\text{-}arg}_m(\pi_A, b) \geq \mathsf{best\text{-}arg}_m(\pi_B, b)$
end operator

B Naïve velvet scheme

The argument for why the Naïve velvet protocol works 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 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 7. 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

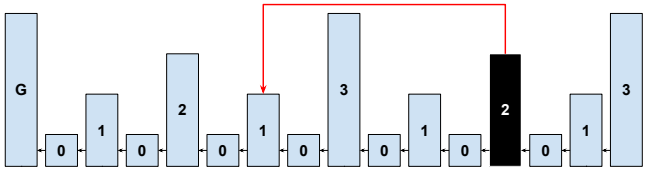


Fig. 7: A thorny pointer of an adversarial block, colored black, in an honest party's chain. The thorny block points to a 1-superblock which is an ancestor 1-superblock, but not the *most recent* ancestor 1-superblock.

than in the soft fork case, the comparison will remain in favor of the honest party. As described in Section 4, this conclusion is incorrect.

C 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 $\mathsf{C}_{\mathcal{A}}$ 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 8.

The described attack is a combined attack which combines both superblock suppression (initially described in [21]) 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 unsup-

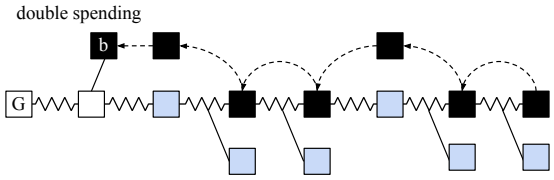


Fig. 8: The adversary suppresses honestly generated blocks and chainsews thorny blocks in C_B. Blue blocks are honestly generated blocks of some level of attack. The adversary tries to suppress them. If the suppression is not successful, the adversary can still use the block she mined in her proof.

pressed 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 [12]. 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 [12] results. It holds that $pq(n-t) < \dfrac{f}{1-f}$. 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 $Z(U) < \dfrac{t}{n-t} \cdot \dfrac{f}{1-f}|U| + \epsilon f|U|$ which will be used in our proofs. We consider $f < \dfrac{1}{20}$ 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 [8]. Considering also that in Bitcoin the block generation time is in expectation 600 seconds, we conclude to an estimate $f = \dfrac{18}{600}$ or $f = 0.03$.

The following definition and lemma are known [37] 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 5 (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 3 (Unsuppressibility). *Consider a collection of polynomially many block properties $\mathcal{Q}$. In a typical execution every set of consecutive rounds U has a subset S of uniquely successful rounds such that*

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

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 4. *Consider Algorithm 2 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

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

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

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

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 5. *Consider Algorithm 2 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 $\mathsf{C}_U = \{b \in \mathsf{C} : b$ 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 1 (Suffix Proofs Security under velvet fork). *Assuming honest majority under velvet fork conditions (4) 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 2 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 $|\alpha_{\mathcal{A}}| < |\alpha_B|$.

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

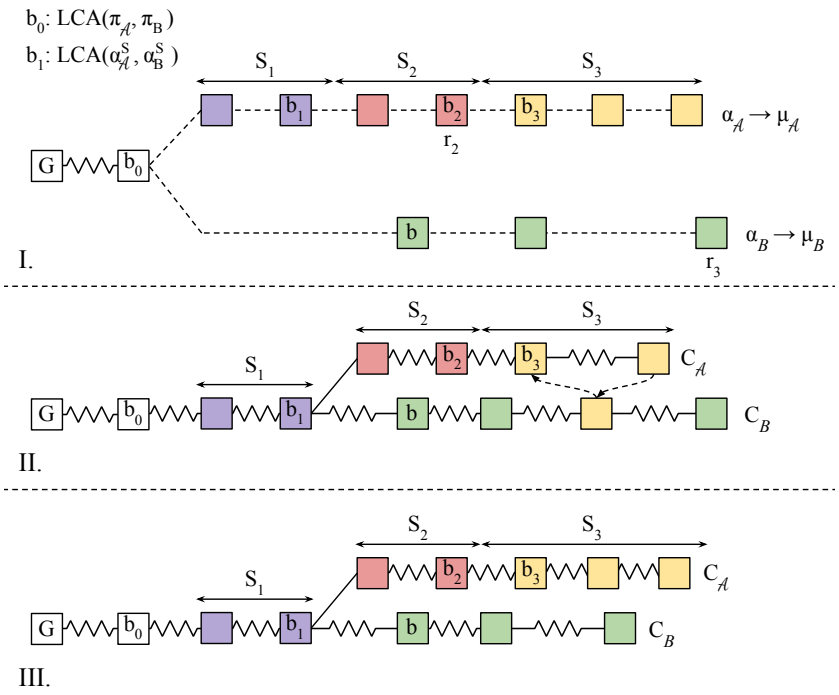


Fig. 9: I. the three round sets in two competing proofs at different levels, II. the corresponding 0-level blocks implied by the two proofs, III: blocks in C_B and block set $\widetilde{\mathsf{C}}_{\mathcal{A}}$ from the verifier's perspective.

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

Claim 1: $\alpha^1_{\mathcal{A}} = (\alpha_{\mathcal{A}}\{b_0 : b_1\} \cup b_1)$ *contains only a few blocks.* Let $|\alpha^1_{\mathcal{A}}| = k_1$. We have defined the blocks $b_0 = LCA(\pi_{\mathcal{A}}, \pi_B)$ and $b_1 = LCA(\alpha^{\mathcal{S}}_{\mathcal{A}}, \mathsf{C}^{\mathcal{S}}_B)$. First observe that because of the Lemma 2 there are no thorny blocks in $\alpha^1_{\mathcal{A}}$ since $\alpha^1_{\mathcal{A}}[-1] = b_1$ is a smooth block. This means that if b_1 was generated at round r_{b_1} and $\alpha^{\mathcal{S}}_{\mathcal{A}}[-1]$ in round r then $r \geq r_{b_1}$. Therefore, $\alpha^1_{\mathcal{A}}$ 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 $k_1 \leq \dfrac{\delta_1 2^{-\mu_{\mathcal{A}}}}{(1+\epsilon)\dfrac{t}{n-t}\dfrac{f}{1-f}}$. In order to show this we consider block b the first block in α_B. Now suppose for contradiction that $k_1 > \dfrac{\delta_1 2^{-\mu_{\mathcal{A}}}}{(1+\epsilon)\dfrac{t}{n-t}\dfrac{f}{1-f}}$. Then from lemma 5 we have that block b is generated during S_1. But b is of lower level $\mu_{\mathcal{A}}$ and $\alpha^1_{\mathcal{A}}$ contains smooth blocks of C_B. Therefore b is also included in $\alpha^1_{\mathcal{A}}$, 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 $\mathsf{C}^2_{\mathcal{A}}$. 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 $\mathsf{C}^2_{\mathcal{A}}$ 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 $r_2 ... r_3$. 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 4 $\dfrac{m}{k}$ times we have that $2^{\mu_{\mathcal{A}}}|\alpha^3_{\mathcal{A}}| < 2^{\mu_B}(|\alpha^{S_3}_B \uparrow^{\mu_B}| + \dfrac{m\delta_3}{k})$. 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: $2^{\mu_B}|\alpha_B^3| - 2^{\mu_\mathcal{A}}|\alpha_\mathcal{A}^3| > 2^{\mu_\mathcal{A}}(k_1 + k_2)$ or $2^{\mu_B}|\alpha_B^3| > 2^{\mu_\mathcal{A}}|\alpha_\mathcal{A}^1 + \alpha_\mathcal{A}^2 + \alpha_\mathcal{A}^3|$ or $2^{\mu_B}|\alpha_B| > 2^{\mu_\mathcal{A}}|\alpha_\mathcal{A}|$ Therefore we have proven that $2^{\mu_B}|\pi_B{\uparrow^{\mu_B}}| > 2^{\mu_\mathcal{A}}|\pi_\mathcal{A}^{\mu_\mathcal{A}}|$. □

D 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. Our experimental setting is as follows. We fix $\mu_A = 0$ 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.

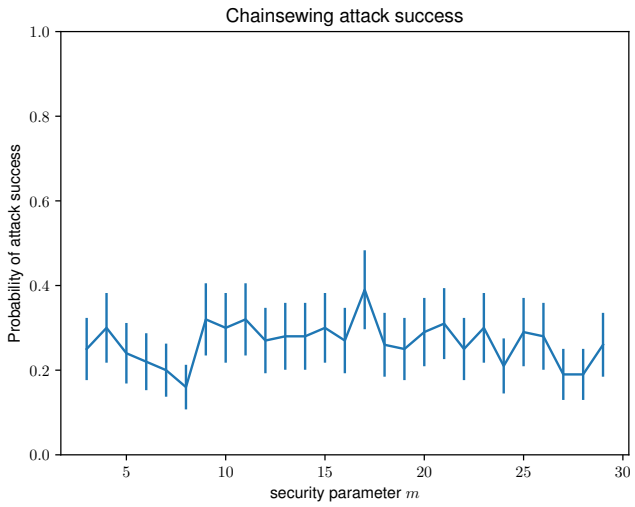


Fig. 10: The measured probability of success of the Chainsewing attack mounted under our parameters for varying values of the security parameter m. Confidence intervals at 95%.

For performance reasons, our model for the simulation slightly deviates from the Backbone model on which the theoretical analysis of Section C is based and instead follows the simpler model of Ren [29]. 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 10. 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%.

References

1. B. Bünz, L. Kiffer, L. Luu, and M. Zamani. Flyclient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020.
2. V. Buterin. Hard forks, soft forks, defaults and coercion. Available at: https://vitalik.ca/general/2017/03/14/forks_and_markets.html, 2017.
3. V. Buterin et al. A next-generation smart contract and decentralized application platform. Available at: https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf, 2014.
4. A. Chepurnoy, C. Papamanthou, and Y. Zhang. Edrax: A cryptocurrency with stateless transaction validation. *IACR Cryptology ePrint Archive*, 2018:968, 2018.
5. E. Chin, P. von Styp-Rekowsky, and R. Linus. Nimiq. Available at: https://nimiq.com, 2018.
6. G. Christoglou. Enabling crosschain transactions using nipopows. Master's thesis, Imperial College London, 2018.
7. S. Daveas, K. Karantias, A. Kiayias, and Z. Dionysis. A Gas-Efficient Superlight Bitcoin Client in Solidity. *IACR Cryptology ePrint Archive*, 2020:927, 2020.
8. C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, 2013.
9. E. Developers. Ergo: A Resilient Platform For Contractual Money, 2019. https://ergoplatform.org/docs/whitepaper.pdf.
10. C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.
11. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.
12. J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310, 2015.
13. E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security Symposium*, pages 129–144, 2015.
14. K. Karantias. Enabling NIPoPoW Applications on Bitcoin Cash. Master's thesis, University of Ioannina, Ioannina, Greece, 2019.
15. K. Karantias. SoK: A Taxonomy of Cryptocurrency Wallets. *IACR Cryptology ePrint Archive*, 2020:868, 2020.
16. K. Karantias, A. Kiayias, and D. Zindros. Compact storage of superblocks for nipopow applications. In *The 1st International Conference on Mathematical Research for Blockchain Economy*. Springer Nature, 2019.
17. K. Karantias, A. Kiayias, and D. Zindros. Proof-of-burn. In *International Conference on Financial Cryptography and Data Security*, 2019.
18. K. Karantias, A. Kiayias, and D. Zindros. Smart contract derivatives. In *International Conference on Mathematical Research for Blockchain Economy*. Imperial College London, Springer, 2020.
19. A. Kiayias, P. Gaži, and D. Zindros. Proof-of-stake sidechains. In *IEEE Symposium on Security and Privacy*. IEEE, IEEE, 2019.
20. A. Kiayias, N. Lamprou, and A.-P. Stouka. Proofs of proofs of work with sublinear complexity. In *International Conference on Financial Cryptography and Data Security*, pages 61–78. Springer, 2016.

21. A. Kiayias, A. Miller, and D. Zindros. Non-Interactive Proofs of Proof-of-Work. In *International Conference on Financial Cryptography and Data Security*. Springer, 2020.
22. A. Kiayias and D. Zindros. Proof-of-work sidechains. In *International Conference on Financial Cryptography and Data Security*. Springer, Springer, 2019.
23. J.-Y. Kim, J.-M. Lee, Y.-J. Koo, S.-H. Park, and S.-M. Moon. Ethanos: Lightweight bootstrapping for ethereum. *arXiv preprint arXiv:1911.05953*, 2019.
24. B. Laurie, A. Langley, and E. Kasper. Rfc6962: Certificate transparency. *Request for Comments. IETF*, 2013.
25. E. Lombrozo, J. Lau, and P. Wuille. BIP 0141: Segregated witness (consensus layer). Available at: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki, 2015.
26. I. Meckler and E. Shapiro. CODA: Decentralized Cryptocurrency at Scale. 2018.
27. R. C. Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.
28. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.
29. L. Ren. Analysis of Nakamoto consensus. *IACR Cryptology ePrint Archive*, 2019:943, 2019.
30. C.-P. Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.
31. P. Todd. Merkle mountain ranges, October 2012. https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md.
32. Y. Tong Lai, J. Prestwich, and G. Konstantopoulos. Flyclient - consensus-layer changes. Available at: https://zips.z.cash/zip-0221, Mar 2019.
33. G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151:1–32, 2014.
34. K. Wüst and A. Gervais. Ethereum eclipse attacks. Technical report, ETH Zurich, 2016.
35. A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt. SoK: Communication across distributed ledgers, 2019.
36. A. Zamyatin, N. Stifter, A. Judmayer, P. Schindler, E. Weippl, W. Knottenbelt, and A. Zamyatin. A wild velvet fork appears! inclusive blockchain protocol changes in practice. In *International Conference on Financial Cryptography and Data Security*. Springer, 2018.
37. D. Zindros. *Decentralized Blockchain Interoperability*. PhD thesis, Apr 2020.