Superlight Blockchain Client with Minority Upgrade

Anonymous Author(s)

ABSTRACT

Superlight clients allow decentralized wallets to learn facts about the blockchain without downloading all the block headers. They achieve exponentially faster communication compared to SPV clients. For proof-of-work, they implement the so-called NIPoPoW primitive, for which there exist two variants: Superblock clients and FlyClient. Both of these protocols require consensus changes to existing blockchains and at least a soft fork to implement. In this paper, we discuss how a blockchain can be upgraded to support superblock clients without a soft fork. We show that it is possible to implement the needed changes without modifying the consensus protocol and by requiring only a minority of miners to upgrade, an upgrade termed a "velvet fork" in the literature. While previous work conjectured that NIPoPoW can be safely deployed using velvet forks as-is, we show that previous constructions are insecure, and that using velvet techniques to interlink a blockchain can pose insidious security risks. We describe a novel attack which we term a "chain-sewing" attack which is only possible in velvet situations: An adversary can cut-and-paste portions of various chains from independent forks, sewing them together to form a proof that looks like a chain but is not. We demonstrate that a minority adversary can thwart previous constructions with overwhelming probability. We put forth the first provably secure velvet NIPoPoW construction. Our construction is secure against adversaries that are bounded by 1/2 of the upgraded honest miner population. We prove our construction achieves persistence and liveness and analyze the trade-offs between the upgraded population parameter and the succinctness of the construction. Like non-velvet NIPoPoWs, our approach allows proving generic predicates about chains using infix proofs and as such can be adopted in practice for fast synchronization of transactions and accounts.

CCS CONCEPTS

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

KEYWORDS

blockchain, lightclient, NIPoPoW

1 INTRODUCTION

[4][5][1][2][3]

2 THE CHAINSEWING ATTACK

We will now describe an explicit attack against the NIPoPoW suffix proof construction under velvet fork. As already argued, since the protocol is implemented under velvet fork, any adversarial block that is mined in the proper way except containing false interlink data will be accepted as valid. Taking advantage of such thorny blocks in the chain, the adversary maintaining one or more forked chains could produce suffix proofs containing blocks of any chain. The attack is described in detail in the following.

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

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

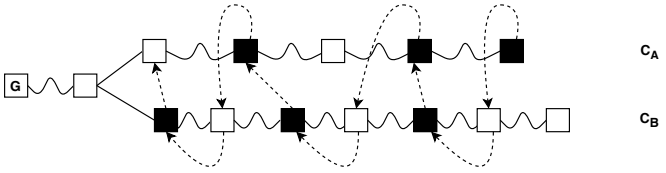

Figure 1: ***Generic Chainsewing Attack.*** C_B ***is the chain of an honest player and*** C_A ***the adversary's chain. Adversarially generated blocks are colored black. Dashed arrows represent interlink pointers included in the adversary's suffix proof. Wavy lines imply one or more blocks.***

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

At this point the adversary will produce a suffix proof for chain C_A containing the subchain $C\{ab\} \cup C\{b : a'\} \cup C\{a' : b'\}$. Notice

that following the interlink pointers constructed in such a way, a light client perceives $C\{ab\} \cup C\{b : a'\} \cup C\{a' : b'\}$ as a valid chain.

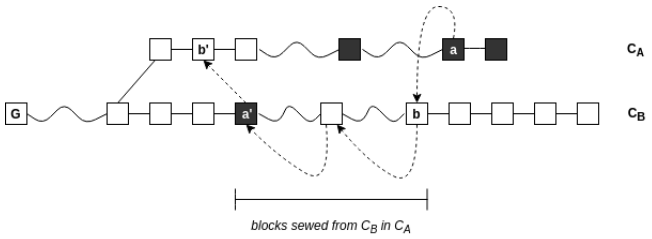

Figure 2: *Chainsewing Attack. C_B represents the chain of an honest player. C_A is an adversarial fork. Adversarially generated blocks are colored black. Dashed arrows represent interlink pointers included in the adversary's suffix proof. Wavy lines imply one or more blocks. Firm lines imply the previousId relationship between two sequential blocks.*

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

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

3 PROTOCOL UPDATE

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

3.0.1 Velvet fork parameter. Under stable conditions a velvet fork suggest that only a minority of upgraded parties needs to support the protocol update. Let g express the percentage of honest upgraded parties to the total number of miners. We will refer to g as the "velvet parameter".

Definition 1. Velvet Parameter *Let g be the velvet parameter for NIPoPoW protocols. Then if n_h the upgraded honest miners and n the total number of miners t out of which are corrupted, it holds that $n_h = g(n - t)$.*

3.0.2 Honest Majority Assumption. Compared to the typical setting of 1/2-bounded adversary, the protocol we propose requires stronger honest majority assumptions to be provably secure. In particular, our protocol is secure for adversary of total hashing power less than 1/2 of the upgraded honest miners, meaning less than 1/3 of the total number of miners generating blocks with interlinks. Therefore we define the Velvet Honest Majority assumption, which will be used in our security proof.

Definition 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 $\frac{t}{n_h} < \frac{1-\delta}{2}$.*

We describe a protocol patch that operates as follows. The NIPoPoW protocol under velvet fork works as usual but each miner constructs smooth blocks. This means that a block's interlink is constructed excluding thorny blocks. In this way, although thorny blocks are accepted in the chain, they are not taken into consideration when updating the interlink structure for the next block to be mined. No honest block could now point to a thorny superblock that may act as the passing point to the fork chain in an adversarial suffix proof. Thus, after this protocol update the adversary is only able to inject adversarially generated blocks from an honestly adopted chain to her own fork. At the same time, thorny blocks cannot participate in an honestly generated suffix proof except for some blocks in the proof's suffix (χ). This arguments holds because thorny blocks do not form a valid chain along with honestly mined blocks anymore. Consequently, as far as the blocks included in a suffix proof is concerned, we can think of thorny blocks as belonging in the adversary's fork chain for the π part of the proof, which is the comparing part between proofs. Figure 3 illustrates this remark.

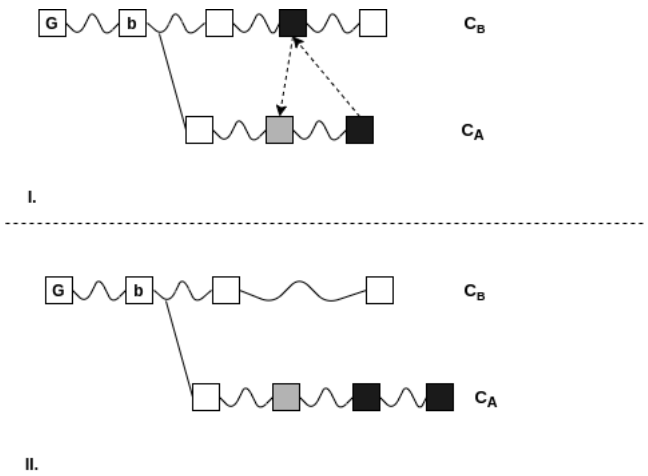

Figure 3: *The adversarial fork chain C_A and chain C_B of an honest player. Thorny blocks are colored black. Dashed arrows represent interlink pointers. Wavy lines imply one or more blocks. When an adversarially generated block is sewed from C_B into the adversary's suffix proof the verifier conceives C_A as longer and C_B as shorter. I: The real picture of the chains. II: Equivalent picture from the verifier's perspective considering the blocks included in the corresponding suffix proof for each chain.*

The protocol patch we suggest can be summarized as follows:

Protocol Patch for NIPoPows suffix proofs under velvet fork. In order to make NIPoPoWs safe under velvet fork conditions we suggest:

(1) Strengthen the Honest Majority Assumption so that $t < \frac{1-\delta}{2} n_h$, where n_h is the number of upgraded honest players.
(2) The NIPoPoW protocol under velvet fork works as usual but a miner constructs a block's interlink without pointers to thorny blocks.

3.1 *Smooth* and *Thorny* blocks

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

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

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

Definition 3. Smooth Pointer *A smooth pointer of a block b for a specific level μ is the interlink pointer to the most recent μ-level ancestor of b.*

A non-smooth pointer may not point to the most recent ancestor of level μ, or even point to a superblock of a forked chain, as shown in Figure 4.

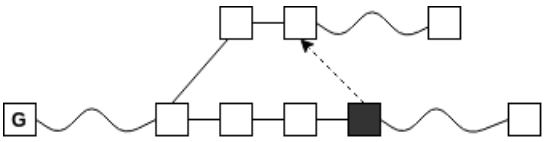

Figure 4: *A non-smooth pointer of an adversarial block, colored black, in an honest player's chain.*

In the same manner it is possible that a false interlink contain arbitrary pointers to blocks of any chain as illustrated in Figure 5. The interlink pointing to arbitrary directions resembles a thorny bush, so we will refer to blocks containing false interlink information as *thorny*.

Definition 4. Thorny Block *A thorny block is a block which contains at least one non-smooth interlink pointer.*

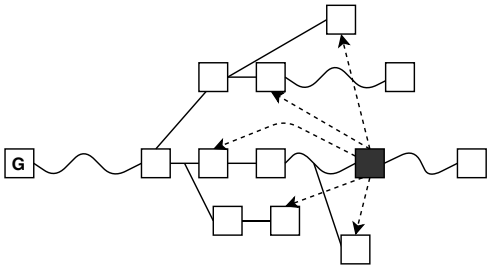

Figure 5: *A thorny block appended in an honest player's chain. The dashed arrows are interlink pointers.*

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

Definition 5. Smooth Block *A smooth block is any block which is not thorny.*

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

Lemma 3.1. *A velvet suffix proof constructed by an honest player cannot contain any thorny block.*

Lemma 3.2. *Let $\mathcal{P}_{\mathcal{A}} = (\pi_{\mathcal{A}}, \chi_{\mathcal{A}})$ be a velvet suffix proof constructed by the adversary and block b_s, generated at round r_s, be the most recent smooth block in the proof. Then $\forall r : r < r_s$ no thorny blocks generated at round r can be included in $\mathcal{P}_{\mathcal{A}}$.*

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

□

Lemma 3.3. *Let $\mathcal{P}_{\mathcal{A}} = (\pi_{\mathcal{A}}, \chi_{\mathcal{A}})$ be a velvet suffix proof constructed by the adversary. Let b_t be the oldest thorny bock included in $\mathcal{P}_{\mathcal{A}}$ which is generated at round r_t. Then any block $b = \{b : b \in \mathcal{P}_{\mathcal{A}} \wedge b$ generated at $r \geq r_t\}$ is thorny.*

Proof. By contradiction. Suppose for contradiciton that b_s is a smooth block generated at round $r_s > r_t$. Then from Lemma 3.2 any block generated at round $r < r_s$ is smooth. But b_t is generated at round $r_t < r_s$ and is thorny, thus we have reached a contradiction.

□

The following corollary emerges immediately from Lemmas 3.2, 3.3. This result is illustrated in Figure 6.

Corollary 3.4. *Any adversarial proof $\mathcal{P}_{\mathcal{A}} = (\pi_{\mathcal{A}}, \chi_{\mathcal{A}})$ containing both smooth and thorny blocks consists of a prefix smooth subchain followed by a suffix thorny subchain.*

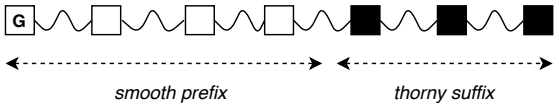

Figure 6: *In the general case the adversarial velvet suffix proof $\mathcal{P}_{\mathcal{A}} = (\pi_{\mathcal{A}}, \chi_{\mathcal{A}})$ consists of an initial part of smooth blocks followed by thorny blocks.*

We now describe the algorithms needed by the upgraded miner, prover and verifier. The upgraded miner acts as usual except for including the interlink of the newborn block in the coinbase transaction. In order to construct an interlink containing only the smooth blocks, the miner keeps a copy of the "smooth chain" (C_S) which consists of the smooth blocks existing in the original chain C. The algorithm for extracting the smooth chain out of C is given in Algorithm 1. Function *isSmoothBlock(B)* checks whether a block B is a smooth velvet by calling *isSmoothPointer(B,p)* for every pointer p in B's interlink. Function *isSmoothPointer(B,p)* returns *true* if p is a valid pointer, in essence a pointer to the most recent *smooth velvet* for the level denoted by the pointer itself. The *updateInterlink* algorithm is given in Algorithm 2, which is essentially the same as in the case of a hard/soft fork, except for working on the smooth chain C_S instead of C.

The construction of the velvet suffix prover is given in Algorithm 3, which is essentially the same to that of a hard/soft fork except for working on smooth chain C_S instead of C.

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

4 CONCLUSIONS

In conclusion, it is rarely a good idea to include the same section three times in a paper, or to have a conclusion that does not conclude.

A LOCATION

Note that in the new ACM style, the Appendices come before the References.

Algorithm 1: Compute smooth chain

```
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
8          k ← k + 1
9      end
10     return C_S
11 end function

12 function isSmoothBlock(B):
13     if B = G then
14         return true
15     end
16     for p ∈ B.interlink do
17         if ¬isSmoothPointer(B, p) then
18             return false
19         end
20     end
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
29         if b = G then
30             return false
31         end
32         b ← Block(b.prevId)
33     end
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
7      return interlink
8  end function
```

REFERENCES

[1] Kiayias A., Miller A., and Zindros D. 2017. Non-interactive proofs of proof-of-work. *IACR Cryptology ePrint Archive* (2017). https://eprint.iacr.org/2017/963.pdf

[2] Zamyatin A., Stifter N., Judmayer A., Schindler P., Weippl E., and Knottenbelt W.J. 2019. A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice. *Zohar A. et al. (eds) Financial Cryptography and Data Security. FC 2018* (2019). https://eprint.iacr.org/2017/963.pdf

[3] Ittay Eyal and Emin Gun Sirer. 2013. Majority is not Enough: Bitcoin Mining is Vulnerable. (2013). arXiv:cs.CR/1311.0243

[4] 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.

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

Algorithm 3: Velvet Suffix Prover

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