NIPoPoWs under Velvet Fork

1 Introduction

Since the release of Bitcoin about a decade ago, the interest in cryptocurrencies has increased tremendously, while a number of other "altcoins" have been constructed in the meantime. Given that cryptocurrencies are starting to be considered a generally accepted means of payment and are used for everyday transactions, the issue of efficiently handling cryptocurrencies by light clients, such as smartphones, has become of great importance.

In this work, we consider the problem of optimizing light clients, or "SPV clients" as described in the original Bitcoin paper[5]. As blockchains are ever growing, the main setback for efficient light client applications is the processing of data amount linear to the size of the blockchain, e.g. for synchronization purposes.

Our work is based on the construction of Non-Interactive Proofs of Proof of Work[1] that achieves SPV proofs of polylogarithmic portion of the blockchain size. The NIPoPoWs construction suggests a protocol update, that could be possibly implemented by a soft or a hard fork. Given the reluctancy of the Bitcoin community to proceed to such forks, we consider the case of a velvet fork[1][2], where it suffices only a portion of the total players to be updated.

Under this scope, our contributions come as follows:

- We revise the security proof for NIPoPoWs suffix proof protocol and compute a concrete value for the security parameter m
- We describe an attack, that we name *Chainsewing Attack*, against NIPoPoWs suffix proof construction, which is used for the light client's synchronization
- We suggest a patch to the NIPoPoWs protocol that eliminates the *Chainsewing Attack* and prove its security

2 Model Definition and Notation

Our analysis concerns proof-of-work cryptocurrencies and is based on the standard Backbone model[4].

2.1 Blockchain Preliminaries

A blockchain, or simply chain, is a timely ordered sequence of blocks. In a cryptocurrecny blockchain, like Bitcoin, a block is a proof-of-work verified set of information about a number of transactions, the previous block in the block sequence and a nonce. The proof-of-work involves a computation over a cryptographic puzzle. More specifically, it involves scanning for a value called nonce, that when included in the block the total hash of the block results to a value lower than a certain threshold.

More formally, let $G(\cdot)$, $H(\cdot)$ be cryptographic hash functions. A *block* is a triple of the form $B = \langle s, x, ctr \rangle$, where s is the previous block *id*, x is the transactions information and $ctr \in \mathbb{N}$, such that satisfy the predicate $validBlock^T(B)$ defined as

$$H(ctr, G(s, x)) < T \tag{1}$$

The threshold parameter $T \in \mathbb{N}$ is called the block's *difficulty level.* Throughout this work we consider a constant value for the threshold T, although this is not the case in a real proof-of-work blockchain.

The rightmost block is the *head* the chain and is called the *Genesis* block often denoted G, while the whole chain is denoted C. So a chain C with $G = \langle s, x, ctr \rangle$ can be extended by appending a block $B = \langle s', x', ctr' \rangle$ as long as it holds that $s' = H(ctr, G(s, x))$. In effect every block is connected to the previous block in the chain by containing its hash. This is called the *prevId* relationship.

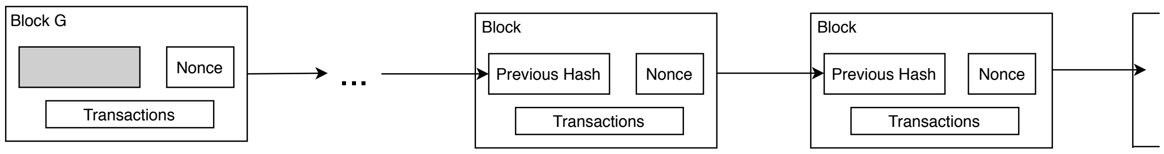

Figure 1: *A high-level representation of a blockchain.*

Figure 1 provides a high level representation of a blockchain including the bootstrap step of the very first block in the chain, where instead of the *prevId*, arbitrary data may be included in *s*.

Consider a peer-to-peer network where each party may have one of the following three roles: lightweight *clients*, full *nodes* and *miners*. Miners maintain an updated copy of the chain locally, while providing computational power, also called hashpower, to extend it. In order to extend the chain by one block, the miner has to perform a proof-of-work as already described. Full nodes can be thought of as miners with zero hashpower. Full nodes are also called *provers*, since they provide proofs answering the queries for specific chain information made by lightweight clients, according to Simplified Payment Verification (SPV) described by Nakamoto[5].

According to SPV scheme lightweight clients only need to store the block headers of the longest valid chain. A block header includes only a Merkle Tree Root of the Merkle Tree comprised by the transactions included in that specific block. In order to validate that a transaction is finalized, a client needs to query the nodes until he is convinced that he has the longest valid chain, search for the block containing that transaction and finally verify an inclusion proof of the transaction in the block of interest.

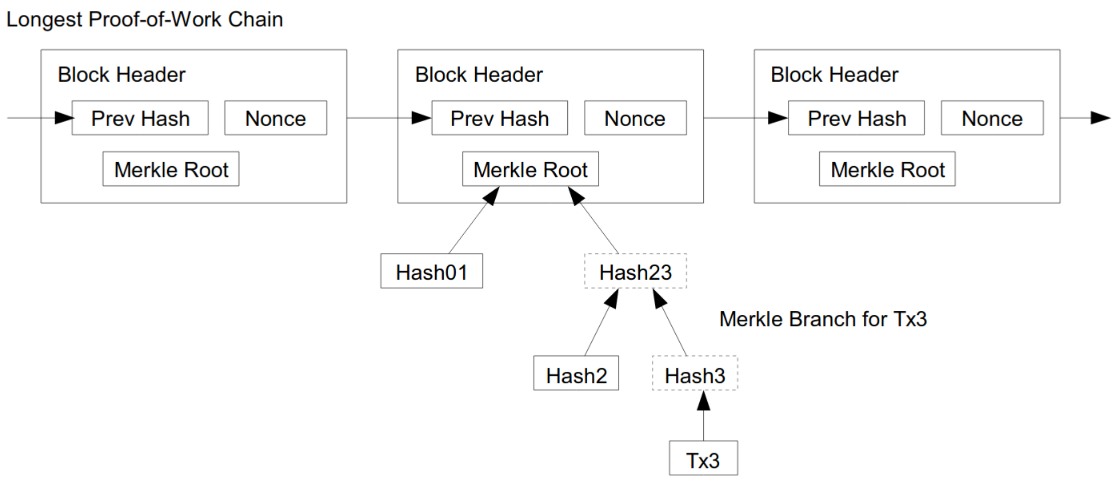

Figure 2: *High level representation of blockchain data kept by a lightweight client and an inclusion proof for a transaction Tx3.[5]*

In the SPV scheme a client needs to store blockchain data of linear size to the whole chain. By the time of writing Bitcoin's blockchain counts to almost 264GB and is estimated to grow more than 50GB per year. Since the growth rate of the chain is rather linear and constant, we need to construct more efficient protocols serving the needs of lightweight clients. To this end, the interaction between lightweight clients and full nodes is in our case supported by the NIPoPoWs[1] primitive which allows polylogarithmic poofs to the size of the chain.

2.2 The Backbone Model

The Backbone protocol is executed by an arbitrary number of parties over an unauthenticated network. We consider n parties in total, t of which may be controlled by an adversary.

Table 1 contains all the parameters of the Backbone protocol and will be a point of reference throughout this work.

λ : security parameter
κ : length of the hash function output
n : number of parties mining, t of which are controlled by the adversary
T : the target hash value used by parties for solving POW
t : number of parties controlled by the adversary
δ : advantage of honest parties, $\frac{t}{n-t} \leq 1-\delta$
f : probability at least one honest party succeeds in finding a POW in a round
ϵ : random variables' quality of concentration in typical executions
k : number of (suffix) blocks for the common prefix property
l : number of blocks for the chain quality property
μ_Q : chain quality parameter
s : number of rounds for the chain growth property
τ : chain growth parameter
L : the total run-tiem of the system

Table 1: The parameters of backbone model analysis. Positive integers $n, t, L, s, l, T, k, \kappa$, positive reals $f, \epsilon, \delta, \mu_Q, \tau, \lambda$ where $f, \epsilon, \delta, \mu_Q \in (0, 1)$.

We will now give a high-level description of the Backbone Protocol and its fundamental components, namely the three supporting algorithms for *chain validation*, *chain comparison* and *proof of work*. We will also define and discuss the three properties of the protocol, namely *Common Prefix*, *Chain Quality* and *Chain Growth*. For a more formal and detailed presentation refer to the Backbone paper[4].

Consider that the protocol has already run for some rounds and a chain C has been formed. Consider also an honest party that wishes to connect to the network, obtain the up-to-date version of the chain and try to extend it. The honest party connects to the network and first tries to synchronize to the current chain. The chain synchronization takes two steps to conclude. First, the newly connected peer receives a number of candidate chains by other peers in the network and validates them one by one as for the structural properties of each block *(Chain Validation)*. In particular, for each block the chain validation algorithm checks that the proof-of-work is properly solved, that the hash of the previous block is properly included in the block and that the the rest of the information included satisfies a certain validity predicate $V(\cdot)$ depending on the application. For example, in Bitcoin application it is checked that all the included transactions are valid according to the UTXO set.

Afterwards, the *Chain Comparison* algorithm is applied, where all the valid chains are compared to each other and the longest one, as for total number of blocks or total hashing power included, is considered the current active chain.

At last, in order to expand the chain by appending one more block to it, the *Proof Of Work* algorithm is applied, where the miner attempts to solve a proof of work as follows. The miner constructs the contents of the block, including the hash of the previous block and a number of new transactions published to the network. Consider that he can calculate the value $h = G(s, x)$ up to this point. Finally it remains to compute the ctr value so that $H(ctr, h) < T$. The protocol is running in rounds and each party can make at most q queries to function $H(\cdot)$ within a single round. If a suitable ctr is found, an honest party quits any queries remaining and announces the new born block to the network.

We can now define the three desired properties of the backbone protocol.

Definition 1. Common Prefix Property *The common prefix property Q_{cp} with parameter $k \in \mathbb{N}$ states that for any pair of honest players P_1, P_2 adopting the chains C_1, C_2 at rounds $r_1 \leq r_2$ respectively, it holds that $C_1^{\lceil k} \preceq C_2$.*

Definition 2. Chain Quality Property *The chain quality property Q_{cq} with parameters $\mu_{cq} \in \mathbb{R}$ and $l \in \mathbb{N}$ states that for any honest party P with chain $\mathcal{C}$ it holds that for any l consecutive blocks of $\mathcal{C}$ the ratio of honest blocks is at least μ_{cq}.*

Definition 3. Chain Growth Property *The chain growth property Q_{cg} with parameters $\tau \in \mathbb{R}$ and $s \in \mathbb{N}$ states that for any honest party P with chain $\mathcal{C}$, it holds that for any s rounds there are at least $\tau \cdot s$ blocks added to the chain of P*

2.3 NIPoPoWs Preliminaries

2.4 Hard, Soft and Velvet Forks

We typically describe the two common types of a blockchain permanent fork as follows.

A *hard fork* is a consensus protocol upgrade which is not backwards compatible. This means that the changes in the protocol break the old rules since the block header's contents change. After a hard fork blocks generated by upgraded players are not accepted by the unupgraded ones. In order the protocol update to be well established, the majority of the players must be upgraded at an early point or else the non-upgraded players may maintain the longest chain under the old rules.

A *soft fork* is a consensus protocol upgrade which is backwards compatible. This is usually implemented by keeping the old rules while adding additional information in a way that unupgraded players can ignore as comments, for example, by adding adding data in the coinbase transaction. In this way unupgraded players accept blocks generated by upgraded miners as valid, while, typically, unupgraded blocks are not accepted by upgraded players. Players are motivated to upgrade in order their blocks to be accepted in the chain as valid.

A *velvet fork* is also a backwards compatible consensus protocol upgrade. Similar to soft fork additional data can be inserted in the coinbase transaction. A velvet fork requires any block compliant to the old protocol rules only to be accepted as valid by both unupgraded and upgraded players. By requiring upgraded miners to accept all blocks, even if they contain false data according to the new protocol rules, we do not modify the set of accepted blocks. Therefore, the upgrade is rather a *recommendation* and not an actual change of the consensus protocol. In reality, the blockchain is never forked. Only the codebase is upgraded and the data on the blockchain is interpreted differently[1].

The goal of this work is to provide a modified NIPoPoWs protocol so that it can be deployed under a velvet fork in a provably secure manner.

3 NIPoPoWs under Soft or Hard Fork

3.1 Suffix Proofs

NIPoPoWs suffix proofs are used to prove predicates that pertain to the suffix of the blockchain. For example, this is the case of light client synchronization to the longest valid chain. [...]

3.1.1 Security of Suffix Proofs

In this section we provide the full security proof for the NIPoPoWs suffix proof protocol[1]. Apart from the proof itself (Theorem 2), we describe the definitions and lemmas being used. We try to give intuition for arguments and conclusions in each step.

Assume t adversarial out of n total parties, each with q PoW random oracle queries per round. We define $p = \frac{T}{2^\kappa}$ the probability of a successful Random Oracle query. We will call a query to the RO *μ-successful* if the RO returns a value h such that $h \leq 2^{-\mu}T$.

We define the boolean random variables X_r^μ, Y_r^μ, Z_r^μ. Fix some round r, query index j and adversarial party index k (out of t). If at round r an honest party obtains a PoW with $id < 2^{-\mu}T$, set $X_r^\mu = 1$, otherwise $X_r^\mu = 0$. If at round r exactly one honest party obtains $id < 2^{-\mu}T$, set $Y_r^\mu = 1$, otherwise $Y_r^\mu = 0$. If at round r the j-th query of the k-th corrupted party is μ-successful, set $Z_{rjk}^\mu = 1$, otherwise $Z_{rjk}^\mu = 0$. Let $Z_r^\mu = \sum_{k=1}^{t} \sum_{j=1}^{q} Z_{rjk}^\mu$. For a set of rounds S, let $X^\mu(S) = \sum_{r \in S} X_r^\mu$ and similarly define Y_S^μ, Z_S^μ.

Definition 4. Typical Execution. An execution of the protocol is (ϵ, η)-typical if:

Block counts don't deviate. For all $\mu \geq 0$ and any set S of consecutive rounds with $|S| \geq 2^\mu \eta k$, we have:

- $(1-\epsilon)E[X^\mu(S)] < X^\mu(S) < (1+\epsilon)E[X^\mu(S)]$ and $(1-\epsilon)E[Y^\mu(S)] < Y^\mu(S)$
- $Z^\mu(S) < (1+\epsilon)E[Z^\mu(S)]$

Round count doesn't deviate. Let S be a set of consecutive rounds such that $Z^\mu(S) \geq k$ for some security parameter k. Then $|S| \geq (1-\epsilon)2^\mu \frac{k}{pqt}$ with overwhelming probability.

Chain regularity. No insertions, no copies and no predictions [4] have occurred.

Theorem 1. ***Typicality*** *Executions are* (ϵ, η)*-typical with overwhelming probability in* k*.*

Proof. **Block counts and regularity.** We refer to [4] for the full proof.

Round count. First, observe that for a specific round r we have $Z_{rjk} \sim Bern(p)$, so for the μ-level superblocks $Z_{rjk}^\mu \sim Bern(2^{-\mu}p)$ and these are jointly independent. Therefore, since for $|S|$ rounds we have $tq|S|$ adversarial RO queries, we have that $Z_S^\mu \sim \text{Bin}(tq|S|, 2^{-\mu}p)$. So $tq|S| \sim \text{NB}(Z_S^\mu, 2^{-\mu}p)$. Negative Binomial distribution is defined as $\text{NB}(r, p')$ and expresses the number of trials in a sequence of independent and identically distributed Bernoulli trials before a specified (r) number of successes occurs. The expected total number of trials of a negative binomial distribution with parameters (r, p') is r/p'. To see this, imagine an experiment simulating the negative binomial performed many times, that is a set of trials is performed until r successes occur. Consider you perform n experiments of total N trials. Now we would expect $Np' = nr$, so $N/n = r/p'$. See that N/n is just the average number of trials per experiment. So we have $E[tq|S|] = \frac{Z_S^\mu}{2^{-\mu}p} \Rightarrow E[|S|] = 2^\mu \frac{Z_S^\mu}{tqp}$. So if $Z^\mu(S) \geq k$ then $E[|S|] \geq 2^\mu \frac{k}{tqp}$. Applying a tail bound to the negative binomial distribution, we obtain that $\Pr[|S| < (1-\epsilon)E(|S|)] \in \Omega(\epsilon^2 m)$.

Lemma 1. *Suppose S is a set of consecutive rounds $r_1 ... r_2$ and C_B is a chain adopted by an honest party at round r_2 of a typical execution. Let $C_S^B = \{ b \in C_B : b$ was generated during $S \}$. Let $\mu_A, \mu_B \in \mathbb{N}$. Suppose $C_S^B \uparrow^{\mu_B}$ is good. Suppose C'_A is a μ_A-superchain containing only adversarially generated blocks generated during S and suppose that $|C'_A| \geq k$. Then $2^{\mu_A}|C'_A| < 2^{\mu_B}|C_S^B \uparrow^{\mu_B}|$.*

Proof. From $|C'_A| \geq k$ we have that $|Z_S^{\mu_A}| \geq k$. Applying Theorem 1, we conclude that $|S| \geq (1-\epsilon')2^{\mu_A}\frac{|C'_A|}{pqt}$. Applying the Chain Growth theorem [4] we obtain $|C_B^S| \geq (1-\epsilon)f|S|$. But from the goodness of $C_B^S \uparrow^{\mu_B}$, we know that $|C_B^S \uparrow^{\mu_B}| \geq (1-\delta)2^{-\mu_B}|C_B^S|$. So we have $|C_B^S \uparrow^{\mu_B}| \geq (1-\delta)2^{-\mu_B}(1-\epsilon)f|S|$ and follows that $|C_B^S \uparrow^{\mu_B}| \geq (1-\delta)2^{-\mu_B}(1-\epsilon)f(1-\epsilon')2^{\mu_A}\frac{|C'_A|}{pqt}$. Consequently we have that $2^{\mu_A}|C'_A| \leq \frac{pqt}{(1-\delta)(1-\epsilon)(1-\epsilon')f}2^{\mu_B}|C_B^S \uparrow^{\mu_B}|$.

So, according to the above equation we have that $2^{\mu_A}|C'_A| < 2^{\mu_B}|C_S^B \uparrow^{\mu_B}|$ considering that honest majority assumption holds, specifically considering that $\frac{pqt}{f} \approx \frac{t}{n-t} \leq 1$.

Definition 5. Adequate level of honest proof *Let π be an honestly generated proof constructed upon some adopted chain C and let $b \in \pi$. Then μ' is defined as $\mu' = \max\{\mu : |\pi\{b :\} \uparrow^{mu}| \geq \max(m+1, (1-\delta)2^{-\mu}|\pi\{b :\} \uparrow^{\mu}\downarrow|)\}$. We call μ' the adequate level of proof π with respect to block b with security parameters δ and m. Note that the adequate level of a proof is a function of both the proof π and the chosen block b.*

Intuitively, adequate is the level μ' of a proof π for a block b if there are at least m blocks after b in π under the condition that there is good chain quality for this level, meaning that there are at least so many blocks at this level as expected considering the number of 0-level blocks.
NOTE: adequate level is mostly useful for Claim 1a of the Security Proof (Theorem 2).

Lemma 2. *Let π be some honest proof generated with security parameters δ, m. Let C be the underlying chain, $b \in C$ be any block and μ' be the adequate level of the proof with respect to b and the same security parameters.*
Then $C\{b:\}\uparrow^{\mu'} = \pi\{b:\}\uparrow^{\mu'}$.

Proof. $\pi\{b:\}\uparrow^{\mu'} \subseteq C\{b:\}\uparrow^{\mu'}$ is trivial. For the converse, we have that in the iteration of the *Prove for loop*[1] with $\mu = \mu^*$, the block stored in variable B precedes b in C.

Note that the Prover's for loop iterates over all levels in the interlink structure, and places in the proof all of the blocks that are of the corresponding level and succeed B in C.

Suppose $\mu = \mu^*$ is the first *for* iteration during which the property is violated. This cannot be the first iteration since $B = C[0]$ and Genesis precedes all blocks. By induction hypothesis we see that during the iteration $\mu = mu^* + 1$, B preceded b. From the definition of μ' we know that μ' is the highest level for which $|\pi\{b:\}\uparrow^{\mu}| \geq \max(m, (1-\delta)2^{-\mu}|\pi\{b:\}\uparrow^{\mu}\downarrow|)$.

Hence, this property cannot hold for $\mu^* > \mu$ and therefore $|\pi\{b:\}\uparrow^{\mu}| < m$ or $\neg$local-good$_\delta(\pi\{b:\}\uparrow\mu^*[1:], C, \mu^*)$.

In case local-good is violated, variable B remains unmodified and the induction step holds. If local-good is not violated, then $|\pi\{b:\}\uparrow^{\mu^*}[1:]| < m$ and so $\pi\uparrow^{\mu^*}[-m]$, which is the updated value of B at the end of μ^* iteration, precedes b.

Lemma 3. *Suppose the verifier evaluates $\pi_A \geq \pi_B$ in a protocol interaction where B is honest and assume during the comparison that the compared level of the honest party is μ_B. Let $b = LCA(\pi_A, \pi_B)$ and let μ'_B be the adequate level of π_B with respect to b. Then $\mu'_B \geq \mu_B$.*

Proof. Because μ_B is the compared level of the honest party, from the definition of the $\geq_m$ operator, we have $2^{\mu_B}|\pi\{b:\}\uparrow^{\mu_B}| > 2^{\mu'_B}|\pi\{b:\}\uparrow^{\mu'_B}|$. This is true, otherwise the Verifier would have chosen level μ'_B as level of comparison. The proof is by contradiction. Suppose $\mu'_B < \mu_B$. By definition, μ'_B is the maximum level such that $|\pi_B\{b:\}\uparrow^{\mu}[1:]| \geq max(m, (1-\delta)2^{-\mu}|\pi_B\{b:\}\uparrow^{\mu}[1:]\downarrow|)$, therefore μ_B does not satisfy this condition. But we know that $|\pi_B\{b:\}\uparrow^{\mu}[1:]| > m$ because μ_B was selected by the Verifier. Therefore $2^{\mu_B}|\pi\{b:\}\uparrow^{\mu_B}| < (1-\delta)|C\{b:\}|$.
But also μ'_B satisfies goodness, so $2^{\mu'_B}|\pi\{b:\}\uparrow^{\mu'_B}| > (1-\delta)|C\{b:\}|$.
From the last two equations we obtain $2^{\mu_B}|\pi\{b:\}\uparrow^{\mu_B}| < 2^{\mu'_B}|\pi\{b:\}\uparrow^{\mu'_B}|$ which contradicts the initial equation.

@To Be Discussed: would the verifier ever choose a non-adequate level for proof comparison? Intuitively the above Lemma says: the comparison level chosen by the Verifier can be no other than the adequate level in respect to block b ($LCA(\pi_A, \pi_B)$), since any other choice would be a level of non-good quality, because of the definition of the adequate level. A level of non-good quality would contain less PoW than that of the adequate level for the range of interest $C\{b:\}$.

Theorem 2. ***Security of suffix proofs*** *Assuming honest majority, the non-interactive proofs-of-proof-of-work construction for computable κ-stable monotonic suffix-sensitive predicates is secure with overwhelming probability in κ.*

Proof. By contradiction. Let Q be a $\kappa-$stable monotonic suffix-sensitive chain predicate. Assume NIPoPoWs on Q is insecure. Then, during an execution at some round r_3, $Q(C)$ is defined and the verifier V disagrees with some honest participant. Assume the execution is typical. V communicates with adversary A and honest prover B. The verifier receives proofs π_A, π_B. Because B is honest, π_B is a proof constructed based on underlying blockchain C_B (with $\pi_B \subseteq C_B$), which B has adopted during round r_3 at which π_B was generated. Furthermore, π_A was generated at round $r'_3 \leq r_3$.

The verifier outputs $\neg Q(C_B)$. Thus it is necessary that $\pi_A \geq \pi_B$. We will show that this is a negligible event.

Let $b = LCA(\pi_A, \pi_B)$. Let $b^\star$ be the most recently honestly generated block in C_B preceding b. Note that $b^\star$ necessarily exists because Genesis is honestly generated. Let the levels of comparison decided by the verifier be μ_A and μ_B respectively. Let μ'_B be the adequate level of proof π_B with respect to block b. Call $\alpha_A = \pi_A \uparrow^{\mu_A} \{b :\}$, $\alpha'_B = \pi_B \uparrow^{\mu'_B} \{b :\}$.

Note that we consider the parts of the proofs succeeding block b the decisive ones for the verifier's choice. This is to adversary's advantage, since the parts preceding this block demonstrate the proof-of-work contained in the common (sub)chain, thus the adversary could only include equal or less proof-of-work in her proof for this part of the chain.

We will now show three successive claims: First, α_A and $\alpha'_B \downarrow$ are mostly disjoint. Second, a_A contains mostly adversarially generated blocks. And third, the adversary is able to produce this a_A with negligible probability.

Let $\alpha_A = k_1 + k_2 + k_3$ and let k_1, k_2, k_3 be as defined in the following Claims.

Claim 1: $\alpha_A, \alpha'_B \downarrow$ are mostly disjoint. We show this by taking the two possible cases for the relation of μ_A, μ'_B.

Claim 1a: If $\mu'_B \leq \mu_A$ then they are completely disjoint. In such a case of inequality, every block in α_A would also be of lower level μ'_B. Applying Lemma 2 to $C\{b :\} \uparrow^{\mu'_B}$ we see that $C\{b :\} \uparrow^{\mu'_B} = \pi\{b :\} \uparrow^{\mu'_B}$. Subsequently, any block in $\pi_A \uparrow^{\mu_A} \{b :\}[1 :]$ would also be included in proof α'_B, but $b = LCA(\pi_A, \pi_B)$ so there can be no succeeding block common in α_A, α'_B.

Claim 1b: If $\mu'_B > \mu_A$ then $|\alpha_A[1 :] \cap \alpha'_B \downarrow [1 :]| = k_1 \leq 2^{\mu'_B - \mu_A}$.
First observe that because the adversary is winning $2^{\mu_A}|\alpha_A| > 2^{\mu'_B}|\alpha'_B| \geq 2^{\mu'_B} m \Rightarrow |\alpha_A| > 2^{\mu'_B - \mu_A} m$. Let's call b_1 the first block in α'_B after block b. Suppose for contradiction that $k_1 > 2^{\mu'_B - \mu_A}$. Since $C_B^{\mu'_B}$ is of good chain quality, this would mean that block b_1, of level μ'_B is also of level μ_A. Since it is of level μ_A the adversary could include it in the proof but b_1 cannot exist in both α_A, α'_B since $\alpha_A \cap \alpha'_B = \emptyset$ by definition. In case that the adversary chooses not to include b_1 in the proof than she can include no other blocks of C_B in her proof, since it would not consist a valid chain.

From Claim $1a$ and Claim $1b$, we conclude that there are $|\alpha_A| - k_1$ blocks after block b in α_A which do not exist in $\alpha_B \downarrow$. We now set $b_2 = LCA(C_B, \alpha_A)$. This makes b_2 the last block before the fork point at the 0-level chain included in the adversary's proof.

Intuition: in this case the common blocks of $\alpha_A, \alpha'_B \downarrow$ may only be blocks of level μ_A which precede the first μ'_B block appearing in α'_B. If this block of level μ'_B was common, it could also be included in α_A. If it is included this would be the LCA of α_A, α'_B. If it is not, then the adversary could no more include blocks from the common part of chain C_B in her proof since they no longer form a valid chain in α_A. The quantity $2^{\mu'_B - \mu_A}$ means: in the range between two consequent μ'_B-level blocks, we have $n = 2^{\mu'_B}$ 0-level blocks and, thus, $2^{-\mu_A} n = 2^{\mu'_B - \mu_A}$ blocks of μ_A-level.

Claim 2: At least k_3 superblocks of α_A are adversarially generated. We show this by showing that $\alpha_A[k_1 + k_2 + 1 :]$ contains no honestly generated blocks. Suppose for contradiction that the block $\alpha_A[i]$ for some $i \geq k_1 + k_2 + 1$ was honestly generated. This means that an honest party adopted the chain $\alpha_A[: i-1] \downarrow$ at some round $r_2 \leq r_3$. Because of the way honest parties adopt chains, the superchain $\alpha_A[: i-1]$ has an underlying properly constructed 0-level anchored chain C_A such that $\alpha_A[: i-1] \subseteq C_A$. Let j be the index of block b_2 within α_A, $j_\downarrow$ be the index of block b_2 within C_A and $k_{2\downarrow} = |\alpha_A[j : j + k_2] \downarrow|$. See Figure 3 for a demonstration. Observe that $|C_A[: \{\alpha_A[i-1]\}]| \geq |C_A[: j_\downarrow + k_{2\downarrow}]|$, while $C_A[j_\downarrow : j_\downarrow + k_{2\downarrow}] \npreceq C_B$ as proved in Claim 1. But C_A was adopted by an honest party at round r_2, which is prior to round r_3 during which C_B was adopted by an honest party B. This contradicts the Common Prefix[4] with parameter $k_{2\downarrow}$. I t follows that with overwhelming probability in $k_{2\downarrow}$, the $k_3 = |\alpha_A| - k_2 - k_1$ last blocks of the adversarial proof have been adversarially generated.

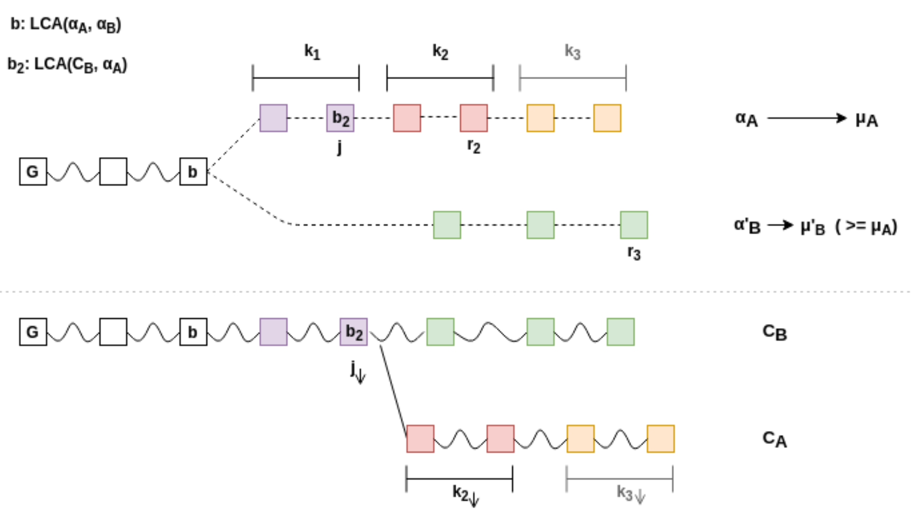

Figure 3: *Two competing proofs at different levels. At the bottom the corresponding 0-level chains are represented.*

Intuition: Because of Common Prefix on $k_{2\downarrow}$ parameter, where $k_{2\downarrow} = |\alpha_A[j : j + k_2] \downarrow|$, where $E[k_{2\downarrow}] = 2^{\mu_A} k_2$, there can be no honest party adopting C_A at any round $i \geq k_1 + k_2 + 1$.

From these two Claims we have that k_1 blocks in α_A are blocks of the common zero-level chain, while k_2 blocks are blocks after the fork point at the zero-level chain. Subsequently, k_2 is subject to the Common Prefix $\kappa-$parameter limitations as described in the Backbone paper[4].

Claim 3: Adversary A is able to produce α_A that wins against α_B with negligible probability.
Let b' be the latest honestly generated block in a_A, or $b' = b^*$ if no such block exists in a_A. Let r_1 be the round when b' was generated. Consider the set S of consecutive rounds $r_1..r_3$. Every block in $\alpha_A[-k_3 :]$ has been adversarially generated during S and $|\alpha_A[-k_3 :]| = |\alpha_A\{b' :\}| = k_3$. C_B is a chain adopted by an honest party at round r_3 and filtering the blocks by the rounds during which they were generated to obtain C_B^S, we see that if b'' is the most recently generated block in α_B in a round $r \leq r_1$, then $C_B^S = C_B\{b'' :\}$. But $C_B^S \uparrow^{\mu'_B}$ is good with respect to C_B^S. Applying Lemma 1, we obtain that with overwhelming probability $2^{\mu_A}|\alpha_A\{b' :\}| < 2^{\mu'_B}|C_B^S \uparrow^{\mu'_B}|$, which is equal to

$$2^{\mu_A}|\alpha_A\{b' :\}| < 2^{\mu'_B}|\alpha'_B\{b'' :\}| \tag{2}$$

since α'_B contains all the μ'_B-level blocks in C_B^S.

In order to complete the proof, let us now consider $\alpha_A^{k_1}$, $\alpha_A^{k_2}$, $\alpha_A^{k_3}$ the parts of α_A where the k_1, k_2, k_3 blocks reside and $\alpha_B^{k_1}$, $\alpha_B^{k_2}$, $\alpha_B^{k_3}$ the parts of α_B containing blocks generated in the corresponding round sets as illustrated in Figure 4.
Subsequently to the above Claims we have that:
Because of the common underlying chain in the first round set:

$$2^{\mu_A}|\alpha_A^{k_1}| \leq 2^{\mu'_B}|\alpha'^{k_1}_B| \tag{3}$$

Because of the adoption by an honest party of chain C_B at a later round r_3, we have for the second round set:

$$2^{\mu_A}|\alpha_A^{k_2}| \leq 2^{\mu'_B}|\alpha'^{k_2}_B| \tag{4}$$

Because of Equation (1), we have for the third round set:

$$2^{\mu_A}|\alpha_A^{k_3}| < 2^{\mu'_B}|\alpha'^{k_3}_B| \tag{5}$$

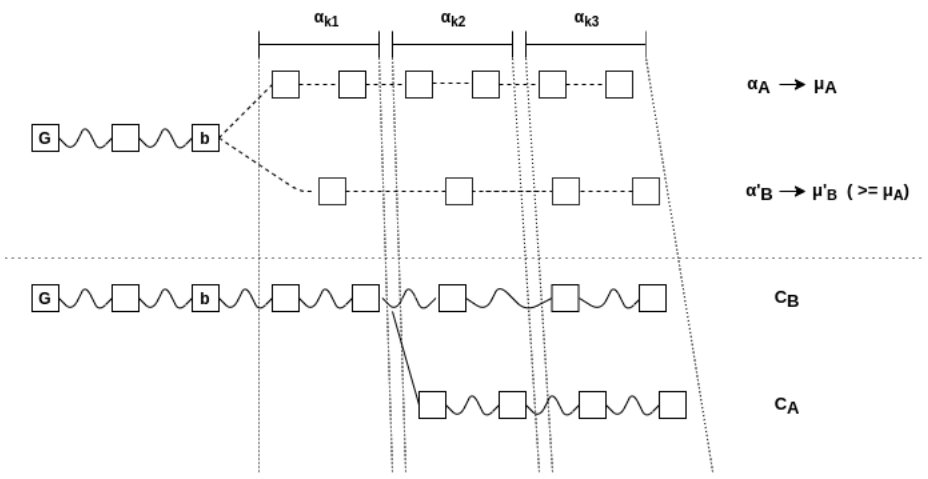

Figure 4: *The three round sets in two competing proofs at different levels. The vertical dashed lines denote the area of interest, across proofs and chains, corresponding to each round set. At the bottom the corresponding 0-level chains are represented.*

So we have

$$2^{\mu_A}(|\alpha_A^{k_1}| + |\alpha_A^{k_2}| + |\alpha_A^{k_3}|) < 2^{\mu'_B}(|\alpha'^{k_1}_B| + |\alpha'^{k_2}_B| + |\alpha'^{k_3}_B|)$$

and finally

$$2^{\mu_A}|\alpha_A| < 2^{\mu'_B}|\alpha'_B| \tag{6}$$

Therefore we have proven that $2^{\mu'_B}|\pi_B \uparrow^{\mu'_B}| > 2^{\mu_A}|\pi_A^{\mu_A}|$. From the definition of μ_B, we know that $2^{\mu_B}|\pi_B \uparrow^{\mu_B}| > 2^{\mu'_B}|\pi_B \uparrow^{\mu'_B}|$ because it was chosen μ_B as level of comparison by the Verifier. So we conclude that $2^{\mu_B}|\pi_B \uparrow^{\mu_B}| > 2^{\mu_A}|\pi_A \uparrow^{\mu_A}|$.

□

It remains to compute the security parameter m that guarantee that all the above hold true in every implementation. It suffices to compute a security parameter value for each set of rounds k_1, k_2, k_3, so that the proof equations 3, 4, 5 hold and then sum these values to obtain parameter m.

In the first set of rounds, for the first k_1 blocks in α_A, we only need one block included in α_B for the part of the proof described in Equation 3. In the second set of rounds we need $2^{-\mu_B}\kappa$ blocks for the part of the proof described in Equation 4, just as it directly results from the Common Prefix property. In order to make m independent of any specific level it suffices to consider the upper bound of κ blocks for this set of rounds. In the last set of rounds we need at least κ adversarially generated blocks in $\alpha_A^{k_3}$ so that Lemma 1 is applicable. Since we assume honest majority, obliging to at least κ blocks for this set of rounds suffices to guarantee for Equation 5.

So, we finally conclude to the following upper bound for the value of the security parameter:

$$m = 2\kappa + 1 \tag{7}$$

3.2 Infix Proofs

3.3 Succinctness

4 NIPoPoWs under Velvet Fork

4.1 *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 6. 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 5.

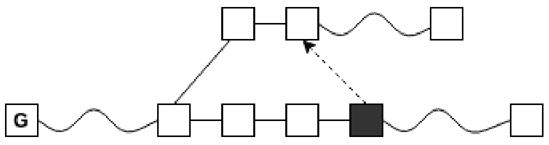

Figure 5: *A non-smooth pointer of an adversarial block, colored black, in an honest player's chain.*

In the same manner it is possible that a false interlink contain arbitrary pointers to blocks of any chain as illustrated in Figure 6. The interlink pointing to arbitrary directions resembles a thorny bush, so we will refer to blocks containing false interlink information as *thorny.*

Definition 7. Thorny Block *A thorny block is a block which contains at least one non-smooth interlink pointer.*

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

4.2 Suffix Proofs

4.2.1 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.

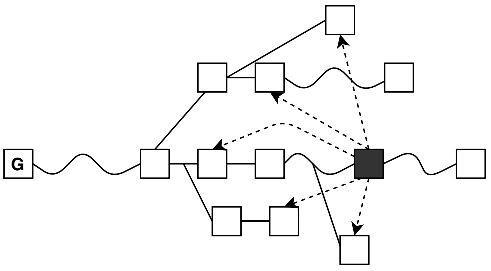

Figure 6: *A thorny block appended in an honest player's chain. The dashed arrows are interlink pointers.*

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 7. 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.

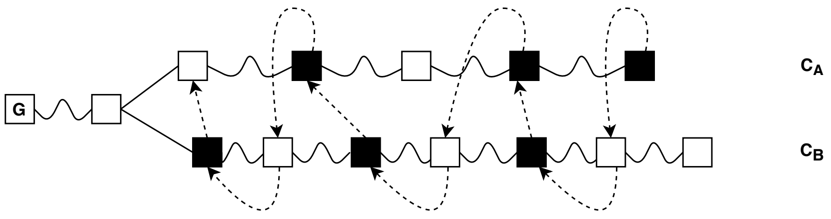

Figure 7: *Generic Chainsewing Attack. C_B is the chain of an honest player and C_A the adversary's chain. Adversarially generated blocks are colored black. Dashed arrows represent interlink pointers included in the adversary's suffix proof. Wavy lines imply one or more blocks.*

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 8 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.

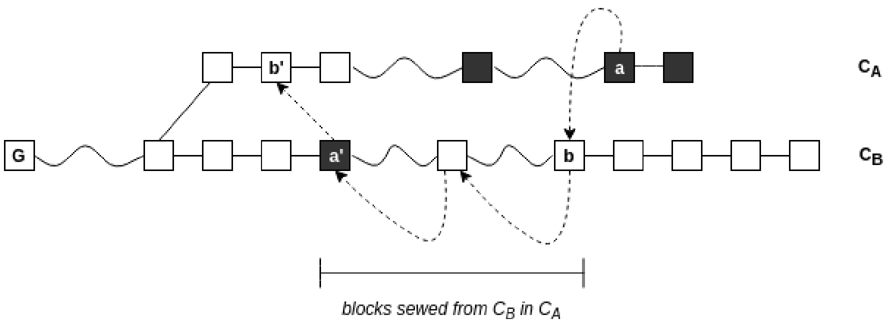

Figure 8: *Chainsewing Attack. C_B represents the chain of an honest player. C_A is an adversarial fork. Adversarially generated blocks are colored black. Dashed arrows represent interlink pointers included in the adversary's suffix proof. Wavy lines imply one or more blocks. Firm lines imply the previousId 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.

4.3 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 a forked chain, which are in majority adversarially generated due to the Common Prefix property, but also an arbitrarily long part of the chain adopted by an honest player. Since thorny blocks are accepted as valid, the verifier cannot distinguish blocks that actually belong in a chain from blocks that only seem to belong in the same chain because they are pointed to via a non-smooth pointer.

We will first define our security assumptions and then discuss some solution approaches for the problem. This discussion will result to the proposed protocol upgrade suggestion.

4.3.1 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 9. 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 it holds that $n_h = g \cdot n$.*

4.3.2 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 1/3 of the upgraded honest miners. Therefore we define the Velvet Honest Majority assumption, which will be used in our security proof.

Definition 10. Velvet Honest Majority *Let n_v be the number of upgraded miners. Then t out of total n_v parties are corrupted such that $\frac{t}{n_v} < \frac{1-\delta}{3}$.*

We believe that if we waive this requirement it is impossible to have a provably secure NIPoPoW protocol under velvet fork conditions. This claim is described and discussed in the following. Note that any reference to adversarial party bound is respect to the number of upgraded parties n_v. Therefore, "(1/2)-bounded adversary" holds for $\frac{t}{n_v} < \frac{1-\delta}{2}$, while "(1/3)-bounded adversary" holds for $\frac{t}{n_v} < \frac{1-\delta}{3}$.

4.3.3 Impossibility of a secure protocol for (1/2)-bounded adversary

During our study on the problem we failed to prove the security of several protocol constructions under $(\frac{1}{2})$-bounded adversary. We finally concluded that such a secure NIPoPoW construction is impossible under velvet fork conditions. Though it is hard to provide a typical proof for this claim, as one should consider any possible construction, we try to argue for it in this section.

Claim: *Assume t adversarial out of total n_v upgraded parties. There is no construction for NIPoPoW suffix proofs under velvet fork conditions, which is both succinct and secure for every adversary, such that $\frac{t}{n_v} < \frac{1-\delta}{2}$.*

Discussion. As explained earlier, since the adversary may use the interlink structure so as to include pointers to arbitrary blocks, she may construct her own chain history utilizing the non-smooth pointers included in the thorny blocks she generates. Such an example is given in Figure 7.

Let us consider a protocol construction p which allows for NIPoPoW suffix proofs under velvet fork and is secure for 1/2-bounded adversary. Our main goals are p to be both secure and succinct. We can even loose our non-interactivity limitations to make it possible to challenge the submitted proofs, so that an honest player can contest against an inconsistent proof. However, note that the same power to challenge any submitted proof is also provided to the adversary.

Now assume an adversary of $n_v/3 < t < n_v/2$. Then, as explained in the Backbone and Selfish Mining papers [4][3] it is possible for the adversary to maintain the longest valid chain being of less than 50% chain quality in favor of adversarially generated blocks. In particular, consider that the adversary mines selfish, meaning that for every block she mines she does not immediately announce it to the network. Instead, she waits until an honest block is announced. In the worst case, consider an adversary that has a network advantage regarding the relay of her blocks. Thus in case of two competing blocks in a round the adversarial one will be appended to the chain, while the honest block is discarded in practice as a temporary fork. Because of this attack the total block ratio in the chain will result as illustrated in Figure 9. For (1/3) adversary half of the blocks in the chain are expected to be adversarial, while for (1/2) adversary no honest block is included in the chain in the worst case scenario.

Observe that the property violated in the chainsewing attack is the *prevId* relation between sequential blocks. However since the main purpose of our work is to achieve succinctness, the *prevId* relations could not be utilized in a viable manner in order to contest adversarial proofs, as it would require proofs of linear length to that of the whole chain. We thus conclude that we should mostly rely on the information given by the interlink pointers as far as the validity of any proof is concerned so as to keep our protocol construction efficient.

Now assume, to honest parties' favor, that it is both possible and efficient to inspect the whole chain history that each block is commited to via its interlink pointers. Keep in mind that an adversary can keep a consistent chain considering the interlink pointers by using only her own mined blocks and, at the same time, these blocks may overwhelm the honestly generated ones.

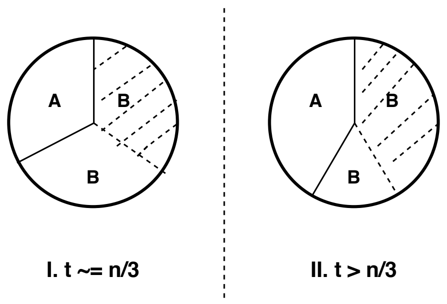

Figure 9: *Pie chart of adversarially and honestly generated blocks appended in the chain during a round set S. Part* ***A*** *stands for blocks mined by the adversary while* ***B*** *for blocks mined by honest players. Lined out parts denote honestly mined blocks that were defeated by adversarially mined ones in the same round due to selfish mining.****I.*** *With $t = n/3$, 50% of the total blocks are adversarially generated in the worst case scenario.* ***II.*** *With $t > n/3$, more than half of the total blocks are adversarially generated in the worst case scenario.*

In our protocol p it should be decided under what policy an honest party generates blocks and constructs suffix proofs. A decision should be made for block generation:

1. interlink data are neutral as for adversarially and honestly generated blocks
2. interlink data point out inconsistent blocks, meaning blocks with incorrect interlinks
3. interlink data exclude inconsistent blocks from being part of the valid chain formed by the super-pointers

Now let's examine the above choices.
In the first case the arguments that an honest player could provide against an inconsistent proof could only be based on the 0-level pointers. Such a contesting proof should at least provide the 0-level subchain of length equal to the following: starting from a block included in the suffix proof and is not a block of the chain until the closest block included in the suffix proof and is a valid block of the chain. This means that contesting proofs are expected to be of length 2^{μ} where $\mu = log(|C|)$, and subsequently are of complexity $\mathcal{O}(|C|)$ which ruins the succinctness of our protocol.

In the second case an honest party could utilize this extra information provided in the interlink to prove an inconsistent proof wrong. However, keep in mind that the adversary could make such claims too. So, a claim of inconsistency for a specific block included should be followed by a proof of its inconsistency. Whatever the information considering the incorrect blocks in the interlink may be, the adversary could make some of it in her own generated blocks in order to contest an honest proof too. So we fall to the previous case turning to *prevId* pointers in order to prove the true chain history, where the contesting proofs are unacceptable efficiency-wise.

In the third case we demand from the honest players to validate the interlink structures of the blocks in the chain and exclude the inconsistent ones from the chain history that the interlinks provide. In this way we could possibly provide contesting proofs using information only from the interlinks thus keeping our proofs succinct. This solution would result to two different chain histories considering the interlink pointers: the one of the honest parties that includes only blocks with valid interlinks, and the one of the adversary which may form her own history of invalid blocks probably belonging to many different underlying chains but could not include honest blocks if at least one invalid block participates in the proof. This solution seems to bring us to a notable trade-off point. On the one hand we manage to uncouple from the 0-level blocks. On the other hand, we compromise that honest players cannot use valid adversarially generated blocks in their proof, while the adversary cannot include honestly generated blocks in her proofs if inconsistent blocks are also included. This solution could not work properly for (1/2)-bounded adversary though. The reason lies in the bad chain quality as pointed out earlier and shown in Figure 9. If the adversary could create his own chain history including inconsistent blocks from two different chains, then she could easily construct a suffix proof that wins over an honest one because the majority of the blocks included in the chain could be adversarial with high probability.

We conclude that such a protocol could not exist for (1/2)-bounded adversary.

4.3.4 Solution for (1/3)-bounded adversary

The vulnerability that makes this attack possible is the acceptance of thorny blocks. Since we operate under a velvet fork we cannot eliminate such blocks, but we need, however, to restrict the adversary from being able to claim portion of another chain as part of her own fork chain.The key observation on the Chainsewing Attack is that the adversary needs at least one adversarially generated block in an honest player's chain (block a'), in order to create a path of superpointers connecting blocks of two, or more, diverse chains. The final proof chain will make use of this crossing point and may contain both honest or adversarial blocks. In case the proof contains only adversarial blocks, the attack cannot harm security for an adversary of less than 1/3 of the total hashing power. This fact will be clear in the security proof section. In short, as argued in the previous section an adversary of 1/3 of the total hashing power may contribute at most 50% of the total blocks in the longest valid chain. An attacker of more than 1/3 hashing power could dominate as of total hashing power expressed in mined blocks in the chain, while the inverse would be true for an attacker of less hashing power than this threshold.

So in order to be successful, the attacker needs to also "sew" honestly generated blocks. Thus there will be at least one honest block in the superblock path which connects a and a', pointing to an adversarial thorny block.

The idea is to ban all blocks generated by honest players from participating in this superblock path. In this way the adversary could not misuse hashing power of the honest players and the sewed blocks could only be adversarially generated, thus the attack could never succeed for an adversary of less than 1/3 of the total hashing power.

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 (except the pointers of level 0). 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 the chain adopted by an honest party 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 10 illustrates this remark.

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 $2t < (1-\delta)(n_v - t)$, where n_v is the number of upgraded players.
2. The NIPoPoW protocol under velvet fork works as usual but a miner constructs a block's interlink without pointers to thorny blocks.

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

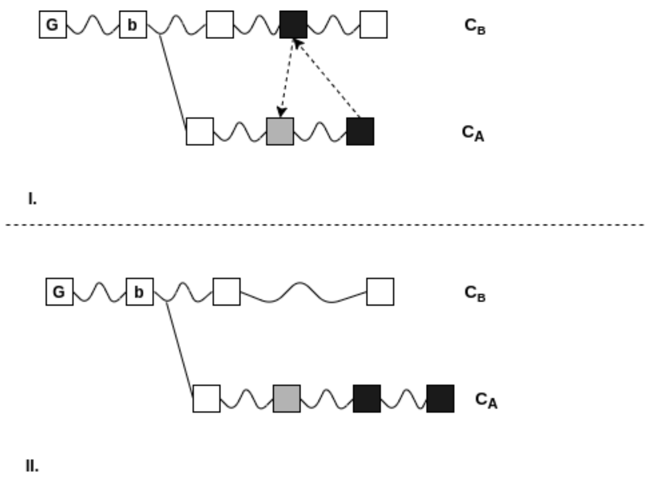

Figure 10: *The adversarial fork chain C_A and chain C_B of an honest player. Thorny blocks are colored black. Dashed arrows represent interlink pointers. Wavy lines imply one or more blocks. When an adversarially generated block is sewed from C_B into the adversary's suffix proof the verifier conceives 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.*

Lemma 4. *A velvet suffix proof constructed by an honest player cannot contain any thorny block.*

Lemma 5. *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 6. *Let $\mathcal{P}_\mathcal{A} = (\pi_\mathcal{A}, \chi_\mathcal{A})$ be a velvet suffix proof constructed by the adversary. Let b_t be the oldest thorny bock included in $\mathcal{P}_\mathcal{A}$ which is generated at round r_t. Then any block $b = \{b : b \in \mathcal{P}_\mathcal{A} \wedge b$ generated at $r \geq r_t\}$ is thorny.*

Proof. By contradiction. Suppose for contradiciton that b_s is a smooth block generated at round $r_s > r_t$. Then from Lemma 5 any block generated at round $r < r_s$ is smooth. But b_t is generated at round $r_t < r_s$ and is thorny, thus we have reached a contradiction.

□

The following corollary emerges immediately from Lemmas 5, 6. This result is illustrated in Figure 11.

Corollary 1. *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.*

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

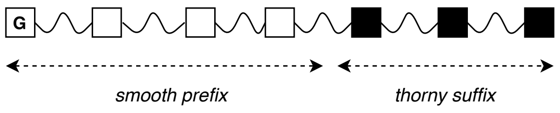

Figure 11: *In the general case the adversarial velvet suffix proof* $\mathcal{P}_{\mathcal{A}} = (\pi_{\mathcal{A}}, \chi_{\mathcal{A}})$ *consists of an initial part of smooth blocks followed by of thorny blocks.*

transaction. In order to construct an interlink containing only the smooth blocks, the miner keeps a copy of the “smooth chain” ($\mathcal{C}_S$) which consists of the smooth blocks existing in the original chain $\mathcal{C}$. The algorithm for extracting the smooth chain out of $\mathcal{C}$ 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 $\mathcal{C}_S$ instead of $\mathcal{C}$.

Algorithm 1: Compute smooth chain

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

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

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

Algorithm 2: Velvet updateInterlink

```
function updateInterlinkVelvet($\mathcal{C}_S$):
    B' $\leftarrow \mathcal{C}_S[-1]$
    interlink $\leftarrow$ B'.interlink
    for $\mu = 0$ to level(B') do
        interlink[$\mu$] $\leftarrow id(B')$
    end
    return interlink
end function
```

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 $\mathcal{C}_S$ instead of $\mathcal{C}$.

Algorithm 3: Velvet Suffix Prover

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

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

4.4 Security of Suffix Proofs

Our security proof is based on the security proof of NIPoPoWs Suffix Security Proof under a soft or hard fork. Keep in mind that we operate under the condition of (1/3)-bounded adversary.

The notion of Chain Quality is decisive for the operation of NIPoPoWs under a velvet fork. Before stepping into the security proof we remind its formal definition2 of this notion and the basic Theorem where its basic metric is computed.

Theorem 3. ***Chain Quality Parameter*** *In a typical execution the chain quality property holds with parameter* $\mu_{cq} > 1 - (1 + \frac{\delta}{2}) \cdot \frac{t}{n-t} - \frac{\delta}{2}$.

Proof. See [4].

Our interest lies in keeping $\mu_{cq} \geq \dfrac{1}{2}$. From Theorem 3 we have:

$$1 - (1 + \frac{\delta}{2}) \cdot \frac{t}{n-t} - \frac{\delta}{2} \geq \frac{1}{2} \Rightarrow (1+\delta)t \leq (1-\delta)(n-t)$$

Thus we have:

$$t \leq (1-\delta)\frac{n}{3} \tag{8}$$

Theorem 4. ***Suffix Proofs Security under velvet fork*** *Assuming honest majority under velvet fork conditions (10) such that* $t \leq (1-\delta)\dfrac{n_v}{3}$*, the non-interactive proofs-of-proof-of-work construction for computable k-stable monotonic suffix-sensitive predicates under velvet fork conditions is secure with overwhelming probability in k.*

Proof. By contradiction. We follow the proof construction of Theorem 2 and extend it. Let Q be a k-stable monotonic suffix-sensitive chain predicate. Assume NIPoPoWs under velvet fork on Q is insecure. Then, during an execution at some round r_3, $Q(C)$ is defined and the verifier V disagrees with some honest participant. Assume the execution is typical. V communicates with adversary A and honest prover B. The verifier receives proofs π_A, π_B. Because B is honest, π_B is a proof constructed based on underlying blockchain C_B (with $\pi_B \subseteq C_B$), which B has adopted during round r_3 at which π_B was generated. Consider $\widetilde{C}_A$ the set of blocks defined as $\widetilde{C}_A = \pi_A \cup \{(\forall b_A \in$

$\pi_A, \exists h, r : b_A \in \mathcal{C}_h^r) : C_h^r\{: b_A\}\}$, where $\mathcal{C}_h^r$ a chain that an honest player p has adopted at some round r. Note that $\widetilde{\mathcal{C}}_A$ forms a chain considering the interlink pointers.

The verifier outputs $\neg Q(C_B)$. Thus it is necessary that $\pi_A \geq \pi_B$. We show that $\pi_A \geq \pi_B$ is a negligible event. Let $b = LCA(\pi_A, \pi_B)$. Let the levels of comparison decided by the verifier be μ_A and μ_B respectively. Let μ'_B be the adequate level of proof π_B with respect to block b. Call $\alpha_A = \pi_A \uparrow^{\mu_A} \{b :\}$, $\alpha'_B = \pi_B \uparrow^{\mu'_B} \{b :\}$.

Our proof construction is based on the following scheme: we show that the competing suffix proofs can be conceived as consisting of three distinct parts. Each part denotes a specific round set and is named after the number of blocks existing in π_A for that round set. Part k_1 stands for the first part of each proof between blocks $b = LCA(\pi_A, \pi_B)$ and $b_2 = LCA(\widetilde{\mathcal{C}}_A, C_B)$. Part k_2 stands for the second part of the proofs considering the rounds from block b_2 and until the Common Prefix is established at $\mathcal{C}_B$ for that fork point. The third and last part, k_3 stands for the rest of the blocks in each proof.
The above are illustrated, among other, in Parts I, II of Figure 12.

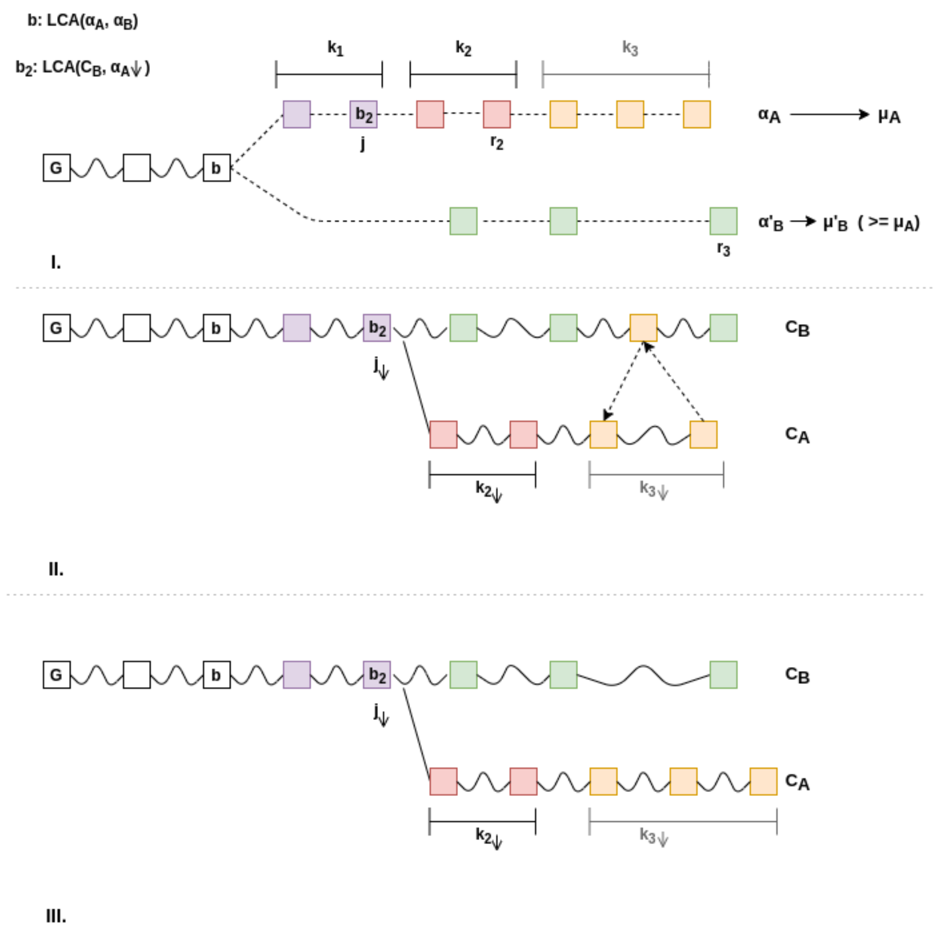

Figure 12: *Wavy lines imply one or more blocks. Dashed lines and arrows imply interlink pointers to superblocks.* ***I****: the three round sets in two competing proofs at different levels,* ***II****: the corresponding 0-level blocks implied by the two proofs,* ***III****: blocks participating in chain $\mathcal{C}_B$ and block set $\widetilde{\mathcal{C}}_A$ as conceived by the verifier's perspective.*

From Corollary 1 we have that the adversarial proof consists of a smooth interlink subchain followed by a thorny interlink subchain. We will refer to the smooth part of $\alpha_{\mathcal{A}}$ as $\alpha_{\mathcal{A}}^{\mathcal{S}}$ and to the thorny part as $\alpha_{\mathcal{A}}^{\mathcal{T}}$.

We will now show three successive claims under velvet fork conditions: First, $\alpha_{\mathcal{A}}^{\mathcal{S}}$ and $\alpha'_B \downarrow$ are mostly disjoint. Second, a_A contains mostly adversarially generated blocks. And third, the adversary is able to produce this a_A with negligible probability.
Let $\alpha_A = k_1 + k_2 + k_3$ and let k_1, k_2, k_3 be as defined in the following Claims.
Let round r_1 be the round when block b is generated and round r_2 when block $b_2 = LCA(\alpha_A, \alpha'_B \downarrow)$ is generated.

Claim 1: $\alpha_{\mathcal{A}}^{\mathcal{S}}$ and $\alpha'_B \downarrow$ are mostly disjoint. Following the proof of Theorem 2 we conclude that $|\alpha_{\mathcal{A}}^{\mathcal{S}} \cap \alpha'_B \downarrow [1:]| \leq k_1 = 2^{\mu'_B - \mu_A}$. In order to see this under the velvet fork conditions, first consider that the adversary behaves honestly for blocks in her proof generated between b and b_2. This means that if b_2 was generated at round r_{b_2} and $\alpha_{\mathcal{A}}^{\mathcal{S}}[-1]$ in round r, then $r \geq r_{b_2}$. In this case Claim 1 of Theorem 2 applies directly. In the opposite case, the adversary includes a thorny block $b_t = \alpha_{\mathcal{A}}^{\mathcal{T}}[0]$ after b and before b_2, thus the inequality still holds and because of Lemma 6 no more honestly generated blocks can be included in α_A after b_t and we can immediately proceed to Claim 3 of this proof.

We conclude that there are at least $|\alpha_A| - k_1$ blocks after block b in α_A which are not honestly generated blocks existing in $\alpha'_B \downarrow$. In other words, there are $|\alpha_A| - k_1$ blocks after block b in α_A, which are either adversarially generated existing in $\alpha_B \downarrow$ either don't belong in $\alpha_B \downarrow$.

Claim 2. At least k_3 superblocks of α_A are adversarially generated. Just as the proof of Theorem 2 and using a similar notation, because of the Common Prefix property on parameter $k_{2\downarrow}$, $\alpha_A[k_1 + k_2 :]$ could contain no honestly generated blocks. In order to see this for the velvet fork conditions let's again consider the case that the adversary behaves honestly for the first $(k_1 + k_2)$ blocks of her proof in which case Claim 2 of Theorem 2 is immediately applied. In the opposite case, the adversary includes in her proof a thorny block at some earlier point. Again, because of Lemma 6 the inequality still holds and no more honestly generated blocks can be included in α_A, so we can proceed to Claim 3 of this proof.

Claim 3. The adversary may submit a suffix proof such that $\alpha_A \geq \alpha_B$ with negligible probability. As argued earlier the last k_3 blocks included in α_A are all adversarially generated. In the worst case all k_3 blocks are sewed from C_B. This is the worst case scenario since each adversarially generated block in C_B may have dropped one honest block out of the chain because of selfish mining. Considering this scenario, because of the strengthened Honest Majority Assumption for (1/3)-bounded adversary, Theorem 3 for Chain Quality guarantees that the majority of the blocks in C_B was computed by honest parties, thus the honestly generated blocks in C_B for the same round set sum to more amount of hashing power.
From all the above Claims we have that:
In the first round set, because of the common underlying chain:

$$2^{\mu_A} |\alpha_A^{k_1}| \leq 2^{\mu'_B} |\alpha'^{k_1}_B| \tag{9}$$

Because of the adoption by an honest party of chain C_B at a later round r_3, we have for the second round set:

$$2^{\mu_A} |\alpha_A^{k_2}| \leq 2^{\mu'_B} |\alpha'^{k_2}_B| \tag{10}$$

In the third round set, because of good Chain Quality under the strengthened Honest Majority Assumption and Theorem 3 we have:

$$2^{\mu_A} |\alpha_A^{k_3}| < 2^{\mu'_B} |\alpha'^{k_3}_B| \tag{11}$$

Consequently we have:

$$2^{\mu_A} |\alpha_A| < 2^{\mu'_B} |\alpha'_B| \tag{12}$$

4.5 Infix Proofs

The security of the original NIPoPoWs protocol suffers under velvet fork conditions for the case of infix proofs as well. Again, since blocks containing incorrect interlink pointers are accepted in the chain, the adversary may create an infix proof for a transaction included in a block mined on a different chain. This attack is presented in detail in the following.

An infix proof attack when applying the original protocol under a velvet fork should be obvious after our previous discussion. So consider the updated protocol for secure suffix proofs as described in the previous section. A problem here is that in the updated protocol some blocks are excluded from the interlink, while we should still be able to provide proofs for transactions included in any block of the chain.

For this reason, let us initially consider an additional protocol patch suggesting to include a second interlink data structure in each block, which will be updated without any block exclusion, just as described in the original protocol and will be used for constructing infix proofs only. In order to be secure we could think of allowing using pointers of the second interlink only for the *followDown* part of the algorithm. But still, the adversary may use an invalid pointer of a block visited during the *followDown* procedure and jump to a block of another chain providing a transaction inclusion proof concerning that block. This attack is illustrated in Figure 13.

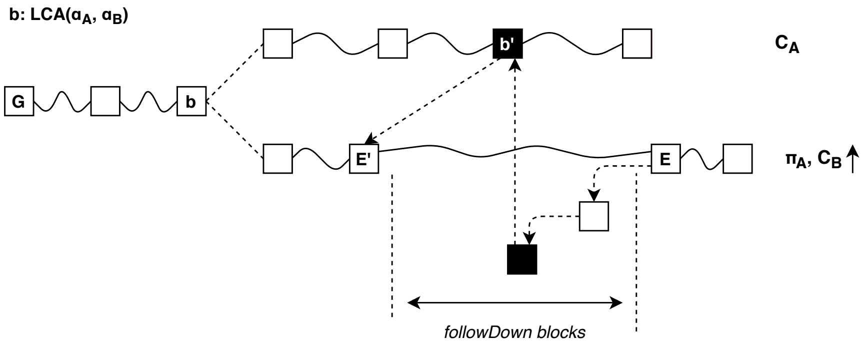

Figure 13: *Adversarial fork chain C_A and an adversarial infix proof based on the chain adopted by an honest player. Wavy lines imply one or more blocks. Blocks generated by the adversary are colored black. Dashed arrows represent interlink pointers included in the proof as part of the followDown procedure. The adversary provides infix proof for a transaction in block b'.*

Thus giving the ability to utilize invalid pointers even in a narrow block window can break the security of our protocol.

Protocol patch for NIPoPoWs infix proofs under velvet fork

However, since we have proved the security of suffix proofs we can include some more information in the blocks participating in these proofs in order to provide secure infix proofs as well.

Specifically, we suggest full nodes to maintain an authenticated data structure, let's say a Merkle Tree, for the blocks consisting the longest valid chain at each time point, and each block to additionally contain the Merkle Tree Root of the blocks' Merkle Tree. In this way an infix proof will consist of a suffix proof in order to obtain the longest valid chain and a block inclusion proof for the block of interest.

For example, in order to prove that a specific transaction tx_1 took place in a block b', the Prover provides:

- a suffix proof π
- a block inclusion proof for b', using the blocks' MTR existing in the tip of the suffix proof chain $\pi[-1]$

- a transaction inclusion proof for tx_1 using the transactions' MTR of block b'

The problem with this infix proof construction is that in order to confirm a transaction in a block of interest b the prover has to provide an inclusion proof for that block of interest using the Merkle Tree Root of a more recent honestly generated block included in the infix proof. Obviously, there is a case that there may be a number of consecutive adversarially generated blocks in the chain consisting the prefix of the chain. Thus, no inclusion proof can be provided for any of these blocks until an honestly generated block is added in the chain and, consequently in the infix proof.

We need to estimate an upper bound for the number of rounds a client has to wait until the block of interest is buried under sufficient number of blocks so that he can obtain the infix proof needed with high probability.

Consider n_b is the upper bound on the consecutive rounds when the adversary could possibly append blocks to the chain, without any honestly generated blocks in between. The adversary mines either by following the selfish mining strategy or not. The state of the chain we examine looks like in Figure 14.

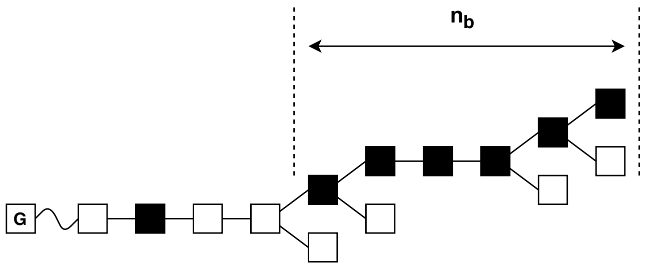

Figure 14: *Wavy lines imply one or more blocks. Blocks generated by the adversary are colored black. n_b implies the number of consecutive adversarially generated blocks in the chain.*

Let p_H the probability that a block is generated by an honest party during a round. Then considering that collisions in RO model is a negligible event we have $p_H = (n-t)qp$. Let p_A be the probability that a block is generated by the adversary during a round, then $p_A = tqp$. Consider N_{bA} the random variable for the number of consecutive blocks generated by the adversary in a total of r rounds. Also let N_{bH} the random variable for the number of blocks generated by honest parties in total of r rounds. Then we have that N_{bA}, N_{bH} follow the Binomial Distribution with probability p_A, p_H respectively.

Because we require consecutive successful rounds for the adversary, we have:

$$Pr[N_{bA} = n_b] = (r - n_b - 1)p_A^{n_b}(1 - p_A)^{r-n_b} \tag{13}$$

while for the honest players we have

$$Pr[N_{bH} \leq n_b] = \sum_{i=0}^{n_b} \binom{n}{k} p_H^i (1 - p_H)^{r-i} \tag{14}$$

If N_b is the random variable denoting the number of consecutive rounds that adversarially generated blocks are appended to the chain, we have:

$$Pr[N_b = n_b] = Pr[N_{bA} = n_b] \cdot Pr[N_{bH} \leq n_b] \tag{15}$$

since the two events are independent.

The velvet infix prover

The construction of an infix proof is described in Algorithm 4. In order to keep the algorithm generic enough for any infix-sensitive predicate, we provide the steps needed until the verification of the block of interest and consider the specific predicate answer as trivial to calculate given the block of interest. The infix prover accepts as input the full chain C and a block of interest b' and returns a proof consisting of the Merkle Tree proof of inclusion for b' and a suffix proof.

Algorithm 4: Velvet Infix Prover

```
1 function ProveInfixVelvet(C, b'):
2     $(\pi, \chi) \leftarrow ProveVelvet_{m,k}(\mathcal{C}_S)$
3     $tip = \pi[-1]$
4     $\pi_{b'} \leftarrow \text{ConstrMTInclProof}(tip, b'.id)$
5     return $(\pi'_b, (\pi, \chi))$
6 end function
```

The velvet infix verifier

The infix proof verification algorithm is described in Algorithm 5. Supposing that the verifier has already concluded to the longest valid chain C after accepting and comparing constesting suffix proofs, the infix verification algorithm only has to confirm the Merkle-Tree inclusion proof $\pi_{b'}$ for the block of interest b'.

Algorithm 5: Velvet Infix Verifier

```
1 function VerifyInfixVelvet($\pi'_b, (\pi, \chi)$):
2     $tip = \pi[-1]$
3     return $\text{VerMTInclProof}(tip.MTR_{blocks}, \pi_{b'}, b'.id)$
4 end function
```

References

[1] Kiayias A., Miller A., and Zindros D. Non-interactive proofs of proof-of-work. *IACR Cryptology ePrint Archive*, 2017.

[2] Zamyatin A., Stifter N., Judmayer A., Schindler P., Weippl E., and Knottenbelt W.J. A wild velvet fork appears! inclusive blockchain protocol changes in practice. *Zohar A. et al. (eds) Financial Cryptography and Data Security. FC 2018*, 2019.

[3] Ittay Eyal and Emin Gun Sirer. Majority is not enough: Bitcoin mining is vulnerable, 2013.

[4] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310, 2015.

[5] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.