National and Kapodistrian University of Athens
School of Science

Department of Informatics and Telecommunications

Postgraduate Studies
Computer Systems: Software and Hardware

Master Thesis

A Study on Superlight Blockchain Clients under Velvet Fork

Andrianna Polydouri

Supervisors: **Aggelos Kiayias**, Associate Professor NKUA
Dionysis Zindros, PhD NKUA

Athens,
15th June 2020

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Σχολή Θετικών Επιστημών
Τμήμα Πληροφορικής κ' Τηλεπικοινωνιών

Μεταπτυχιακές Σπουδές
Υπολογιστικά Συστήματα: Λογισμικό και Υλικό

Διπλωματική Εργασία

Υπερελαφρείς Πελάτες Αλυσίδας υπό "Βελούδινη" Αναβάθμιση Πρωτοκόλλου

Ανδριάννα Πολυδούρη

Επιβλέποντες: Άγγελος Κιαγιάς, Αναπληρωτής Καθηγητής ΕΚΠΑ
Διονύσης Ζήνδρος, Διδάκτωρ ΕΚΠΑ

Αθήνα,
15η Ιουνίου 2020

Master Thesis

A Study on Superlight Blockchain Clients under Velvet Fork

Andrianna Polydouri
Reg.Nr.: M1598

Supervisors:

Aggelos Kiayias, Associate Professor NKUA
Dionysis Zindros, PhD NKUA

Thesis Committee:

Aggelos Kiayias, Associate Professor NKUA
Mema Roussopoulos, Associate Professor NKUA
Yannis Smaragdakis, Professor NKUA

Διπλωματική Εργασία

Υπερελαφρείς Πελάτες Αλυσίδας υπό "Βελούδινη" Αναβάθμιση Πρωτοκόλλου

Ανδριάννα Πολυδούρη
Αρ. Μητρώου: M1598

Επιβλέποντες:
Άγγελος Κιαγιάς, Αναπληρωτής Καθηγητής ΕΚΠΑ
Διονύσης Ζήνδρος, Διδάκτωρ ΕΚΠΑ

Εξεταστική Επιτροπή:
Άγγελος Κιαγιάς, Αναπληρωτής Καθηγητής ΕΚΠΑ
Μέμα Ρουσσοπούλου, Αναπληρώτρια Καθηγήτρια ΕΚΠΑ
Γιάννης Σμαραγδάκης, Καθηγητής ΕΚΠΑ

Abstract

Superlight blockchain clients learn facts about the blockchain state while requiring only polylogarithmic communication in the total number of blocks. For proof-of-work blockchains two known constructions exist: Superblock and FlyClient.

Unfortunately, none of them can be deployed to existing blockchains as they require changes at the consensus layer and at least a soft fork to implement.

In this work, we investigate how a blockchain can be upgraded to support superblock clients without a soft fork. We show that it is possible to implement the needed changes without modifying the consensus protocol and by requiring only a minority of miners to upgrade, a process termed a "velvet fork" in the literature. While previous work conjectured that Superblock and FlyClient clients can be safely deployed using velvet forks as-is, we show that previous constructions are insecure. We describe a novel class of attacks, called "chain-sewing", which arise in the velvet fork setting: an adversary can cut-and-paste portions of various chains from independent forks, sewing them together to fool a superlight client into accepting a false claim. We show how previous velvet fork constructions can be attacked via chain-sewing. Next we put forth the first provably secure velvet superblock client construction which we show secure against adversaries that are bounded by 1/4 of the upgraded honest miner population.

Περίληψη

Οι *υπερ-ελαφρείς* πελάτες αλυσίδων λαμβάνουν ενημερώσεις για την τρέχουσα κατάσταση της αλυσίδας απαιτώντας ανταλλαγή πληροφοριών λογαριθμικού μήκους σε σχέση με το συνολικό μέγεθος της αλυσίδας. Για τις αλυσίδες "απόδειξης-εργασίας" υπάρχουν δύο τέτοιες γνωστές κατασκευές που αναφέρονται στη βιβλιογραφία ως "Superblock" και "FlyClient". Δυστυχώς καμία από τις δύο αυτές κατασκευές δεν μπορεί να ενσωματωθεί άμεσα σε υπάρχουσες αλυσίδες, διότι απαιτούν αλλαγές στο πρωτόκολλο συναίνεσης και, για αυτό, την αναβάθμιση της συντριπτικής πλειόνοτητας των συμμετεχόντων στο κατανεμημένο δίκτυο.

Σε αυτήν την εργασία εξετάζουμε την δυνατότητα αναβάθμισης του πρωτοκόλλου συναίνεσης της αλυσίδας για τη λειτουργία υπερλαφρών πελατών από ένα μικρό μόνο μέρος των συμμετεχόντων παικτών, κάτι που στη βιβλιογραφία αναφέρεται ως "βελούδινο σχίσμα" (velvet fork). Προηγούμενες εργασίες υπέθεταν ότι υπερλαφείς πελάτες Superblock και FlyClient μπορούν να υποστηριχθούν με ασφάλεια μέσω velvet fork χωρίς περαιτέρω αλλαγές στις κατασκευές τους και υπό τις ίδιες προϋποθέσεις ασφαλείας. Δείχνουμε ότι αυτή η υπόθεση είναι εσφαλμένη. Περιγράφουμε ένα νέο είδος επίθεσης, υπό το όνομα "ράψιμο της αλυσίδας" (chainsewing), που εμφανίζεται σε περιβάλλον ενός velvet fork: ένας κακόβουλος χρήστης μπορεί να "κόψει" τμήματα διαφόρων ανεξάρτητων αλυσιδών και να τα "ράψει" μαζί ώστε να ξεγελάσει έναν υπερλαφρύ πελάτη να δεχτεί έναν λανθασμένο ισχυρισμό για την έγκυρη αλυσίδα. Δείχνουμε πώς οι προαναφερθείσες κατασκευές είναι ευάλωτες σε τέτοιου είδους επιθέσεις. Επιπλέον παρουσιάζουμε την πρώτη αποδεδειγμένα ασφαλή "βελούδινη" κατασκευή βασισμένη στα superblocks, για την οποία δίνουμε απόδειξη ασφάλειας για κακόβουλο πληθυσμό που φράσσεται άνω από το 1/4 του αναβαθμισμένου τίμιου πληθυσμού.

Contents

Chapter 1

Introduction

1.1 Motivation

Blockchain systems such as Bitcoin [25] and Ethereum [2, 29] have a predetermined expected rate of block production and maintain chains of blocks that are growing linearly with time. A node synchronizing with the rest of the blockchain network for the first time therefore has to download and validate the whole chain, if it does not wish to rely on a trusted third party. While a lightweight node (SPV) can avoid downloading and validating transactions beyond their interest, it must still download the block headers that contain the proof-of-work [6] of each block in order to determine which chain contains the most proof-of-work. The block header data, while smaller by a significant constant factor, still grow linearly with time. An Ethereum node synchronizing for the first time must download more than 300 MB of block header data for the purpose of proof-of-work verification, even if it elects not to download any transactions. This has become a central problem to the usability of blockchain systems, especially for vendors who are using mobile phones to accept payments or sit behind limited internet bandwidth. They are forced to make a difficult choice between decentralization and the ability to start accepting payments in a timely manner.

1.2 Current approaches

Towards the goal of alleviating the burden of this download for SPV clients, a number of *superlight* clients has emerged. These protocols give rise to Non-Interactive Proofs of Proof-of-Work (NIPoPoW) [18], which are short strings that "compress" the proof-of-work information of the underlying chain. The necessary security property of such proofs is that a minority adversary can only convince a NIPoPoW client that a certain transaction is confirmed, only if they can convince an SPV client, too.

There are two general directions for superlight client implementations: In the *superblock* [18, 12] approach, the client relies on *superblocks*, blocks that have achieved much better proof-of-work than required for block validity. In the *FlyClient* [1] approach, blocks are sampled and committed at random as in a Σ-protocol (e.g. Schnorr's discrete-log protocol [27]) and then using the Fiat–Shamir heuristic [8] a non-interactive proof is calculated. The number of block headers that need to be sent then grows only logarithmically with time. The NIPoPoW client, which is the proof *verifier* in this context, still relies on a connection to full nodes, who, acting as *provers*, perform the sampling of blocks from the full blockchain. No trust assumptions are made for these provers, as the verifier can check the veracity of their claims. As long as the verifier is connected to at least one honest prover (an assumption also made in the SPV protocol [10, 30]), they are able to arrive at the correct claim.

In both approaches, it is essential for the verifier to check that the blocks sampled one way or another have been generated in the same order as they have been presented by the prover. As such, each block in the proof must contain a pointer to the previous block in the proof. As blocks in these proofs are far apart in the underlying blockchain, the legacy *previous block pointer*, which

typically appears within block headers, does not suffice. Both approaches require modifications to the consensus layer of the underlying blockchain to work. In the case of superblock NIPoPoWs, the block header must be modified to include, in addition to a pointer to the previous block, pointers to a small amount of recent high-proof-of-work blocks. In the case of FlyClient, each block must additionally contain pointers to all previous blocks in the chain. Both of these modifications can be made efficiently by organizing these pointers into Merkle Trees [24] or Merkle Mountain Ranges [21, 28] whose root is stored in the block header. The inclusion of extra pointers within blocks is termed *interlinking the chain* [17].

The modified block format, which includes the extra pointers, must be respected and validated by all full nodes and thus requires either a hard fork or at least a soft fork. However, even soft forks require the approval of a supermajority of miners, and new features that are considered non-essential by the community have taken years to receive approval [22]. Towards the goal of implementing superlight clients sooner, we study the question of whether it is possible to deploy superlight clients without a soft fork. We propose a series of modifications to blocks that are *helpful but untrusted*. These modifications mandate that some extra data is included in each block. The extra data is placed inside the block by upgraded miners only, while the rest of the network does not include the additional data into the blocks and does not verify its inclusion, treating them merely as comments. To maintain backwards compatibility, contrary to a soft fork, upgraded miners must accept blocks that do not contain this extra data that have been produced by unupgraded miners, or even blocks that contain invalid or malicious such extra data produced by a mining adversary. This acceptance is necessary in order to avoid causing a chain split with the unupgraded part of the network. Such a modification to the consensus layer is termed a *velvet fork* [32].

In this context the contributions resulting from this work come as follows:

- We revise the security proof for superblock suffix proof protocol and compute a concrete value for the security parameter m
- We illustrate that, contrary to claims of previous works, superlight clients designed to work in a soft fork cannot be readily plugged into a velvet fork and expected to work. We present a novel attack termed the *chain-sewing* attack which thwarts the defenses of previous proposals and allows even a minority adversary to cause catastrophic failures.
- We propose the first *backwards-compatible superlight client*. We put forth an interlinking mechanism implementable through a velvet fork. We then construct a superblock NIPoPoW protocol on top of the velvet forked chain and show it allows to build superlight clients for various statements regarding the blockchain state via both "suffix" and "infix" proofs.
- We prove our construction secure in the synchronous static difficulty model against adversaries bounded to 1/4 of the mining power of the honest upgraded nodes. As such, our protocol works even if a constant minority of miners adopts it.

1.3 Related work.

Proofs of Proof-of-Work have been proposed in the context of superlight clients [6, 18, 1], cross-chain communication [19, 13, 31], as well as local data consumption by smart contracts [14]. Superblock NIPoPoWs have been deployed in production using hard forks [5] and have been conjectured to work in velvet fork conditions [18] (we show here that these conjectures are ill-informed in the light of our chain-sewing attack). Velvet forks [32] have been studied for a variety of other applications and have been deployed in practice, e.g., see [11]. In this work, we focus on consensus state compression. Such compression has been explored in the hard-fork setting using zk-SNARKS [23] as well as in the Proof-of-Stake setting [16]. Complementary to consensus state compression (i.e., the compression of block headers and their ancestry) is compression of application state (namely the State Trie, the UTXO, or transaction history). There is a series of works complementary and composable with ours that discusses the compression of application state [3, 20].

Chapter 2

Background

2.1 Cryptographic Primitives

In this section we define some fundamental cryptographic primitives which are parts of the blockchain technology. The definitions of correctness and security of these primitives are in most cases defined using the notion of *negligible functions*. We therefore first provide the definition of a negligible function in Definition .

Definition 1 (Negligible function)**.** A function f is negligible if for all $c \in \mathbb{R}$ there exists $n_0 \in \mathbb{N}$ such that $f(n) \leq \frac{1}{n^c}$ for all $n \geq n_0$.

Note that in the following we may often refer to *polynomial-probalistic-time (PPT)* adversary simply as "adversary".

2.1.1 Digital Signatures

Digital signature schemes allow a signer S who has established a public key pk to *sign* a message m in such a way that any other party who knows pk (and knows that this public key was established by S) can *verify* that m originated from S and has not been modified in any way [15]. The syntax of a digital signature scheme is formally defined in Definition 1.

Definition 1 (Signature scheme syntax [15])**.** We call a signature scheme is a tuple of probabilistic polynomial-time algorithms $\Pi = (Gen, Sign, Ver)$ satisfying the following:

1. The key-generation algorithm Gen takes as input a security parameter 1^κ and outputs a pair of keys (pk, sk). These are called the public key and the private key, respectively.

$$(sk, pk) \leftarrow \mathsf{Gen}(1^\kappa)$$

2. The signing algorithm Sign takes as input a private key sk and a message m from some underlying message space. It outputs a signature σ.

$$\sigma \leftarrow \mathsf{Sign}_{sk}(m)$$

3. The deterministic verification algorithm Ver takes as input a publock key pk, a message m and a signature σ. It outputs a bit b, with $b = 1$ meaning VALID and $b = 0$ meaning INVALID.

$$\mathsf{Ver}_{pk}(m, \sigma) = 0 \text{ or } 1$$

We require that for every κ, every (pk, sk) output by $\mathsf{Gen}(1^\kappa)$ and every message m in the appropriate underlying plaintext space, it holds that

$$\mathsf{Ver}_{pk}(m, \mathsf{Sign}_{sk}(m)) = 1$$

Correctness of signature schemes. We say that σ is a *valid signature* on a message m with respect to public key pk if $\mathsf{Ver}_{pk}(m, \sigma) = 1$. We formally define the correctness of a signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ using the $\mathsf{Sig\text{-}correct}_{\Pi}(\kappa, m)$ experiment for security paramerter κ and every message m given in Algorithm 1.

Algorithm 1 The Sig-correct signature experiment

```
1: function SIG-CORRECT_Π(κ, m)
2:     (pk, sk) ← Gen(1^κ)
3:     (σ) ← Sign_sk(m)
4:     if Ver_pk(m, σ) = 1 then
5:         return 1
6:     end if
7:     return 0
8: end function
```

Definition 2 (Correctness of signature scheme)**.** A signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is correct if for every m from the underlying message space there exists a negligible function $\mathsf{negl}(\cdot)$ such that:

$$Pr[\mathsf{Sig\text{-}correct}_{\Pi}(\kappa, m) = 0] \leq \mathsf{negl}(\kappa)$$

Security of signature schemes. Given a public key pk generated by a signer S, we say that an adversary outputs a *forgery* if she outputs a message m along with a valid signature σ on m and m was not previously signed by S, since in this case the adversary could simply copy the originally signed message. *Security* of a digital signature scheme means that an adversary cannot output forgery even if she is allowed to obtain Signatures on many other messages of her choice.

We formally define the security of a signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ using the $\mathsf{Sig\text{-}forge}_{\mathcal{A},\Pi}(\kappa)$ experiment for adversary $\mathcal{A}$ and security paramerter κ given in Algorithm 2. In this experiment we consider that the adversary is given pk and oracle access to $\mathsf{Sign}_{sk}(\cdot)$. This oracle returns a valid signature $\mathsf{Sign}_{sk}(m)$ for any message m of the adversary's choice. Let Q denote the set of messages whose signatures were requested by $\mathcal{A}$.

Algorithm 2 The Sig-forge signature experiment

```
1: function SIG-FORGE^cma_{A,Π}(κ)
2:     (pk, sk) ← Gen(1^κ)
3:     (m, σ) ← A(pk)
4:     if Ver_pk(m, σ) = 1 ∧ m ∉ Q then
5:         return 1
6:     end if
7:     return 0
8: end function
```

Definition 3 (Security of signature scheme [15])**.** A signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is existentially unforgeable under an adaptive chosen-message attack if for all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that:

$$Pr[\mathsf{Sig\text{-}forge}^{cma}_{\mathcal{A},\Pi}(\kappa) = 1] \leq \mathsf{negl}(\kappa)$$

2.1.2 Collision-Resistant Hash Functions

In general, hash functions are just functions that take arbitrary-length strings and *compress* then into shorte strings. The classic use of hash functions is in data structures as a way to achieve $\mathcal{O}(1)$ lookup time for retrieving an element. Specifically, if the size of the range of hte hash function H is N, then a table is first allocated with N entries. Then, the element x is stored in cell $H(x)$ in

the table. In order to retrieve x, it suffices to compute $H(x)$ and probe that table entry. Observe that since the output range of H equals the size of the table, the output length must be rather short or else the table will be too large. A "good" hash function for this purpose is one that yields as few *collisions* as possible, where a collision is a pair of distinct data items x and x' for which $H(x) = H(x')$. Notice that when a collision occurs, two elements end up being stored in the same cell. Therefore, many collisions may result in a higher than desired retrieval complexity. In short, what is desired is that the hash function spreads the elements well in the table, thereby minimizing the number of collisions[15].

Collision-resistant or *cryptographic* hash functions are similar in principle to those used in data structures. In particular, they are also functions that compress their input by transforming arbitrary-length input strings into output strings of a fixed shorter length. Furthermore, collisions are a problem. However, the *desire* in data structures to have few collisions is now a *mandatory requirement* in cryptography. That is, a collision-resistant hash function must have the property that no polynomial-time adversary can find a collision in it. Stated differently, no polynomial-time adversary should be able to find a distinct pair of values x and x' such that $H(x) = H(x')$. We stress that in data structures some collisions may be tolerated, whereas in cryptography no collisions whatsoever are allowed. Furthermore,the adversary in cryptography specically searches for a collision, whereas in data structures,the "data items" do not attempt to collide intentionally. This means that the requirements on hash functions in cryptography are much more stringent than the analogous requirements in data structures.It also means that cryptographic hash functions are harder to construct[15].

Defining collision-resistance. A *collision* in a function H is a pair of distinct inputs x, x' such that $H(x) = H(x')$. H is *collison-resistant* if it is infeasible for any PPT algorithm to find a collision in H. Typically, we are interested in functions that have an infinite domain (i.e. they accept all strings of all input lengths) and a finite range. Note that in a such a case collisions will necessarily exist, due to the pigeon-hole principle. The requirement is therefore only that such collisions should be hard to find.

We need a family of hash functions in order to define what a collision-resistant hash function is. That is because the adversary can have hardwired a collision pair in his code and output it every time we ask him. Note that this is not a problem for the security of the one-way function, since in that case we ask the adversary for the inversion of a random element in the range of the function.

For the definition of the collision resistance property we use the experiment Hash-collison described in Algorithm 3.

Algorithm 3 The Hash-collision experiment

```
1: function HASH-COLLISION_{F,A}(κ)
2:     (i) ← Gen(1^κ)
3:     (x, x') ← H_i
4:     if x ≠ x' ∧ H_i(x) ≠ H_i(x') then
5:         return 1
6:     end if
7:     return 0
8: end function
```

Definition 4 (Collision-Resistant Hash Function)**.** A family of hash functions $\mathcal{F} = \{\mathcal{H}_i : D_i \to R_i\}_{i \in \mathcal{I}}$ is collision resistant if it satisfies the following:

- **Easy to sample:** There exists a PPT algorithm Gen, such that for all $\kappa \in \mathbb{N}$, $\mathsf{Gen}(1^\kappa) \in \mathcal{I}$.
- **Easy to compute:** There exeists PPT algorithm that in input $i \in \mathcal{I}$, $x \in D_i$ returns $\mathcal{H}_i(x)$.
- **Compressing:** For all $i \in \mathbb{N}$, $|R_i| < |D_i|$.
- **Collision resistant:** For every PPT adversary $\mathcal{A}$, for all $\kappa \in \mathbb{N}$:

$$Pr[\mathsf{Hash\text{-}collision}_{\mathcal{F},\mathcal{A}}(\kappa) = 1] \leq negl(\kappa)$$

2.1.3 The Random Oracle Model

In many cases there is the need to find a "middle-ground" between a fully-rigorous proof of security on the one hand and no proof whatsoever on the other. This may be achieved by introducing an idealized model in which to prove security of cryptographic schemes. Though the idealization may not be an accurate reflection of reality, we can at least derive some measure of confidence in the soundness of a scheme's design from a proof within the idealized model. Al long as the model is reasonable, such proofs are cenrtainly better than no proofs at all.

Towards this end, the *random oracle model* posits the existence of a public, randomly-chosen function H that can be evaluated *only* by querying the oracle - which can be thought of as a "magic box" - that returns $H(x)$ when given input x. Now, the idealized model of random oracle can be used to design and validate cryptographic schemes via the following two-step approach:

1. First, a scheme is designed and proven secure in the random oracle (RO) model
2. When we want to implement the scheme in the real world, a RO is not available. Instead, the RO H in the scheme is *instantiated* with a cryptographic hash function $\tilde{H}$.

The hope is that the cryptographic hash function $\tilde{H}$ is "sufficiently good" at simulating a random oracle, so that the security proof given in the first step will carry over to the real-world instantiation of the scheme.

Defining the RO model. A good way to think about a RO model is as follows. The "oracle" is simply a box that takes a binary string x as input and returns a binary string y as output. We refer to such interactions with the box as *querying the oracle on* x and call x itself a *query* made to the oracle. The internal workings of the box are unknown and inscrutable. It is guaranteed that the box is *consistent*: that is, if the box ever outputs y for a particular input x, then it always outputs the same output y when given the same input x again. So, we can view the box as implementing a hash function H.

Thus we can now provide a formal definition of RO as in Definition 5.

Definition 5 (Random Oracle)**.** The Random Oracle is an idealized model for cryptographic hash functions which operates as follows:

- given $x \notin \mathit{History}$, choose $y \xleftarrow{r} Y$ and add (x, y) to *History*. Return y.
- given x such that $(x, y) \in \mathit{History}$ for some y, return y.

2.2 Blockchain Basics

2.2.1 The notion of *block*

A *blockchain* is a timely ordered sequence of logical units called *blocks*. In a cryptocurrecny blockchain, like Bitcoin, a block is a Proof-of-Work-verified set of information containing a number of transactions that are hashed and encoded in a Merkle Tree data structure. Each block includes the cryptographic hash of the prior block in the block sequence, linking the two. Thus the linked blocks form a chain. It additionally contains a nonce value which is related to the Proof-of-Work *(PoW)* process. The PoW involves a computation over a cryptographic puzzle. More specifically, it involves scanning for a value, *ctr*, such that when included in the block, the block hashes to a value lower than a certain threshold T. The hash of a block is the block's *id*. The formal definition of a block is given in Definition 2.

Definition 2 (Block)**.** Let $G(\cdot)$, $H(\cdot)$ be cryptographic hash functions and $T \in \mathsf{range}(H)$. 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

$$H(ctr, G(s, x)) < T \tag{2.1}$$

The inverse of 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 PoW blockchain.

2.2.2 The notion of *blockchain*

A blockchain, or simply chain, is a timely ordered sequence of blocks. 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 2.1 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.

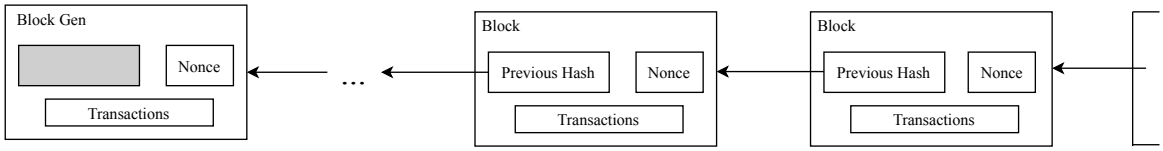

Figure 2.1: A high-level representation of a blockchain.

A cryptocurrency blockchain is essentially a distributed ledger containing the history of the transactions made between the participating parties. The blockchain ledger has the following two fundamental properties: i) it is *immutable* and, ii) it is *append-only*. Sometimes separate blocks can be produced concurrently, creating a temporary fork. In addition to a secure hash-based history, any blockchain has a specified algorithm for scoring different versions of the history so that one with a higher score can be selected over others. Bitcoin uses a proof-of-work system, where the chain with the most cumulative proof-of-work is considered the valid one by the network. Peers supporting the ledger may have different versions of the history from time to time. They keep only the highest-scoring version of the database known to them. Whenever a peer receives a higher-scoring version (usually the old version with a single new block added) they extend or overwrite their own database and retransmit the improvement to their peers. There is never an absolute guarantee that any particular entry will remain in the best version of the history forever. Blockchains are typically built to add the score of new blocks onto old blocks and are given incentives to extend with new blocks rather than overwrite old blocks. Therefore, the probability of an entry becoming superseded decreases exponentially as more blocks are built on top of it, eventually becoming very low.

2.2.3 Transactions

The transactions in an electronic coin are defined as a chain of digital signatures. Each owner transfers coin value to the next owner by digitally signing a hash of the previous transaction and the public key of the next owner and adding these two to the end of the transaction script, which is publicly announced to the network. A payee can verify the signatures to verify the chain of ownership. Of course the payee should somehow verify that the value transferred is not double-spent by the owner without the trust of a third party authority. In the Bitcoin's Unspent-Transaction-Output *(UTXO)* model each unspent coin value is included in the so-called UTXO set. Every peer inspecting the transactions in the network validates that the value transferred in a transaction tx belongs in the current UTXO set before including tx in a block. Each time that a peer updates its blockchain, he updates the UTXO set according to the new transaction history too. By agreeing on a single blockchain history according to the highest score (highest PoW score for Bitcoin) among the all the existing chains in the network, the peers can also agree on a single history as for the sequence of the transactions made so far.

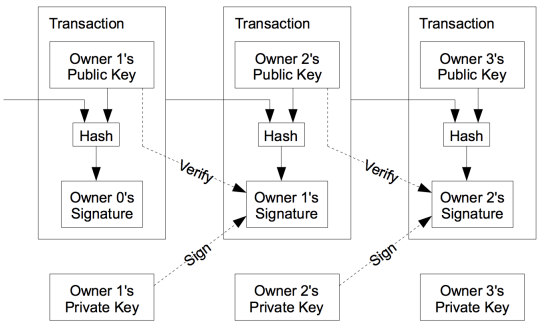

Figure 2.2: Transactions as a chain of digital signatures in Bitcoin [25]

2.2.4 The SPV model

In the Bitcoin blockchain network each peer may have one of the following three roles: *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 clients.

In order to make the client functionality more efficient the Simple Payment Verification was proposed [25]. Based on the SPV scheme, there can be *lightweight clients*, meaning clients that need to store only the block headers of the 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.

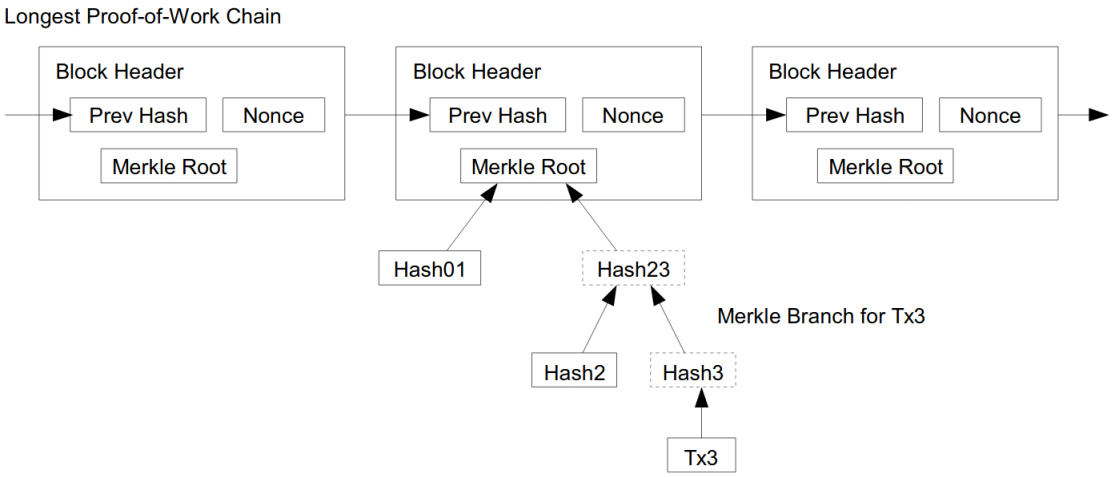

Figure 2.3: High level representation of blockchain data kept by a lightweight client and an inclusion proof for a transaction Tx3.[25]

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 of the chain is constantly increasing in a linear fashion, there is a need to construct more efficient protocols serving the needs of lightweight clients.

2.3 The Backbone Model

2.3.1 The protocol

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 2.1 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 2.1: 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[9].

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.

2.3.2 Basic properties

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

Definition 3 (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 4 (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 5 (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.

2.4 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, resulting to a permanent fork of the chain.

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 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. Considering that the majority of the players have upgraded to the new protocol rules, unupgraded players will accept all blocks but see their own blocks being abandoned, thus they are motivated to upgrade as well.

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[18].

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.

Chapter 3

Superblocks under Soft Fork

3.1 Introduction

The SPV clients described in the Bitcoin paper need to process only the block headers of the chain in order to synchronize. This is much more efficient compared to the synchronization of a full node, however it still requires processing data which grow linearly with the size of the chain.

Superblock NIPoPoWs provide synchronization and up-to-date chain information with only poly-logarithmic data to the size of the chain. Let us describe the underlying primitive and try to provide some intuition about it.

As already explained, each block appended to the chain must contain an appropriate nonce value, so that the hash of the whole block is a number lower than a specific threshold. Assuming difficulty $= 1/T$ then for a block's hash id it must hold that $\frac{\text{id}}{T} < 1$. This is illustrated in the part I of Figure 3.1. Remember that κ is the length of the hash function output.

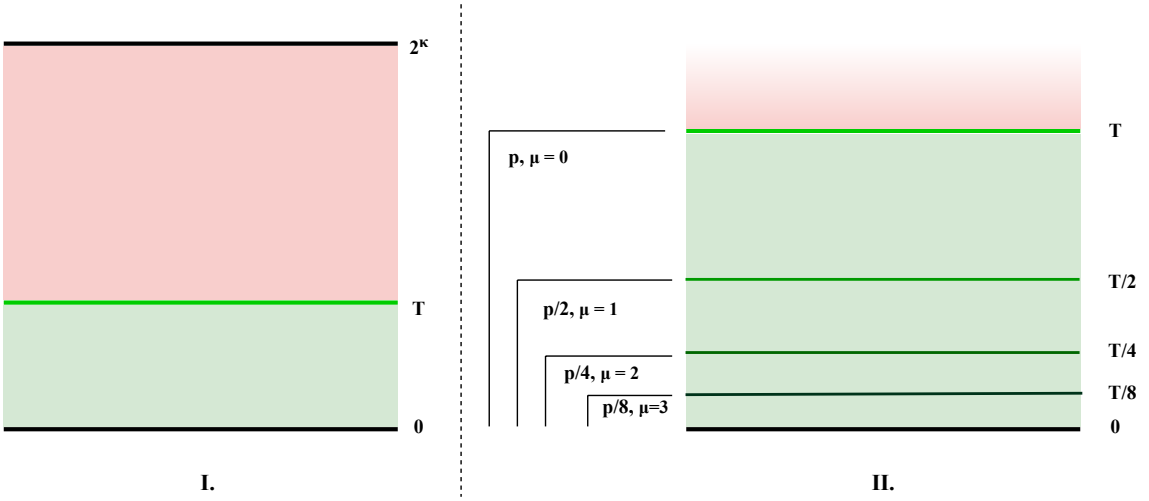

Figure 3.1: Graphical representation of PoW domain. I. valid blocks ids lie in the green section. II. blocks of higher level are generated with lower probability.

Note that because of the Random Oracle model for the hash function, the outputs of the PoW attempts are uniformly distributed in the domain $\{0, 2^\kappa\}$. The values regarding any subdomain follow uniform distribution too. In essence, a block b with $id < T$ is generated with probability $p = \frac{T}{2^\kappa}$, while a block b_1 with $id_1 < \frac{T}{2}$ is generated with probability $p_1 = \frac{T}{2 \cdot 2^\kappa}$ or $p_1 = \frac{p}{2}$. As

you can see it seems that b_1 is a "luckier" block than block b, and we could even say that it is twice as lucky since such a block is generated half of the times that b is, in expectation. This can be generalized to the following form: a block b' with $\mathrm{id}' < \frac{T}{2^\mu}$ is generated with probability $p' = \frac{p}{2^\mu}$. We call μ the level of the block b'. It should now be obvious that blocks of high levels appear rarely in the chain according to their level. The higher the level μ, the more rare the blocks of that level in the chain. Part II of Figure 3.1 illustrates this result.

From all the above we can conclude to the following. All blocks in the chain are level 0. Blocks of level μ are called μ-superblocks, while it holds that μ-superblocks for $\mu > 0$ are also $(\mu - 1)$-superblocks. The level of a block is given as $\mu = \lfloor log(T) - log(id(B)) \rfloor$ and denoted $level(B)$. By convention, for the genesis block we set $id = 0$ and $\mu = \infty$.

The important observation on the superblocks is that they are expected to appear in the chain with a constant frequency according to their level. This is validated for the bitcoin blockchain in [12]. In a blockchain protocol execution it is expected that half of the blocks will be of level 1, 1/4 of the blocks will be of level 2, 1/8 of the blocks of level 3 and, generally, $1/2^\mu$ blocks will be of level μ. In expectation the number of superblock levels appearing in a chain C will be $\Theta(log(\mathsf{C}))$. Figure 3.2 illustrates the blockchain superblocks starting from level 1 and going up to level 4 in the case that superblocks are distributed exactly according to expectation. Each level contains half of the blocks of the level below [18].

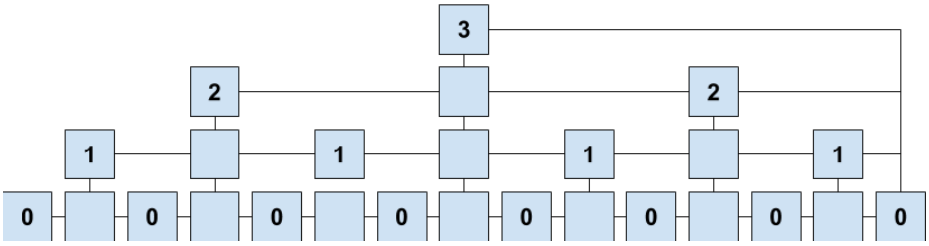

Figure 3.2: Ideal superblock distribution. Higher levels have achieved higher difficulty during mining [18].

The main idea behind superblock lightclients should now seem straightforward. Since one $\mu-$superblock roughly represents 2^μ 0-level blocks, why don't we provide only these very lucky blocks instead of communicating the headers of all blocks in the chain in order to synchronize?

3.2 Interlinking the chain

In order to be able to utilize the superblocks for constructing succinct proofs of PoW it is proposed that block headers additionally include the *interlink* data structure. The interlink of a block b contains pointers to the most recent $\mu-$level ancestor of b for every $\mu \leq log(\mathsf{C})$. The interlink turns the blockchain into a skiplist-like data structure, as illustrated in Figure 3.3.

Updated interlink information has to be included in each block during mining. The algorithm for the honest miner is given in Algorithm 4 as described in [17]. The updateInterlink algorithm accepts a block B', which already contains an interlink strucure, and evaluates the interlink that has to be included as part of the next block. It copies the existing interlink and then modifies its pointers from level 0 to $level(B')$, so that they point to block B'. For every freshly mined block relayed to the network, a node has now additionally to check the validity of the interlink data included.

For the rest of this work, we use the notation defined in the superblock NIPoPows paper [18], which we rewrite here so as to serve as an easier point of reference. Blockchains are sequences but it is more convenient to use set notation for some operations. Specifically $B \in \mathsf{C}$, $\mathsf{C}_1 \subseteq \mathsf{C}_2$ and $\emptyset$ have the obvious meaning. $\mathsf{C}_1 \cup \mathsf{C}_2$ is the chain obtained by sorting the blocks contained in both C_1 and C_2 into a sequence (this may be not always defined). We will freely use set builder notation $\{B \in \mathsf{C} : p(B)\}$. $\mathsf{C}_1 \cap \mathsf{C}_2$ is the chain $\{B : B \in \mathsf{C}_1 \wedge B \in \mathsf{C}_2\}$. The lowest common ancestor is

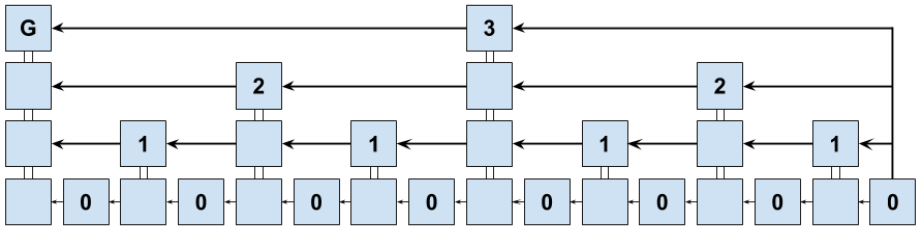

Figure 3.3: The hierarchical blockchain. Each block has a pointer to its nearest $\mu-$level ancestor.

Algorithm 4 updateInterlink [17]

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

$LCA(\mathsf{C}_1, \mathsf{C}_2) = (\mathsf{C}_1 \cap \mathsf{C}_2)[-1]$. if $\mathsf{C}_1[0] = \mathsf{C}_2[0]$ and $\mathsf{C}_1[-1] = \mathsf{C}_2[-1]$, we say that chains $\mathsf{C}_1, \mathsf{C}_2$ *span* the same block range.

It is frequently useful to construct a chain containing only the superblocks of another chain. Given C and level μ, the *upchain* $\mathsf{C}\uparrow$ is defined a $\{B \in \mathsf{C} : level(B) \geq \mu\}$. A chain containing only $\mu-$superblocks is calles a $\mu-$superchain. It is also useful, given a $\mu-$superchain to go back to the regular chain C. Given chains $\mathsf{C}' \subset \mathsf{C}$, the *downchain* $\mathsf{C}'\downarrow_{\mathsf{C}}$ is defined as $\{\mathsf{C}[\mathsf{C}'[0] : \mathsf{C}'[-1]]\}$. C is the *underlying chain* of C'. The underlying chain is often implied by context, so we will simply write $\mathsf{C}'\downarrow$. By the above definition, the $\mathsf{C}' \uparrow$ is absolute: $(\mathsf{C} \uparrow^{\mu})^{\mu+i} = \mathsf{C} \uparrow^{\mu+i}$. Given a set of consecutive rounds $S = \{r, r+1, \cdots, r+j\} \subseteq \mathbb{N}$, we define $\mathsf{C}^S = \{B \in \mathsf{C} : B$ was generated during S$\}$.

3.3 Suffix Proofs

NIPoPoWs suffix proofs are used to prove predicates that pertain to the suffix of the blockchain as defined in Definition 6. For example, this is the case of light client synchronization to the longest valid chain.

Definition 6 (Suffix Sensitivity [18])**.** A chain predicate $\mathcal{Q}$ is called k-suffix sensitive if for all chains C, C' with $|\mathsf{C}| \geq k$ and $|\mathsf{C}'| \geq k$ such that $\mathsf{C}[-k:] = \mathsf{C}'[-k:]$ we have that $\mathcal{Q}(\mathsf{C}) = \mathcal{Q}(\mathsf{C}')$.

3.3.1 The Prover

The suffix prover is given in Algorithm 5. The honest prover takes as input an honestly adopted chain C and the security parameters m, k and returns a suffix proof (π, χ), which forms a valid chain regarding the interlink pointers. Keep in mind that we currently assume a soft fork for the protocol deployment, thus for a new block its interlink structure is checked before the block is accepted as valid.

Parameter k roughly pertains to the number of blocks needed to bury a block so that it remains stable in the longest valid chain, in essence that the containing transactions will not be reverted (e.g. $k = 6$). m is the security parameter which sets the lower bound of a superchain's length participating in a NIPoPoW proof. We set $m \geq 2k + 1$ and analyze the meaning and importance of this relation in the suffix proofs' security analysis section.

The proof suffix χ is simply the last k blocks of C. The prefix π is constructed by selecting various blocks of every level $\mu \leq log(\mathsf{C})$ from $\mathsf{C}[-k]$. At the highest possible level μ' at which at least m exist, all these blocks are included. Then from level $\mu' - 1$ the blocks spanning the same range as the last m $\mu'-$superblocks are included. This is inductively followed for every lower level until level 0 is reached. Thus, from this underlying superchain $2m$ blocks will be included in the proof in expectation and always at least m blocks.

Figure 3.4 illustrates an example proof constructed for parameters $m = k = 3$. The top superchain level which contains at least m blocks is level $\mu = 3$. For the $m-$sized suffix of that level, 5 blocks of level 2 are included for support spanning the same range.

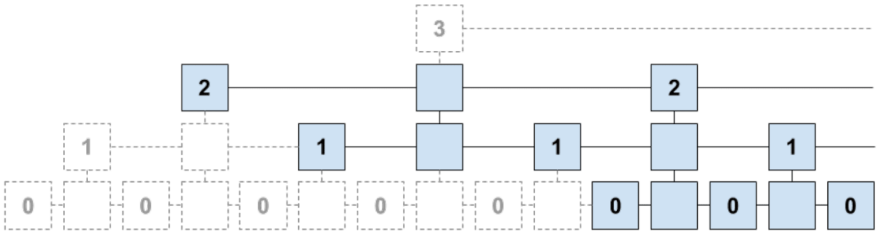

Figure 3.4: Superblock NIPoPoW proof prefix π for $m = 3$ [18].

Algorithm 5 The Suffix Prover for the superblock NIPoPoW protocol [18]

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

3.3.2 The Verifier

The suffix verifier is given in Algorithm 6 and consists of two functions and an operator definition. Function Verify forms a generic verifier for any protocol-specific proof comparison operator $\leq_m$. This proof comparison operator is specifically instantiated for the superblock suffix verifier and utilizes the supporting function $\mathsf{best\text{-}arg}$.

The verifier algorithm is parameterized by a chain predicate $\mathcal{Q}$ and the security parameters k, m. The verifier receives several proofs from different provers which are represented as a collection of proofs $\mathcal{P}$. We consider that at least one received is constructed by an honest prover. Iterating over $\mathcal{P}$ the verifier extracts the best one.

As already described, each proof is a valid chain considering the interlink pointers. For an honest prover the proof contains a subset of blocks of the adopted chain. Proofs consist of two parts π and χ; π is the proof prefix and χ the proof suffix. For honest provers, χ contains the last k blocks of the adopted chain, while π contains a subset of superblocks selected as explained in the prover algorithm.

For each proof the verifier first checks its validity by ensuring that $|\chi| = k$ and that (π, χ) is an anchored chain ($\mathsf{validChain}(\cdot)$). The best known prefix is initialized with the genesis block. At each

loop iteration the verifier compares the next candidate proof prefix π against the currently known best prefix $\tilde{\pi}$ and updates both $\tilde{\pi}, \tilde{\chi}$ if needed.

The $\geq_m$ operator performs the comparison of two proofs. It takes proofs $\pi_{\mathcal{A}}, \pi_B$ and return true if the first one is winning and false otherwise. It first computes the LCA block b between the two proofs. Since the proofs are valid chains, parties $\mathcal{A}, B$ have common underlying chain up to block b, so it suffices to compare the proofs for the diverging chains after b. The verifier selects as level of comparison of each proof the best possible argument by calling the best-arg function. In essence, the verifier selects the level containing the greatest amount of proof-of-work for each proof. This best argument selection by the verifier is called the *principle of charity*. To find the best argument the verifier parses the proof one level at a time, weights the corresponding superblocks and selects the one resulting to the highest total PoW. The weighting of a $\mu-$superchain is performed by estimating the underlying number of blocks as $2^{\mu}|\pi\uparrow^{\mu}\{b:\}|$. As discussed, the highest possible score across all levels is returned. Once the best argument of both proofs is known, they are directly compared and the winner returned. An advantage is given to first proof in case of a tie by using $\geq$ operator favoring $\mathcal{A}$. At the end of the loop the verifier will have determined the best proof $\tilde{\pi}, \tilde{\chi}$.

Note that χ might be needed for the final predicate evaluation but does not aprticipate in any comparison since it is of constant and equal size for any valid proof. We will prove that this proof will belong to an honest prover with high probability in the next section.

Algorithm 6 The Suffix Verifier for the superblock NIPoPoW protocol [18]

function $\mathsf{Verify}^{\mathcal{Q}}_{m,k}(\mathcal{P})$
  $\tilde{\pi} \leftarrow (Gen)$
  for $(\pi, \chi) \in \mathcal{P}$ **do**
    if validChain$(\pi\chi) \wedge |\chi| = k \wedge \pi \geq_m \tilde{\pi}$ **then**
      $\tilde{\pi} \leftarrow \pi$
      $\tilde{\chi} \leftarrow \chi$
    end if
  end for
  return $\tilde{\mathcal{Q}}(\tilde{\chi})$
end function

operator $\pi_{\mathcal{A}} \geq_m \pi_b$
  $b \leftarrow (\pi_{\mathcal{A}} \cap \pi_B)[-1]$
  return $\mathsf{best\text{-}arg}_m(\pi_{\mathcal{A}}, b) \geq \mathsf{best\text{-}arg}_m(\pi_B, b)$
end operator

function $\mathsf{best\text{-}arg}_m(\pi, b)$
  $M \leftarrow \{\mu : |\pi\uparrow^{\mu}\{b:\}| \geq m\} \cup \{0\}$
  return $\max_{\mu \in M}\{2^{\mu} \cdot |\pi\uparrow^{\mu}\{b:\}|\}$
end function

3.4 Security Analysis of Suffix Proofs

3.4.1 Superchain Quality & Suppression Attacks

As explained earlier, every $\mu-$superblock represents 2^{μ} 0-level blocks in expectation, or $\mu-$ superblocks occur approximately once every 2^{μ} blocks. Put in another way, each $\mu-$superchain C has an underlying chain of length $2^{\mu}|\mathsf{C}|$ in expectation. This chain property is referred "superchain quality" or "goodness" of the chain. A formal definition can be found in [18].

It's easy to see that the goodness property holds with overwhelming probability for chains generated in optimistic environments, i.e. if no adversary tries to violate the superblock distribution. However, an adversary can harm superchain quality by performing suppression attack to one or more superblock level. In order to see this consider an adversary $\mathcal{A}$ who breaks the superchain

quality of an honestly adopted chain C_B at level μ as follows. $\mathcal{A}$ starts playing after $|\mathsf{C}_B| \geq 2$ and inspects the appended blocks at every round. If level($\mathsf{C}_B[-1]$) $< \mu$, then $\mathcal{A}$ remains idle. However, if level($\mathsf{C}_B[-1]$) $\geq \mu$ then $\mathcal{A}$ tries to mine an adversarial block b on top of $\mathsf{C}_B[-2]$. If successful she tries to mine one more block b' on top of b. If successful again, she broadcasts b, b'. Now, with non-negligible probability [18] the adversary generates b, b' soon enough, so that honest party B adopts the chain containing them. This would mean that the last μ-superblock is successfully supressed. In any case, the adversary continues her attack by restarting her strategy, inspecting the level of $\mathsf{C}[-1]$ and acting accordingly. An execution of this attack is illustrated in Figure 3.5. It's abvious that several honest μ-superblocks are successfully suppressed causing harm to the superchain quality.

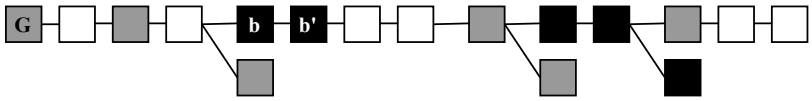

Figure 3.5: Superquality attack. The adversary performs a selfisj mining [7] attack (black blocks) whenever an honest μ-superblock (grey) is mined. The attack affects the distribution of μ-superblocks in the honest chain [18].

In parallel to performing this attack the adversary may also maintain a fork chain. Assuming honest majority, the adversary's fork chain will be shorter than the honest parties' chain at 0-level. However, the adversary's μ-superchain might be longer than that of the honest parties' because of the superquality attack on the honest chain. Thus the adversary may successfully fool the verifier to favor her proof. As formally proven in [18] the adversary can produce a winning suffix proof for her fork chain with non-negligible probability.

The modified proof construction is given in Algorithm 7 and allows the prover to take superchain goodness into account. More in detail, at each step the prover checks the μ-superblock distribution quality in the subchain α before moving on to the lower superchain level. If goodness is indeed maintained at the current level then the prover regularly covers the span of the last m μ-level blocks with blocks of level $\mu - 1$. Otherwise, if goodness is violated for level μ then the prover completely ignores this level and tries to span the whole range of subchain α with blocks of the lower level $\mu - 1$.

Algorithm 7 The *goodness aware* Suffix Prover for the superblock NIPoPoW protocol [18]

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

3.4.2 Security of Suffix Proofs

In this section we revise the security proof for the Superblock NIPoPoWs suffix proof protocol provided in [18]. In particular we revise Lemma 1 which lies in the heart of the security proof, as well as the security proof itself (Theorem 2). We also repeat any additional definition and lemma needed for the security proof, aiming to provide extended and intuitive explanation for each one of them. Finally, we formally calculate a lower bound for the security parameter m as a function of k.

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 6 (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 $X^\mu(S) \geq k$ for some security parameter k. Then $|S| \geq (1-\epsilon)2^\mu \frac{k}{pq(n-t)}$ with overwhelming probability.

Chain regularity. No insertions, no copies and no predictions [9] have occurred.

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

Proof. **Block counts and regularity.** We refer to [9] for the full proof.

Round count. First, observe that for a specific round r we have $X_r \sim Bern(p)$, so for the μ-level superblocks $X_r^\mu \sim Bern(2^{-\mu}p)$ and these are independent trials. Therefore, since for $|S|$ rounds we have $(n-t)q|S|$ RO queries, we have that $X_S^\mu \sim \text{Bin}((n-t)q|S|, 2^{-\mu}p)$. So $(n-t)q|S| \sim \text{NB}(X_S^\mu, 2^{-\mu}p)$. Negative Binomial distribution is defined as $\text{NB}(r, p')$ and expresses the expected 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 altogether. 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 $\mathbb{E}[(n-t)q|S|] = \frac{X_S^\mu}{2^{-\mu}p} \Rightarrow \mathbb{E}[|S|] = 2^\mu \frac{X_S^\mu}{(n-t)qp}$. If $X^\mu(S) \geq k$ then we have that $\mathbb{E}[|S|] \geq 2^\mu \frac{k}{(n-t)qp}$ and we can apply a tail bound to the negative binomial distribution, so we obtain that $\Pr[|S| < (1-\epsilon)\mathbb{E}(|S|)] \in \Omega(\epsilon^2 m)$. □

The following lemma lies in the heart of the formal security proof (Theorem 2). The revision of this lemma is one of our major contributions in this context, as it leads to revisions in the security proof theorem as well.

Lemma 1. Suppose S is a set of consecutive rounds $r_1 \cdots r_2$ and C_B is a chain adopted by an honest party at round r_2 of a typical execution. Let $\mathsf{C}_S^B = \{$ b $\in \mathsf{C}_B$: b was generated during $S\}$. Let $\mu_\mathcal{A}, \mu_B \in \mathbb{N}$. Suppose $\mathsf{C}_S^B{\uparrow}^{\mu_B}$ is good and suppose that $|\mathsf{C}_B^S{\uparrow}^{\mu_B}| \geq k$. Suppose C_A is a $\mu_\mathcal{A}$-superchain containing only adversarial blocks generated during S. Then $2^{\mu_\mathcal{A}}|\mathsf{C}_\mathcal{A}| < 2^{\mu_B}|\mathsf{C}_S^B{\uparrow}^{\mu_B}|$.

Proof. From $|\mathsf{C}_B^S{\uparrow}^{\mu_B}| \geq k$ we have that $|X_S^{\mu_B}| \geq k$. Applying Theorem 1, we conclude that

$$|S| \geq (1-\epsilon)2^{\mu_B}\frac{|\mathsf{C}_B^S{\uparrow}^{\mu_B}|}{pq(n-t)}. \tag{3.1}$$

We also know from the goodness of $\mathsf{C}_B^S{\uparrow}^{\mu_B}$ that

$$|\mathsf{C}_B^S{\uparrow}^{\mu_B}| \geq (1-\delta)2^{-\mu_B}|\mathsf{C}_B^S| \tag{3.2}$$

So we have

$$|S| \geq (1-\epsilon)(1-\delta)\frac{|\mathsf{C}_B^S|}{pq(n-t)} \tag{3.3}$$

For the number of $\mu_{\mathcal{A}}$-blocks that the adversary generated during S we know that $|\mathsf{C}_{\mathcal{A}}| \leq (1+\epsilon)2^{-\mu_{\mathcal{A}}}Z(S)$. But we know that $Z(S) < \frac{t}{n-t}\cdot\frac{f}{1-f}|S| + \epsilon f|S|$ so we have that

$$|\mathsf{C}_{\mathcal{A}}| < (1+\epsilon)2^{\mu_{\mathcal{A}}}(\frac{t}{n-t}\cdot\frac{f}{1-f}+\epsilon f)|S| \tag{3.4}$$

By substituting 3.3 in 3.4 we have that

$$|\mathsf{C}_{\mathcal{A}}| < (1+\epsilon)^2(1-\delta)2^{-\mu_{\mathcal{A}}}(\frac{t}{n-t}\cdot\frac{f}{1-f}+\epsilon f)\frac{|\mathsf{C}_B^S|}{pq(n-t)} \tag{3.5}$$

But from 3.2 we also know that $|\mathsf{C}_B^S| \leq \frac{2^{\mu_B}|\mathsf{C}_B^S\uparrow^{\mu_B}|}{1-\delta}$ and by substituting to 3.5 we have that

$$2^{\mu_{\mathcal{A}}}|\mathsf{C}_{\mathcal{A}}| \leq (1+\epsilon)^2(\frac{t}{n-t}\cdot\frac{f}{1-f}+\epsilon f)\frac{2^{\mu_B}|\mathsf{C}_B^S\uparrow^{\mu_B}|}{pq(n-t)} \tag{3.6}$$

So we conclude to $2^{\mu_{\mathcal{A}}}|\mathsf{C}_{\mathcal{A}}| < 2^{\mu_B}|\mathsf{C}_S^B\uparrow^{\mu_B}|$ considering that $\frac{f/(1-f)}{pq(n-t)} < \frac{1}{(1-\delta)(1+\epsilon)^2}$. □

We will now define the *adequate level* of an honest suffix proof and some resulting properties, that will be used in the formal security proof.

Definition 7 (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.

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*[18] 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. □

The intuition behind the adequate level is the following. Adequate is the level μ' of a proof π with respect to block b, if this level is of good chain quality and doesn't miss any μ'-superblock coming after b. Because of the goodness-aware prover algorithm 7 each proof π has an adequate level for every block $b \in \pi$. Note that the adequate level is used in Claim 1a of the Security Proof (Theorem 2).

The following lemma is not used in our revised version of the security proof (Theorem 2). However, we include it in this section since we have fixed some minor mistakes of [18]. We also believe that it provides intuition regarding the complicated notion of adequate level.

Lemma 3. Suppose the verifier evaluates $\pi_{\mathcal{A}} \geq \pi_B$ 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 $b = LCA(\pi_{\mathcal{A}}, \pi_B)$ 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 $2^{\mu_B}|\pi\{b:\}\uparrow^{\mu_B}| < (1-\delta)|\mathsf{C}\{b:\}|$.
But also μ'_B satisfies goodness, so $2^{\mu'_B}|\pi\{b:\}\uparrow^{\mu'_B}| > (1-\delta)|\mathsf{C}\{b:\}|$.
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. □

The intuition behind the above lemma is the following. The comparison level chosen by the verifier can be no other than the adequate level in respect to block $LCA(\pi_A, \pi_B)$, since any other choice would be a level of non-good quality, because of the definition of the adequate level. So, in fact, a more accurate lemma should claim that $\mu'_B = \mu_B$.

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

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

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

Let $b_0 = LCA(\pi_{\mathcal{A}}, \pi_B)$. Let $b^\star$ be the most recently honestly generated block in C_B preceding b. Note that $b^\star$ necessarily exists because $\mathsf{Genesis}$ is honestly generated. Let the levels of comparison decided by the verifier be $\mu_{\mathcal{A}}$ and μ_B respectively. Let μ'_B be the adequate level of proof π_B with respect to block b_0. Call $\alpha_{\mathcal{A}} = \pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}\{b_0:\}$, $\alpha'_B = \pi_B\uparrow^{\mu'_B}\{b_0:\}$.

Note that the proof parts succeeding block b_0 are the decisive ones for the verifier's final 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 PoW in her proof for this part of the chain.

Our proof construction is based on the following intuition: we consider that $\alpha_{\mathcal{A}}$ consists of three distinct parts $\alpha^1_{\mathcal{A}}, \alpha^2_{\mathcal{A}}, \alpha^3_{\mathcal{A}}$ with the following properties. Block $b_0 = LCA(\pi_{\mathcal{A}}, \pi_B)$ is the fork point between $\pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}, \pi_B\uparrow^{\mu_B}$, as already defined. Let block $b_1 = LCA(\alpha_{\mathcal{A}}, \alpha'_B\downarrow)$ be the fork point between $\alpha_{\mathcal{A}}$ and C_B as an honest prover could observe. Then part $\alpha^1_{\mathcal{A}}$ contains the blocks between b_0 exclusive and b_1 inclusive, generated during the set of consecutive rounds $\mathcal{S}_1$ and also $|\alpha^1_{\mathcal{A}}| = k_1$. Consider b_2 the last block in $\alpha_{\mathcal{A}}$ that was generated by an honest party. Part $\alpha^2_{\mathcal{A}}$ contains the blocks between b_1 exclusive and b_2 inclusive generated during the set of consecutive rounds $\mathcal{S}_2$ and $|\alpha^2_{\mathcal{A}}| = k_2$. Consider b_3 the next block of b_2 in $\alpha_{\mathcal{A}}$. Then $\alpha^3_{\mathcal{A}} = \alpha_{\mathcal{A}}[b_3:]$ and $|\alpha^3_{\mathcal{A}}| = k_3$ consisting of adversarial blocks generated during the set of consecutive rounds $\mathcal{S}_3$. Therefore $|\alpha_{\mathcal{A}}| = k_1 + k_2 + k_3$ and we will show that $|\alpha_{\mathcal{A}}| < |\alpha_B|$.

We will now show three successive claims: First, $\alpha^1_{\mathcal{A}}$ and $\alpha'_B\downarrow$ have only a few blocks in common. Second, $a^2_{\mathcal{A}}$ contains only a few blocks. And third, the adversary is able to produce this $a_{\mathcal{A}}$ with negligible probability.

Claim 1: $\alpha_{\mathcal{A}}, \alpha'_B{\downarrow}$ have only a few blocks in common.

We show this by taking the two possible cases for the relation of $\mu_{\mathcal{A}}$, μ'_B.

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

Claim 1b: If $\mu'_B > \mu_{\mathcal{A}}$ then $|\alpha_{\mathcal{A}}[1:] \cap \alpha'_B{\downarrow}\ [1:]| = k_1$ where $k_1 \leq 2^{\mu'_B - \mu_{\mathcal{A}}}$.

Let's call b' the first block in α'_B after block b_0. Suppose for contradiction that $k_1 > 2^{\mu'_B - \mu_{\mathcal{A}}}$. Since $\mathsf{C}_B^{\mu'_B}$ is of good chain quality, this would mean that block b', of level μ'_B is also of level $\mu_{\mathcal{A}}$. Since it is of level $\mu_{\mathcal{A}}$ the adversary could include it in the proof but b' cannot exist in both $\alpha_{\mathcal{A}}, \alpha'_B$ since $\alpha_{\mathcal{A}} \cap \alpha'_B = \emptyset$ by definition. In case that the adversary chooses not to include b' in the proof then she can include no other blocks of C_B in her proof, since it would not consist a valid chain. Thus wehave reached to a contradiciton in either way.

From Claim 1a and Claim 1b, we conclude that there are $|\alpha_{\mathcal{A}}| - k_1$ blocks after block b_0 in $\alpha_{\mathcal{A}}$ which do not exist in $\alpha_B{\downarrow}$.

The intuition behind Claim 1b is that the common blocks of $\alpha^1_{\mathcal{A}}, \alpha'_B{\downarrow}$ may only be blocks of level $\mu_{\mathcal{A}}$ which precede the first μ'_B block appearing in α'_B. If b' is included in the adversary's proof, then b' would be the LCA of $\alpha_{\mathcal{A}}, \alpha'_B$, which violates the minimality of b_0. If it is not, then the adversary could include no more blocks from the common part of chain C_B in her proof, since they would no longer form a valid interlinked chain in $\alpha_{\mathcal{A}}$. The quantity $2^{\mu'_B - \mu_{\mathcal{A}}}$ should be thought as follows: in the range between two consequent μ'_B-level blocks, we have $n = 2^{\mu'_B}$ 0-level blocks and, thus, $2^{-\mu_{\mathcal{A}}} n = 2^{\mu'_B - \mu_{\mathcal{A}}}$ blocks of $\mu_{\mathcal{A}}$-level.

Before stepping into Claim 2 remember that we have defined block $b_1 = LCA(\alpha'_B{\downarrow}, \alpha_{\mathcal{A}})$. This makes b_1 the last block before the 0-level fork point, included in the adversary's proof.

Claim 2: $a^2_{\mathcal{A}}$ contains only a few blocks.

We show this by showing that $\alpha_{\mathcal{A}}[k_1 + k_2 + 1:]$ contains no honestly generated blocks due to the Common Prefix property. Suppose for contradiction that the block $\alpha_{\mathcal{A}}[i]$ for some $i \geq k_1 + k_2 + 1$ was honestly generated. This means that an honest party adopted the chain $\alpha_{\mathcal{A}}[:i-1]{\downarrow}$ at some round $r_2 \leq r_3$. Because of the way honest parties adopt chains, the superchain $\alpha_{\mathcal{A}}[:i-1]$ has an underlying properly constructed 0-level anchored chain $\mathsf{C}_{\mathcal{A}}$ such that $\alpha_{\mathcal{A}}[:i-1] \subseteq \mathsf{C}_{\mathcal{A}}$. Let j be the index of block b_1 within $\alpha_{\mathcal{A}}$, $j_{\downarrow}$ be the index of block b_2 within $\mathsf{C}_{\mathcal{A}}$ and $k_{2{\downarrow}} = |\alpha_{\mathcal{A}}[j : j + k_2]{\downarrow}\ |$. See Figure 3.6 for a demonstration. Observe that $|\mathsf{C}_{\mathcal{A}}[:\{\alpha_{\mathcal{A}}[i-1]\}]| \geq |\mathsf{C}_{\mathcal{A}}[:j_{\downarrow} + k_{2{\downarrow}}]|$, while $\mathsf{C}_{\mathcal{A}}[j_{\downarrow} : j_{\downarrow} + k_{2{\downarrow}}] \not\preceq \mathsf{C}_B$ as proved in Claim 1. But $\mathsf{C}_{\mathcal{A}}$ 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[9] with parameter $k_{2{\downarrow}}$. It follows that with overwhelming probability in $k_{2{\downarrow}}$, the $k_3 = |\alpha_{\mathcal{A}}| - k_2 - k_1$ last blocks of the adversarial proof have been adversarially generated.

The intuition behond Claim 2 is that because of Common Prefix on $k_{2{\downarrow}}$ parameter, where $k_{2{\downarrow}} = |\alpha_{\mathcal{A}}[j : j + k_2]{\downarrow}\ |$, and because $\mathbb{E}[k_{2{\downarrow}}] = 2^{\mu_{\mathcal{A}}} k_2$, there can be no honest party adopting $\mathsf{C}_{\mathcal{A}}$ at any round $i \geq k_1 + k_2 + 1$.

Before stepping into Claim 3 remember that we have defined block b_2 the last honestly generated block in $\alpha_{\mathcal{A}}$. b_2 is equal to $b^\star$ if no such block exists in $\alpha_{\mathcal{A}}$. Remember also that b_3 is the next block of b_2 in $\alpha_{\mathcal{A}}$, as shown in Figure 3.6.

Claim 3: Adversary $\mathcal{A}$ is able to produce $\alpha_{\mathcal{A}}$ that wins against α_B with negligible probability.

Let r_1 the round when b_3 was generated. Consider the set S_3 of consecutive rounds $r_1..r_3$. Every block in $\alpha_{\mathcal{A}}[-k_3:]$ has been adversarially generated during S_3 and $|\alpha_{\mathcal{A}}[-k_3:]| = |\alpha_{\mathcal{A}}\{b_3:\}| = k_3$. 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^S_B, we see that if b' is the most recently generated block in α_B in a round $r \leq r_1$, then $\mathsf{C}^S_B = \mathsf{C}_B\{b':\}$. But $\mathsf{C}^S_B{\uparrow}^{\mu'_B}$ is good with respect to C^S_B. Applying Lemma 1,

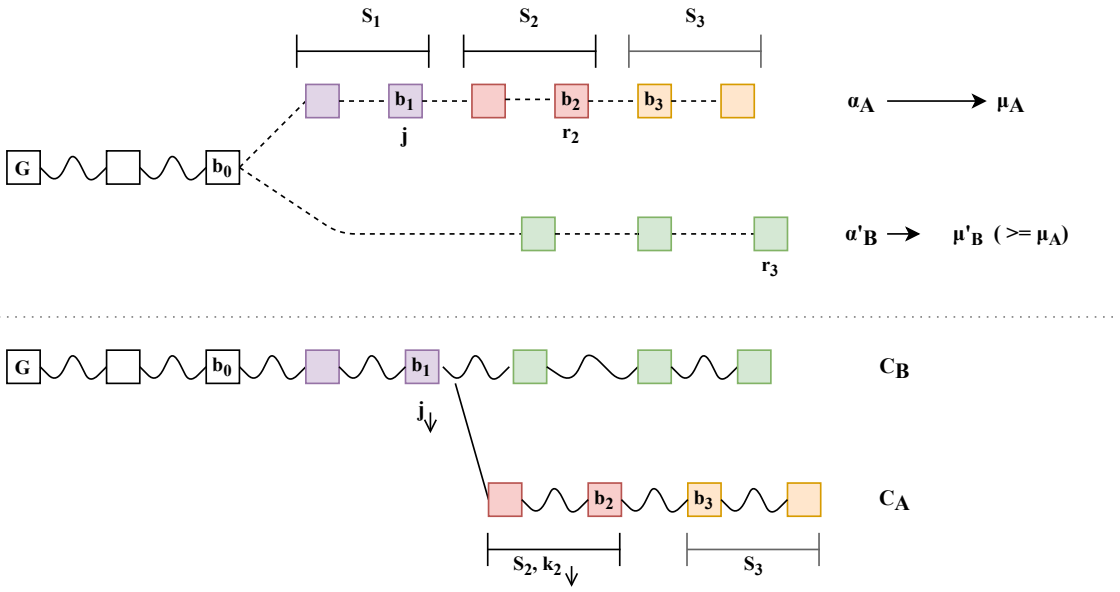

Figure 3.6: Two competing proofs at different levels. At the bottom the corresponding 0-level chains are represented.

we obtain that with overwhelming probability $2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}\{b_3:\}| < 2^{\mu'_B}|\mathsf{C}_B^S\uparrow^{\mu'_B}|$, which is equal to

$$2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}\{b_3:\}| < 2^{\mu'_B}|\alpha'_B\{b':\}| \tag{3.7}$$

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

In order to complete the proof, let us now consider $\alpha^1_{\mathcal{A}}$, $\alpha^2_{\mathcal{A}}$, $\alpha^3_{\mathcal{A}}$ and α'^1_B, α'^2_B, α'^3_B the parts of the proofs containing the blocks generated during S_1, S_2, S_3 correspondingly as illustrated in Figure 3.7.

Subsequently to the above Claims we have that:

Because of the common underlying chain during the first round set S_1:

$$2^{\mu_{\mathcal{A}}}|\alpha^1_{\mathcal{A}}| \leq 2^{\mu'_B}|{\alpha'^1}_B| \tag{3.8}$$

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 S_2:

$$2^{\mu_{\mathcal{A}}}|\alpha^2_{\mathcal{A}}| \leq 2^{\mu'_B}|{\alpha'^2}_B| \tag{3.9}$$

Because of Equation (1), we have for the third round set S_3:

$$2^{\mu_{\mathcal{A}}}|\alpha^3_{\mathcal{A}}| < 2^{\mu'_B}|{\alpha'^3}_B| \tag{3.10}$$

Subsequently we have

$$2^{\mu_{\mathcal{A}}}(|\alpha^1_{\mathcal{A}}| + |\alpha^2_{\mathcal{A}}| + |\alpha^3_{\mathcal{A}}|) < 2^{\mu'_B}(|{\alpha'^1}_B| + |{\alpha'^2}_B| + |{\alpha'^3}_B|)$$

and finally

$$2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}| < 2^{\mu'_B}|{\alpha'}_B| \tag{3.11}$$

Therefore we have proven that $2^{\mu'_B}|\pi_B\uparrow^{\mu'_B}| > 2^{\mu_{\mathcal{A}}}|\pi_{\mathcal{A}}^{\mu_{\mathcal{A}}}|$. 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 $2^{\mu_B}|\pi_B\uparrow^{\mu_B}| > 2^{\mu_{\mathcal{A}}}|\pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}|$.

□

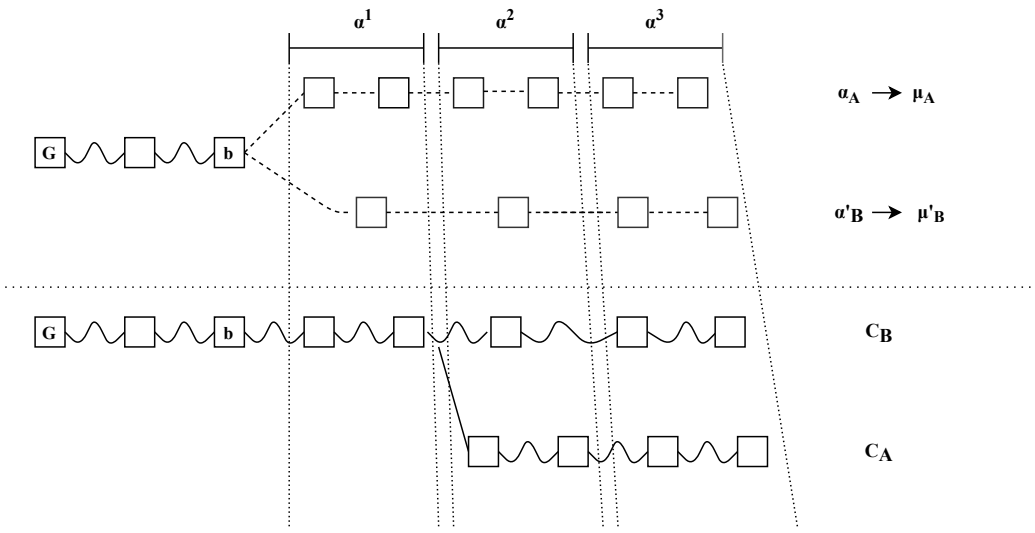

Figure 3.7: The three round sets in two competing proofs at different levels. The vertical dashed lines denote the area of interest, across proofs and chains, corresponding to each round set. At the bottom the corresponding 0-level chains are represented.

It remains to calculate 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 S_1, S_2, S_3, so that the proof equations 3.8, 3.9, 3.10 hold and then sum these values to obtain parameter m.

In the first set of rounds S_1 we need one block to be included in α_B to "cover" the blocks included in $\alpha^1_{\mathcal{A}}$ in order to assure Equation 3.8. In the second set of rounds, S_2, we need $2^{-\mu_B}k$ blocks in α_B to cover the blocks included in $\alpha^2_{\mathcal{A}}$ in order to assure Equation 3.9. This is a direct result of the Common Prefix property. However, in order to make m independent of any specific level it suffices to require k blocks for S_2. In the last set of rounds, S_3, we need at least k adversarially generated blocks in α_B so that Lemma 1 is applicable.

By adding these together we finally conclude to the following lower bound for the value of the security parameter:

$$m = 2k + 1 \tag{3.12}$$

3.5 Infix Proofs

Since we do not provide any new contribution to the infix proofs under soft fork conditions, we provide the algorithms for the full infix proofs construction as well as a high level description of the construction. These are needed in order to keep up with the novelties described later in the velvet infix proofs section. For a more detailed presentation of this construction, the reader should refer to [18]. Most of this section's content is borrowed from [18].

By requesting a suffix proof a client can synchronize to the latest valid chain or learn about information that can be extracted by the last k block of the chain. Nevertheless, suffix proofs act as the stepping stone for the construction of another useful class of proofs, which answer to more general predicates that can depend on multiple blocks buried deep within the blockchain.

More specifically, an *infix proof* allows proving any predicate $Q(\mathsf{C})$ that depends on a number of blocks that may appear anywhere within the chain (excpet for the last k block for stability reasons). These blocks consitute a subset C' of blocks, the *witness*, which may not necesserily form a stand-alone subchain. This allows proving useful queries such as, whether a transaction is confirmed.

The following definition formally states the notion of an *infix-sensitive* predicate.

Definition 7 (Infix Sensitivity)**.** A chain predicate Q_d, k is infix sensitiveif it can be written in the form

$$Q_{d,k}(\mathsf{C}) = \begin{cases} \text{true, if } \exists \mathsf{C}' \subseteq \mathsf{C}[:-k] : |\mathsf{C}'| \leq d \wedge D(\mathsf{C}') \\ \text{false, otherwise} \end{cases}$$

Where D is an arbitrary computable predicate.

The construction of infix proofs is given in Algorithms 8, 9. The infix prover accepts as parameters the full blockchain C and the subset C' which contains the blocks of interest fro answering a specific predicate in question. the prover calls the suffix prover algorithm to produce a proof (π, χ) for C. Then, the infix prover adds some auxiliary blocks to the poof prefix π, ensuring that these auxiliary blocks form a chain with the rest of the blocks in π. Such auxiliary blocks are collected as follows. For every block of interest B of the subset C', the immediate previous (E') and immediate next (E) superblocks in π are found. Then, a chain R which connects E back to B' is found by the algorithm followDown. B' contains a pointer to E' in its interlink completing the chain. Observe that B' necessarily includes a pointer to E' because of the choice of E'.

Algorithm 8 The Infix Prover for the superblock NIPoPoW protocol [18]

```
1: function ProveInfix_{m,k}(C, C', depth)
2:     (π, χ) ← Prove_(m, k)(C)
3:     for B' ∈ C' do
4:         for E ∈ π do
5:             if depth(E) ≥ depth(B') then
6:                 R ← followDown(E, B', depth)
7:                 aux ← aux ∪ R break
8:             end if
9:             E' ← E
10:        end for
11:    end for
12:    return (aux ∪ π, χ)
13: end function
```

The followDown algorithm proceeds as follows. Starting from block $hi = E$, it tries to follow a pointer to as far as possible without surpassing the block of interest B'. Any pointer that surpasses B' is aborted and a pointer of lower is tried, which causes a smaller step within the skiplist. If a pointer was folllowed without surpassing B', the operation continues from the new pointed block, until eventually B' is reached. An example execution is illustrated in Figure 3.8.

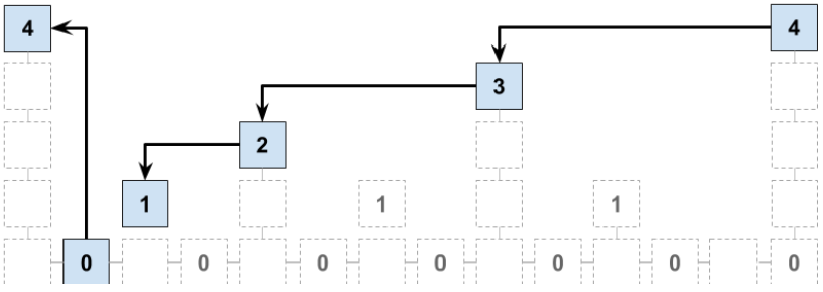

Figure 3.8: An infix proof descend. Only blue blocks are included in the proof.Blue blocks of level 4 are part of π, while the blue blocks of level 1 to 3 are produced by the followDown to get to the block of level 0, which is part of C'.

The verification algorithm is given in Algorithm 10. The algorithm calls the suffix verifier. It maintains a block-DAG collecting blocks from all proofs received, i.e. all proofs in $\mathcal{P}$. This DAG is maintained in the blockById hashmap. Using it, ancestors uses simple graph search to extract the set of ancestor blocks of a block of interest. In the final predicate evaluation the set of ancestors of the best chain tip's is passed to the predicate. The ancestors are included to avoid an adversary who presents an honest chain but skips the block of interest from the auxiliary blocks in the infix proof.

Algorithm 9 The followDown function which produces the necessary blocks to connect a superblock hi to a preceding regular block lo [18]

function ProveInfix$_{m,k}$(C, C′, depth)
  $B \leftarrow hi$
  $aux \leftarrow [\,]$
  $\mu \leftarrow level(hi)$
  while $B \neq lo$ **do**
    $B' \leftarrow$ blockById[B.interlink[μ]]
    if depth[B'] < depth[lo] **then**
      $\mu \leftarrow \mu - 1$
    else
      $aux \leftarrow aux \cup B$
      $B \leftarrow B'$
    end if
  end while
  return aux
end function

Algorithm 10 The verify algorithm for the superblock NIPoPoW infix protocol [18]

function ancestors(B, blockById)
  if $B = Gen$ **then**
    return $\{B\}$
  end if
  C $\leftarrow \emptyset$
  for $B' \in B$.interlink **do**
    C $\leftarrow$ C $\cup$ ancestors(B')
  end for
  return C $\cup B$
end function

function VERIFY-INFIX$^{D}_{l,m,k}(\mathcal{P})$
  blockById $\leftarrow \emptyset$
  for $(\pi, \chi) \in \mathcal{P}$ **do**
    for $B \in \pi$ **do**
      blockById[$B.id$] $\leftarrow B$
    end for
  end for
  $\tilde{\pi} \leftarrow$ best $\pi \in \mathcal{P}$ according to suffix verifier
  return D(ancestors($\tilde{\pi}[-1]$, blockById))
end function

3.6 Succinctness

As background knowledge and in order to show the final contribution of the presented construction we will now refer to the succinctness of the produced proofs. We will explain the major shortcomings of superblock NIPoPoWs and an estimation for the proofs' length in average case scenarios. The reader should refer to [18] for a detailed analysis. The content of this section is also mostly borrowed by [18].

The basic weakness of the superblock NIPoPow construction results from the importance of the rare "lucky" blocks in the chain and the suppression attacks that the adversary can perform on them, as described in a previous section. In essence, the adversary could perform such an attack on a target level μ rather easy, since μ-superblock appear rarely in the chain and the adversary can suppress them with only one or two blocks of any lower level. Thus, by generating less lucky blocks the adversary has the chance to abort a very lucky honestly generated block.

Therefore, full succinctness in the standard honest majority model is impossible to prove for superblock NIPoPoWs. Recall that by reducing superquality through suppression attacks for a sufficiently low level μ the adversary can cause the honest prover to digress in their proofs from high level superchains down to low-level superchains, causing a linear proofs to be produced. For instance, if superquality is harmed at $\mu = 3$, the prover will need to digress down to level $\mu = 2$ and include the whole 2-superchain, which will be of size $|\mathsf{C}|/2$ in expectation.

The following theorem gives the succinctness estimation for the particular "optimistic" case where the adversary does not use their (minority) computational power or network power.

Theorem 1 (Optimistic succinctness)**.** *If all players are honest and the network scheduling is random, superblock NIPoPoWs produced by honest provers are succinct with the number of blocks bounded by* $4mlog(|\mathsf{C}|)$*, with overwhelming probability in* m*.*

Chapter 4

Superblocks under Velvet Fork

4.1 Velvet Interlinks

More recently, velvet forks have been introduced [32]. In a velvet fork, blocks created by upgraded miners (called *velvet blocks*) are accepted by unupgraded miners as in a soft fork. Additionally, blocks created by unupgraded miners are also accepted by upgraded miners. This allows the protocol to upgrade even if only a minority of miners chooses to upgrade. To maintain backwards compatibility and to avoid causing forks, the additional data included in a block is *advisory* and must be accepted whether it exists or not. Even if the additional data is invalid or malicious, upgraded nodes (in this context also called *velvet nodes*) are forced to accept the blocks. The simplest approach to velvet fork the chain for interlinking purposes is to have upgraded miners include the interlink pointer in the blocks they produce, but accept blocks with missing or incorrect interlinks. As we show in the next section, this approach is flawed and susceptible to unexpected attacks. A surgical change in the way velvet blocks are produced is necessary to achieve proper security.

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

Definition 8 (Velvet Parameter)**.** The *velvet parameter* g is defined as the percentage of honest parties that have upgraded to the new protocol. The absolute number of honest upgraded parties is denoted n_h and it holds that $n_h = g(n - t)$.

Velvet forks maintain backwards and forwards compatibility. This requires any block produced by upgraded miners to be accepted by unupgraded nodes (as in a soft fork), but also blocks produced by unupgraded miners to be accepted by upgraded nodes. For the particular case of superblock NIPoPoWs under velvet forks, upgraded miners must include the interlink data structure within their blocks, but must also accept blocks missing the interlink structure or Velvet forks maintain backwards and forwards compatibility. This requires any block produced by upgraded miners to be accepted by unupgraded nodes (as in a soft fork), but also blocks produced by unupgraded miners to be accepted by upgraded nodes. containing an invalid interlink. Unupgraded honest nodes will produce blocks that contain no interlink, while upgraded honest nodes will produce blocks that contain truthful interlinks. Therefore, any block with invalid interlinks will be adversarially generated. However, such blocks cannot be rejected by the upgraded nodes, as that would give the adversary an opportunity to cause a hard fork.

A block generated by the adversary can thus contain arbitrary data in the interlink and yet be adopted by an honest party. Because the honest prover is an upgraded full node, it can determine what the correct interlink pointers are by examining the whole previous chain, and can thus deduce whether a block contains invalid interlink data. In that case, the prover can simply treat such blocks as unupgraded. 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.

In any velvet protocol, a specific portion within a block, which is treated as a comment by unupgraded nodes, is reused to contain auxiliary data by upgraded miners. Because these auxiliary

data can be deterministically calculated, upgraded full nodes can verify the authenticity of the data in a new block they receive. We distinguish blocks based on whether they follow the velvet protocol rules or they deviate from them.

Definition 9 (Smooth and Thorny blocks)**.** A block in a velvet protocol upgrade is called *smooth* if it contains auxiliary data and the data corresponds to the honest upgraded protocol. A block is called *thorny* if it contains auxiliary data, but the data differs from the honest upgraded protocol. A block can be neither smooth nor thorny if it does not contain auxiliary data.

In the case of velvet forks for interlink purposes, the auxiliary data consists of the Merkle Tree containing the interlink pointers to the most recent superblock ancestor at every level μ.

4.1.1 A naïve velvet scheme.

In previous work [18], it was conjectured that superblock NIPoPoWs remain secure under a velvet fork. We call this scheme the *Naïve Velvet NIPoPoW* protocol, because it is not dissimilar from the NIPoPoW protocol in the soft fork case. In particular, the naïve velvet NIPoPoW protocol that was put forth works as follows. Each upgraded honest miner attempts to mine a block b that includes interlink pointers in the form of a Merkle Tree included in its coinbase transaction. For each level μ, the interlink contains a pointer to the most recent among all the ancestors of b that have achieved at least level μ, regardless of whether the referenced block is upgraded or not and regardless of whether its interlinks are valid. Unupgraded honest nodes will keep mining blocks on the chain as usual; because the status of a block as superblock does not require it to be mined by an upgraded miner, the unupgraded miners contribute mining power to the creation of superblocks as desired.

The prover in the naïve velvet NIPoPoWs then worked as follows. The honest prover constructed the NIPoPoW proof $\pi\chi$ as usual by selecting certain superblocks from his chain C as representatives in π and by setting $\chi = \mathsf{C}[-k:]$. The outstanding issue in this case, however, is that these blocks in π do not form a chain because, while superblocks, some of them may not be upgraded and they may not contain any pointers (or they may contain invalid pointers). The honest prover needs to provide a connection between two consecutive blocks $\pi[i+1]$ and $\pi[i]$ in the superchain, and suppose $\pi[i]$ is the most recent μ-superblock preceding $\pi[i+1]$. The block $\pi[i+1]$ is a superblock and exists at some position j in the underlying chain C of the prover, i.e., at $\pi[i+1] = \mathsf{C}[j]$. If $\mathsf{C}[j]$ is a smooth block, then the interlink pointer at level μ within it can be used directly. Otherwise, the prover used the *previd* pointer of $\pi[i+1] = \mathsf{C}[j]$ to repeatedly reach the parents of $\mathsf{C}[j]$, namely $\mathsf{C}[j-1], \mathsf{C}[j-2], \cdots$ until a smooth block b between $\pi[i]$ and $\pi[i+1]$ was found in C. The block b then contains a pointer to $\pi[i]$, as $\pi[i]$ is also the most recent μ-superblock ancestor of b. The blocks $\mathsf{C}[j-1], \mathsf{C}[j-2], \cdots, b$ are then included in the proof to illustrate that $\pi[i]$ is an ancestor of $\pi[i+1]$.

The argument for why the above scheme work is as follows. First of all, the scheme does not add many new blocks to the proof. In expectation, if a fully honestly generated chain is processed, after in expectation $\frac{1}{g}$ blocks have been traversed, a smooth block will be found and the connection to $\pi[i]$ will be made. Thus, the number of blocks needed in the proof increases by a factor of $\frac{1}{g}$. Security was argued as follows: An honest party includes in their proof as many blocks as in a soft forked NIPoPoW, albeit by using an indirect connection. The crucial feature is that it is not missing any superblocks. Even if the adversary creates interlinks that skip over some honest superblocks, the honest prover will not utilize these interlinks, but will use the "slow route" of level 0 instead. The adversarial prover, on the other hand, can only use honest interlinks as before, but may also use false interlinks in blocks mined by the adversary. However, these false interlinks cannot point to blocks that are of incorrect level. The reason is that the verifier can look at the hash of each block to verify its level and therefore cannot be lied to. The only problem a fake interlink can cause is that it can point to a μ-superblock which is not *the most recent ancestor*, but some other block. It was then argued that the only other possibility was to point to blocks that are older μ-superblock ancestors in the same chain, as illustrated in Figure 4.1. However, the adversarial prover can only harm herself by making use of these pointers, as the result will simply by a superchain with fewer blocks.

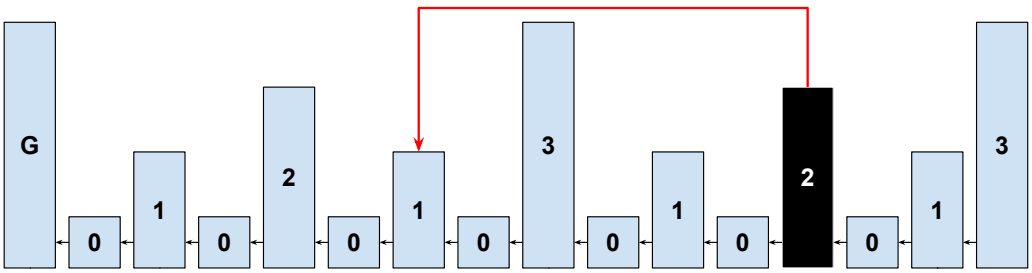

Figure 4.1: A thorny pointer of an adversarial block, colored black, in an honest party's chain. The thorny block points to a 1-superboock which is an ancestor 1-superblock, but not *the most recent ancestor* 1-superblock.

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

4.2 The Chainsewing Attack

We now make the critical observation that a thorny block can include interlink pointers to blocks that are not its own ancestors in the 0-level chain. Because it must contain a pointer to the hash of the block it points to, they must be blocks that have been generated previously, but they may belong to a different 0-level chain. This is shown in Figure 4.2.

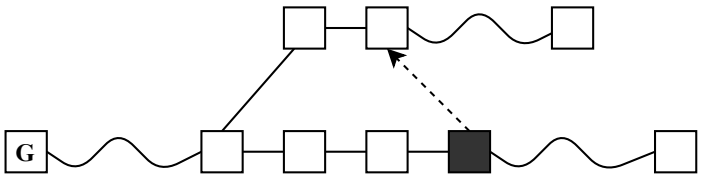

Figure 4.2: A thorny block, colored black, in an honest party's chain, uses its interlink to point to a fork chain.

In fact, as the interlink vector contains multiple pointers, each pointer may belong to a different fork. This is illustrated in Figure 4.3. The interlink pointing to arbitrary directions resembles a thorny bush.

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

Assume chain C_B was adopted by an honest party B and chain $\mathsf{C}_{\mathcal{A}}$, a fork of C_B at some point, is maintained by the adversary $\mathcal{A}$. After the fork point $b = (\mathsf{C}_B \cap \mathsf{C}_{\mathcal{A}})[-1]$, the honest party produces a block extending b in C_B containing a transaction tx. The adversary includes a conflicting (double spending) transaction tx' in a block extending b in $\mathsf{C}_{\mathcal{A}}$. The adversary wants to produce a suffix proof convincing a light client that $\mathsf{C}_{\mathcal{A}}$ is the longer chain. In order to achieve this, the adversary

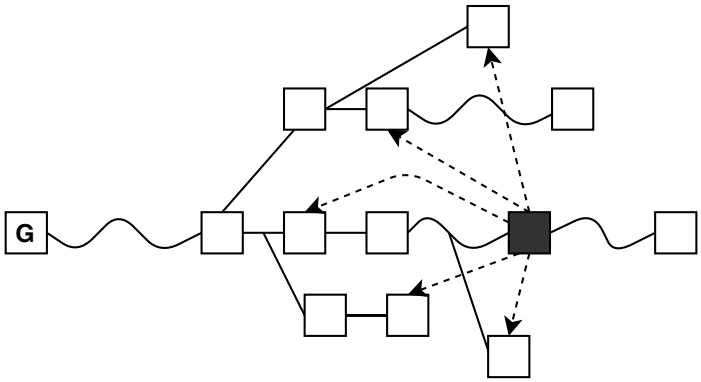

Figure 4.3: A thorny block appended to an honest party's chain. The dashed arrows are interlink pointers.

needs to include a greater amount of total proof-of-work in her suffix proof, $\pi_{\mathcal{A}}$, in comparison to that included in the honest party's proof, π_B, so as to achieve $\pi_{\mathcal{A}} \geq_m \pi_B$. Towards this purpose, she miners intermittently on both C_B and $\mathsf{C}_{\mathcal{A}}$. She produces some thorny blocks in both chains $\mathsf{C}_{\mathcal{A}}$ and C_B which will allow her to usurp selected blocks of C_B and present them to the light client as if they belonged to $\mathsf{C}_{\mathcal{A}}$ in her suffix proof.

The general form of this attack for an adversary sewing blocks to one forked chain is illustrated in Figure 4.4. Dashed arrows represent interlink pointers of some level $\mu_{\mathcal{A}}$. Starting from a thorny block in the adversary's forked chain and following the interlink pointers, jumping between $\mathsf{C}_{\mathcal{A}}$ and C_B, a chain of blocks crossing forks is formed, which the adversary claims as part of her suffix proof. Blocks of both chains are included in this proof and a verifier cannot distinguish the non-smooth pointers participating in this proof chain and, as a result, considers it a valid proof. Importantly, the adversary must ensure that any blocks usurped from the honest chain are not included in the honest NIPoPoW to force the NIPoPoW verifier to consider an earlier LCA block b; otherwise, the adversary will compete after a later fork point, negating any sewing benefits.

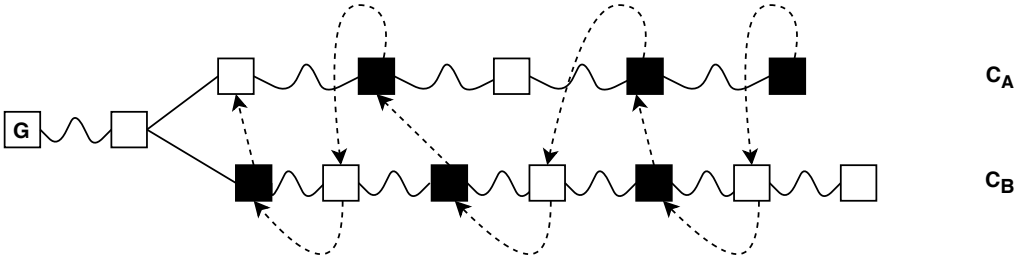

Figure 4.4: Generic Chainsewing Attack. C_B is the chain of an honest party and $\mathsf{C}_{\mathcal{A}}$ the adversary's chain. Adversarially generated blocks are colored black. Dashed arrows represent interlink pointers included in the adversary's suffix proof. Wavy lines imply one or more blocks.

This generic attack can be made concrete as follows. The adversary chooses to attack at some level $\mu_{\mathcal{A}} \in \mathbb{N}$ (ideally, if the honest verifier does not impose any succinctness limits, the adversary sets $\mu_A = 0$). As shown in Figure 4.5, she first generates a block b' in her forked chain $\mathsf{C}_{\mathcal{A}}$ containing the double spend, and a block a' in the honest chain C_B which thorny-points to b'. Block a' will be

accepted as valid in the honest chain C_B despite the invalid interlink pointers. The adversary also chooses a desired superblock level $\mu_B \in \mathbb{N}$ that she wishes the honest party to attain. Subsequently, the adversary waits for the honest party to mine and sews any blocks mined on the honest chain that are of level below μ_B. However, she must bypass blocks that she thinks the honest party will include in their final NIPoPoW, which are of level μ_B (the blue block designated c in Figure 4.5). To bypass a block, the adversary mines her own thorny block d on top of the current honest tip (which could be equal to the block to be bypassed, or have progressed further), containing a thorny pointer to the block preceding the block to be bypassed and hoping that it will not exceed level μ_B (if it exceeds that level, she discards her d block). Once m blocks of level μ_B have been bypassed in this manner, the adversary starts bypassing blocks of level $\mu_B - 1$, because the honest NIPoPoW will start including lower-level blocks. The adversary continues descending in levels until a sufficiently low level $\min \mu_B$ has been reached at which point it becomes uneconomical for the adversary to continue bypassing blocks (typically for a 1/4 adversary, $\min \mu_B = 2$). At this point, the adversary forks off of the last sewed honest block. This last honest block will be used as the last portion of the adversarial π part of the NIPoPoW proof. She then independently mines a k-long suffix for the χ portion and creates her NIPoPoW $\pi\chi$. Lastly, she waits for enough time to pass so that the honest party's chain progresses sufficiently to make the previous bypassing guesses correct and so that no blocks in the honest NIPoPoWs coincide with blocks that have not been bypassed. This requires to wait for the following blocks to appear in the honest chain: $2m$ blocks of level μ_B; after the m^{th} first μ_B-level block, a further $2m$ blocks of level $\mu_B - 1$; after the m^{th} such block, a further $2m$ blocks of the preceding level, and so on until level 0 is reached.

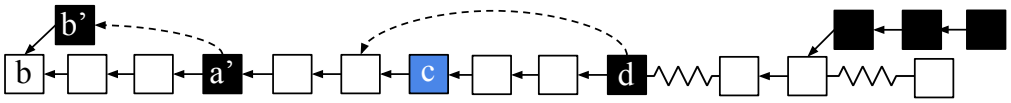

Figure 4.5: A portion of the concrete Chainsewing Attack. The adversary's blocks are shown in black, while the honestly generated blocks are shown in white. Block b' contains a double spend, while block a' sews it in place. The blue block c is a block included in the honest NIPoPoW, but it is bypassed by the adversary by introducing block d which, while part of the honest chain, points to c's parent. After a point, the adversary forks off and creates $k = 3$ of their own blocks.

In this attack the adversary uses thorny blocks to "sew" portions of the honestly adopted chain to her own forked chain. This justifies the name given to the attack. Note that in order to make this attack successful, the adversary needs only produce few superblocks, but she can arrogate an arbitrarily large number of honestly produced blocks. Thus the attack succeeds with non-negligible probability.

Chainsewing Attack Simulation

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

For performance reasons, our model for the simulation slightly deviates from the Backbone model on which the theoretical analysis of Section **??** is based and instead follows the simpler model of Ren [26]. This model favours the honest parties, and so provides a lower bound for probability of

adversarial success, strengthening our results. Here, block arrival is modelled as a Poisson process and blocks are deemed to belong to the adversary with probability t/n, while they are deemed to belong to the honest parties with probability $(n-t)/n$. Block propagation is assumed instant and every party learns about a block as soon as it is mined. As such, the honest parties are assumed to work on one common chain and the problem of non-uniquely successful rounds does not occur.

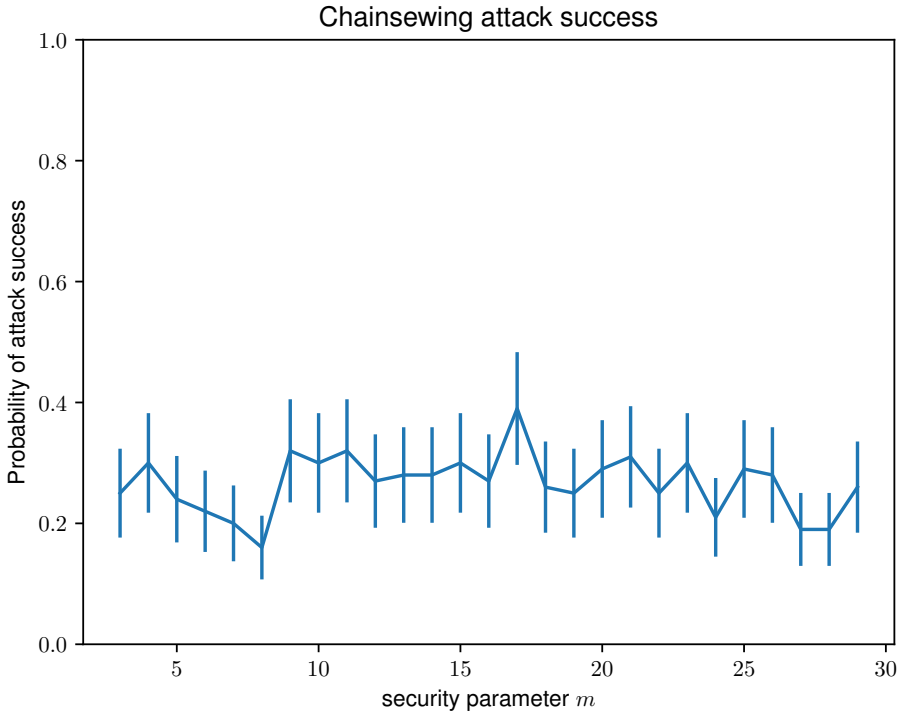

Figure 4.6: The measured probability of success of the Chainsewing attack mounted under our parameters for varying values of the security parameter m. Confidence intervals at 95%.

We consistently find a success rate of approximately 0.26 which remains more or less constant independent of the security parameter, as expected. We plot our results with 95% confidence intervals in Figure 4.6. This is in contrast with the best previously known attack in which, for all examined values of the security parameter, the probability of success remains below 1%.

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 was able to claim not only blocks of shorter forked chains, but also arbitrarily long parts of the chain generated by an honest party. Since thorny blocks are accepted as valid, the verifier cannot distinguish blocks that actually belong in a chain from blocks that only seem to belong in the same chain because they are pointed to from a thorny block.

The idea for a secure protocol is to distinguish the smooth from the thorny blocks, so that smooth blocks can never point to thorny blocks. In this way we can make sure that thorny blocks acting as passing points to fork chains, as block a' does in Figure 4.5, cannot be pointed to by honestly generated blocks. Therefore, the adversary cannot utilize honest mining power to construct a stronger suffix proof for her fork chain. Our velvet construction mandates that honest miners create blocks that contain interlink pointers pointing only to previous smooth blocks. As such, newly created smooth blocks can only point to previously created smooth blocks and not thorny

blocks. Following the terminology of Section **??**, the smoothness of a blocks in this new construction is a stricter notion than smoothness in the naïve construction.

In order to formally describe the suggested protocol patch, redefine the notion of a smooth block recursively by introducing the notion of a smooth interlink pointer.

Definition 10 (Smooth Pointer)**.** A *smooth pointer* of a block b for a specific level μ is the interlink pointer to the most recent μ-level smooth ancestor of b.

We describe a protocol patch that operates as follows. The superblock NIPoPoW protocol works as usual but each honest miner constructs smooth blocks whose interlink contains only smooth pointers; thus it is constructed excluding thorny blocks. In this way, although thorny blocks are accepted in the chain, they are not taken into consideration when updating the interlink structure for the next block to be mined. No honest block could now point to a thorny superblock that may act as 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 argument holds because thorny blocks do not form a valid chain along with honestly mined blocks anymore. Consequently, as far as the blocks included in a suffix proof are concerned, we can think of thorny blocks as belonging in the adversary's fork chain for the π part of the proof, which is the comparing part between proofs. Figure 4.7 illustrates this remark. The velvet NIPoPoW verifier is also modified to only follow interlink pointers, and never previd pointers (which could be pointing to thorny blocks, even if honestly generated).

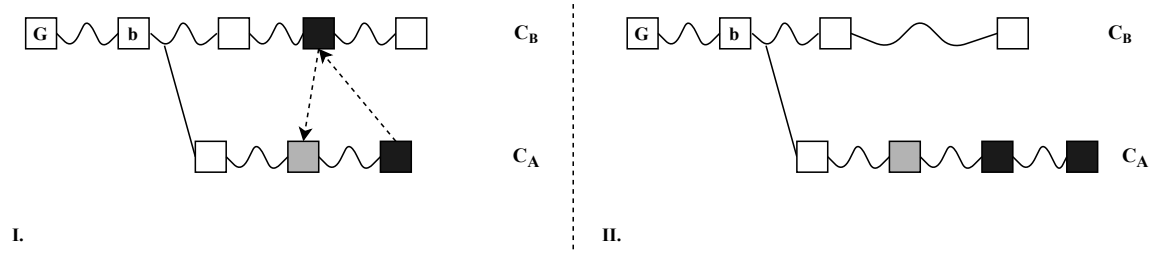

Figure 4.7: The adversarial fork chain C_A and chain C_B of an honest party. Thorny blocks are colored black. Dashed arrows represent interlink pointers. Wavy lines imply one or more blocks. After the protocol update, when an adversarially generated block is sewed from C_B into the adversary's suffix proof the verifier 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.

With this protocol patch we conclude that the adversary cannot usurp honest mining power to her fork chain. This comes with the cost that the honest prover cannot utilize thorny blocks despite belonging in the honest chain. Due to this fact we define the Velvet Honest Majority Assumption for (1/4)-bounded adversary under which the security of the protocol is guaranteed.

Definition 11 (Velvet Honest Majority)**.** Let n_h be the number of upgraded honest miners. Then t out of total n parties are corrupted such that $\dfrac{t}{n_h} < \dfrac{1-\delta_v}{3}$.

The number of upgraded honest miners can be calculated via the "velvet parameter".

Definition 8 (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 $n_h = g(n-t)$.*

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

Lemma 4. A velvet suffix proof constructed by an honest party 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 $b = \{b : b \in \mathcal{P}_{\mathcal{A}} \wedge \text{b generated at } r \geq r_t\}$ 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 4.8.

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.

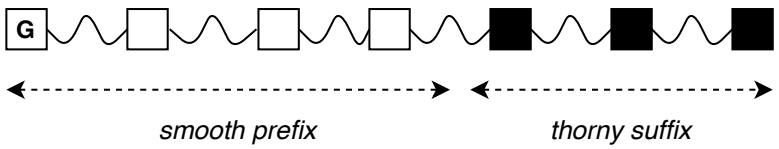

Figure 4.8: General case of the adversarial velvet suffix proof $\mathcal{P}_{\mathcal{A}} = (\pi_{\mathcal{A}}, \chi_{\mathcal{A}})$ consisting of an initial part of smooth blocks followed by thorny blocks.

We now describe the algorithms 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 11. Function *isSmoothBlock(B)* checks whether a block B is smooth by calling *isSmoothPointer(B,p)* for every pointer p in B's interlink. Function *isSmoothPointer(B,p)* returns *true* if p is a valid pointer, 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 12. It is the same as in the case of a soft fork, but works on the smooth chain C_S instead of C.

The construction of the velvet suffix prover is given in Algorithm 13. Again it deviates from the soft fork case by working on the smooth chain C_S instead of C. Lastly, the Verify algorithm for the NIPoPoW suffix protocol remains the same as in the case of hard or soft fork, keeping in mind that no *previd* links can be followed when verifying the ancestry of the chain to avoid hitting any thorny blocks.

Algorithm 11 Smooth chain for suffix proofs

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

Algorithm 12 Velvet updateInterlink

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

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

During our study on the problem we failed to prove the security of a protocol construction under the hypothesis $(\frac{1}{2})$-bounded adversary, though we tried for a number of them. 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

Algorithm 13 Velvet Suffix Prover

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

try to argue for it in this section. Our hope is that this discussion will help the reader to obtain a deeper understanding on the problem, as it resembles our own research journey.

Claim: *Assume t adversarial out of n total parties and $n_h = g(n-t)$ the number of upgraded honest parties. There is no construction for superblock NIPoPoW suffix proofs under velvet fork conditions which is both succinct and secure for every adversary, such that* $\frac{t}{n_h} < (1-\delta)$.

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 4.4. Let us consider a protocol construction p which allows for superblock NIPoPoW velvet suffix proofs and is secure for 1/2-bounded adversary. Our main goals are p to be both secure and succinct. We are even willing to 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, if this would be of needed. However, note that in such a case the same power to challenge any submitted proof is also provided to the adversary.

Now assume an adversary of $\frac{1-\delta}{2} < \frac{t}{n_h} < 1-\delta$, meaning that the adversarial parties count more than 1/3 but less than the 1/2 of the upgraded honest parties. Then, by following a selfish mining strategy [9][7] it is possible for the adversary to maintain more than 50% of the blocks in the longest valid chain. In particular, consider that the adversary mines selfish, meaning that for every block that she mines, she does not immediately announce it to the network. Instead, she waits until an honestly generated block is announced and at this time she announces her block too. In the worst case, consider that the adversary 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 as a temporary fork. Because of this attack the block distribution in the chain between adversarial and honest parties can be as illustrated in Figure 4.9 in the worst case scenario. For (1/3) adversary half of the blocks in the chain are expected to be adversarial, while for (1/2) adversary no honest blocks are included in the chain.

Now observe that in the chainsewing attack the *prevId* block property between two sequential blocks is violated while constructing an attacking proof. Therefore, an initial naive approach would be to resort to the *prevId* pointers which have been validated with the proof-of-work, in order to contest and unveil an invalid proof chain. 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 data 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, or else we have no chance in keeping our protocol construction efficient.

Now assume, to honest parties' favor, that given a block b we have the means to efficiently inspect all the blocks in the intelrinked chain where b belongs. This could be implemented by adding an appropriate Merkle-Mountain-Range root in each block header. Now one should consider the following choices regarding the next step in the protocol construction:

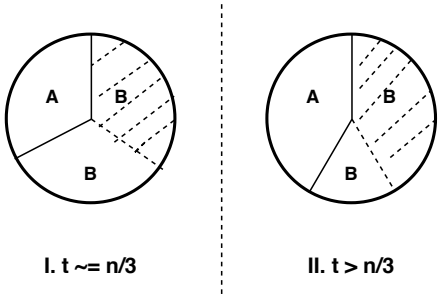

Figure 4.9: Pie chart of adversarially and honestly generated blocks appended in the chain during a round set S. Part **A** stands for blocks mined by the adversary while **B** for blocks mined by honest players. Lined out parts denote honestly mined blocks that were defeated by adversarially mined ones 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.

1. either require proofs that additionally contain consistency sub-proofs as for the interlinked chain history of each block included or

2. permit inconsistencies in the interlinked chain history of the included blocks in order to avoid excluding adversarially generated blocks from honest proofs, with the hope that an honest prover will be able to unveil a submitted proof that utilizes invalid interlink pointers by using the history commited in the MMRs.

Observe that neither of these approaches can provide a secure protocol.

In the first case, the requirement for consistency sub-proofs essentially compels the adversary to comply to the honest chain history, if she means to utilize honeslty generated blocks in her proof. However, observe that the adversary can even maintain a consistent interlink chain by using only her own mined blocks and these blocks may overwhelm the honestly generated ones, as shown in Figure 4.9. At the same time, the honest provers are obliged to exclude the inconsistent adversarially generated blocks from their proofs, thus being doomed to utilize only a minority of blocks resulting to non-winning proofs with non-negligible probability.

In the second case, in an attempt to overcome the shortcomings of the previous approach, no chain consistency proofs are required. The idea is that an honest prover inspecting the submitted proofs can detect an invalid proof and try to efficiently prove the broken $prevId$ property. Remember that he cannot just send the $prevId$ pointers, as this would destroy the succinctness of our protocol as discussed before. He should use the interlink chain history commintments instead. Consider that after inspecting an invalid proof the honest prover observes a block b pointing to a superblock b' of level $\mu_{b'}$, but b, b' are not connected with $prevId$ pointers. This means that b' belongs to a fork chain. In consequence, any honest block generated after b contains a commitment for the correct $\mu_{b'}$ superblock that should be pointed by the adversarial block b. Now consider that the honest prover can submit a merkle proof denoting the inconsistency in order to make the adversarial proof fail. However, this power of calling into question the consistency of a block in the proof is also a power for the adversary, who can question the consistency of honest blocks in an honest proof based on any adversarial block included in the proof. The problem is, that the verifier has no way to distinguish the chain history commitments between honest and adversarial parties. Consequently, this approach would fail to construct a secure protocol too.

We conclude that such a protocol p, both succinct and secure, could not exist for (1/2)-bounded adversary.

4.4 Analysis of Velvet Protocol Patch

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

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

More in detail, consider the adversary who wishes to attack a specific block level μ_B and generates a NIPoPoW proof containing a block b of a fork chain which contains a double spending transaction. Then she acts as follows. She mines on her fork chain $\mathsf{C}_{\mathcal{A}}$ but when she observes a μ_B-level block in C_B she tries to mine a thorny block on C_B in order to suppress this μ_B block. This thorny block contains an interlink pointer which jumps onto her fork chain, but a previd pointer to the honest chain. If the suppression succeeds she has managed to damage the distribution of μ_B-superblocks within the honest chain, at the same time, to mine a block that she can afterwards use in her proof. If the suppression does not succeed she can still use the thorny block in her proof. The above are illustrated in Figure 4.10.

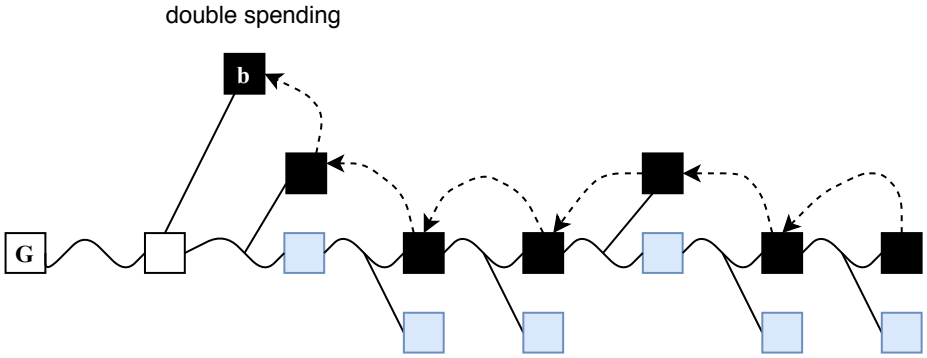

Figure 4.10: The adversary suppresses honestly generated blocks and chainsews thorny blocks in C_B. Blue blocks are honestly generated blocks of some level of attack. The adversary tries to suppress them. If the suppression is not successful, the adversary can still use the block she mined in her proof.

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

For the analysis, we use the techniques developed in the Backbone line of work [9]. Towards that end, we follow their definitions and call a round *successful* if at least one honest party made a successful random oracle query during the round, i.e., a query b such that $H(b) \leq T$. A round in which exactly one honest party made a successful query is called *uniquely successful* (the adversary could have also made successful queries during a uniquely successful round). Let $X_r \in \{0,1\}$ and $Y_r \in \{0,1\}$ denote the indicator random variables signifying that r was a successful or uniquely successful round respectively, and let $Z_r \in \mathbb{N}$ be the random variable counting the number of successful queries of the adversary during round r. For a set of consecutive rounds U, we define

$Y(U) = \sum_{r \in U} Y_r$ and similarly define X and Z. We denote $f = \mathbb{E}[X_r] < 0.3$ the probability that a round is successful.

Let λ denote the security parameter (the output size κ of the random oracle is taken to be some polynomial of λ). We make use of the following known [9] results. It holds that $pq(n-t) < \frac{f}{1-f}$. For the Common Prefix parameter, it holds that $k \geq 2\lambda f$. Additionally, for any set of consecutive rounds U, it holds that $\mathbb{E}[Z(U)] < \frac{t}{n-t} \cdot \frac{f}{1-f}|U|$, $\mathbb{E}[X(U)] < pq(n-t)|U|$, $\mathbb{E}[Y(U)] > f(1-f)|U|$. An execution is called *typical* if the random variables X, Y, Z do not deviate significantly (more than some error term $\epsilon < 0.3$) from their expectations. It is known that executions are typical with overwhelming probability in λ. Typicality ensures that for any set of consecutive rounds U with $|U| > \lambda$ it holds that $Z(U) < \mathbb{E}[Z(U)] - \epsilon\,\mathbb{E}[X(U)]$ and $Y(U) > (1-\epsilon)\,\mathbb{E}[Y(U)]$. From the above we can conclude to $Y(U) > (1-\epsilon)f(1-f)|U|$ and $Z(U) < \frac{t}{n-t} \cdot \frac{f}{1-f}|U| + \epsilon f|U|$ which will be used in our proofs. We consider $f < \frac{1}{20}$ a typical bound for parameter f. This is because in our (1/4)-bounded adversary assumption we need to reach about 75% of the network, which requires about 20 seconds [4]. Considering also that in Bitcoin the block generation time is in expectation 600 seconds, we conclude to an estimate $f = \frac{18}{600}$ or $f = 0.03$.

The following definition and lemma are known [33] results and will allow us to argue that some smooth superblocks will survive in all honestly adopted chains. With foresight, we remark that we will take Q to be the property of a block being both smooth and having attained some superblock level $\mu \in \mathbb{N}$.

Definition 9 (Q-block)**.** A *block property* is a predicate Q defined on a hash output $h \in \{0,1\}^\kappa$. Given a block property Q, a valid block with hash h is called a Q-block if $Q(h)$ holds.

Lemma 7 (Unsuppressibility)**.** Consider a collection of polynomially many block properties $\mathcal{Q}$. In a typical execution every set of consecutive rounds U has a subset S of uniquely successful rounds such that

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

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

Lemma 8. Consider Algorithm 12 under velvet fork with parameter g and (1/4)-bounded velvet honest majority. Let U be a set of consecutive rounds $r_1 \cdots r_2$ and C the chain of an honest party at round r_2 of a typical execution. Let $\mathsf{C}_U^S = \{b \in \mathsf{C} : \text{b is smooth} \wedge \text{b was generated during U}\}$. Let $\mu, \mu' \in \mathbb{N}$. Let C' be a μ' superchain containing only adversarial blocks generated during U and suppose $|\mathsf{C}_U^S{\uparrow^\mu}| > k$. Then for any $\delta_3 \leq \frac{3\lambda f}{5}$ it holds that $2^{\mu'}|\mathsf{C}'| < 2^\mu(|\mathsf{C}_U^S{\uparrow^\mu}| + \delta_3)$.

Proof. From the Unsuppressibility Lemma we have that there is a set of uniquely successful rounds $S \subseteq U$, such that $|S| \geq Y(U) - 2Z(U) - \delta'$, where $\delta' = 2\lambda f(\frac{t}{n-t} \cdot \frac{1}{1-f} + \epsilon)$. We also know that Q-blocks generated during S are distributed as in an unsuppressed chain. Therefore considering the property Q for blocks of level μ that contain smooth interlinks we have that $|\mathsf{C}_U^S{\uparrow^\mu}| \geq (1-\epsilon)g2^{-\mu}|S|$. We also know that for the total number of μ'-blocks the adversary generated during U that $|\mathsf{C}'| \leq (1+\epsilon)2^{-\mu'}Z(U)$. Then we have to show that $(1-\epsilon)g(Y(U) - 2Z(U) - \delta') > (1+\epsilon)Z(U)$ or $((1+\epsilon) + 2g(1-\epsilon))Z(U) < g(1-\epsilon)(Y(U) + \delta')$. But it holds that $(1+\epsilon) + 2g(1-\epsilon) < 3$, therefore it suffices to show that

$$3Z(U) < g(1-\epsilon)(Y(U) + \delta') - 2^\mu\delta_3 \tag{4.1}$$

Substituting in Equation **??** the bounds of X, Y, Z discussed above, it suffices to show that

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

or

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

or

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

or

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

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

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

or

$$\frac{t}{n-t} < \frac{g}{3} \cdot \frac{(1-\epsilon)^2 f(1-f) - \frac{(1-\epsilon)\delta'}{|U|} - \frac{2^{\mu}\delta_3}{|U|}}{\frac{f}{1-f}} - \epsilon' \tag{4.2}$$

In order to show Equation 4.2 we use $f \leq \frac{1}{20}$ which is a typical bound for our setting as discussed above. Because all blocks in C were generated during U and $|\mathsf{C}| > k$, $|U|$ follows negative binomial distribution with probability $2^{-\mu}pq(n-t)$ and number of successes k. Applying a Chernoff bound we have that $|U| > (1-\epsilon)\frac{k}{2^{-\mu}pq(n-t)}$. Using the inequalities $k \geq 2\lambda f$ and $pq(n-t) < \frac{f}{1-f}$, we deduce that $|U| > (1-\epsilon)2^{\mu}2\lambda(1-f)$. So we have that

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

or

$$\frac{\delta'}{|U|} < \frac{t}{n-t} \cdot \frac{f}{(1-\epsilon)(1-f)^2} + \epsilon < 0.01 + \epsilon$$

We also know that $\delta_3 \leq \frac{3\lambda f}{5}$, so

$$\frac{2^{\mu}\delta_3}{|U|} < \frac{2^{\mu}\frac{3\lambda f}{5}}{2^{\mu}2\lambda(1-f)}$$

or

$$\frac{2^{\mu}\delta_3}{|U|} < \frac{3f}{10(1-f)} < 0.01 + \epsilon$$

By substituting the above and the typical f parameter bound in Equation (4.2) we conclude that it suffices to show that $\frac{t}{n-t} < \frac{1-\epsilon''}{3}g$ which is equivalent to $\frac{t}{n-t} < \frac{1-\delta_v}{3}g$ for $\epsilon'' = \delta_v$, which is the (1/4) velvet honest majority assumption, so the claim is proven. □

Lemma 9. Consider Algorithm 12 under velvet fork with parameter g and (1/4)-bounded velvet honest majority. Consider the property Q for blocks of level μ. Let U be a set of consecutive rounds and C the chain of an honest party at the end of U of a typical execution and $\mathsf{C}_U = \{b \in \mathsf{C} :$ b was generated during U$\}$. Suppose that no block in C_U is of level μ.
Then $|U| \leq \delta_1$ where $\delta_1 = \dfrac{(2+\epsilon)2^\mu + \delta'}{(1-\epsilon)f(1-f) - 2\frac{t}{n-t}\frac{f}{1-f} - 3\epsilon f}$.

Proof. The statement results immediately form the Unsuppressibility Lemma. Suppose for contradiciton that $|U| > \delta_1$. Then from the Unsuppressibility Lemma we have that there is a subset of consecutive rounds S of U for which it holds that $|S| \geq Y(U) - 2Z(U) - \delta'$ where $\delta' = 2\lambda f(\frac{t}{n-t} \cdot \frac{1}{1-f} + \epsilon)$. By substituting $Y(U) > (1-\epsilon)f(1-f)|U|$ and $Z(U) < \frac{t}{n-t}\frac{f}{1-f} + \epsilon f|U|$ we have that $|S| > (2+\epsilon)2^\mu$ but Q-blocks generated during S follow the distribution as in a chain where no suppression attacks occur. Therefore at least one block of level μ would appear in C_U, thus we have reached a contradiction and the statement is proven. □

Theorem 2 (Suffix Proofs Security under velvet fork)**.** *Assuming honest majority under velvet fork conditions (8) such that* $t \leq (1-\delta_v)\frac{n_h}{3}$ *where* n_h *the number of upgraded honest parties, the Non-Interactive Proofs of Proof-of-Work construction for computable k-stable monotonic suffix-sensitive predicates under velvet fork conditions in a typical execution is secure.*

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

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

Suppose for contradiction that the verifier outputs $\neg Q(\mathsf{C}_B)$. Thus it is necessary that $\pi_\mathcal{A} \geq_m \pi_B$. We show that $\pi_\mathcal{A} \geq_m \pi_B$ is a negligible event. Let the levels of comparison decided by the verifier be $\mu_\mathcal{A}$ and μ_B respectively. Let $b_0 = LCA(\pi_\mathcal{A}, \pi_B)$. Call $\alpha_\mathcal{A} = \pi_\mathcal{A}{\uparrow}^{\mu_\mathcal{A}}\{b_0:\}$, $\alpha_B = \pi_B{\uparrow}^{\mu_B}\{b_0:\}$.

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

Our proof construction is based on the following intuition: we consider that $\alpha_\mathcal{A}$ consists of three distinct parts $\alpha_\mathcal{A}^1, \alpha_\mathcal{A}^2, \alpha_\mathcal{A}^3$ with the following properties. Block $b_0 = LCA(\pi_\mathcal{A}, \pi_B)$ is the fork point between $\pi_\mathcal{A}{\uparrow}^{\mu_\mathcal{A}}, \pi_B{\uparrow}^{\mu_B}$. Let block $b_1 = LCA(\alpha_\mathcal{A}^\mathcal{S}, \mathsf{C}_B^S)$ be the fork point between $\pi_\mathcal{A}^\mathcal{S}{\uparrow}^{\mu_\mathcal{A}}, \mathsf{C}_B$ as an honest prover could observe. Part $\alpha_\mathcal{A}^1$ contains the blocks between b_0 exclusive and b_1 inclusive generated during the set of consecutive rounds $\mathcal{S}_1$ and $|\alpha_\mathcal{A}^1| = k_1$. Consider b_2 the last block in $\alpha_\mathcal{A}$ generated by an honest party. Part $\alpha_\mathcal{A}^2$ contains the blocks between b_1 exclusive and b_2 inclusive generated during the set of consecutive rounds $\mathcal{S}_2$ and $|\alpha_\mathcal{A}^2| = k_2$. Consider b_3 the next block of b_2 in $\alpha_\mathcal{A}$. Then $\alpha_\mathcal{A}^3 = \alpha_\mathcal{A}[b_3:]$ and $|\alpha_\mathcal{A}^3| = k_3$ consisting of adversarial blocks generated during the set of consecutive rounds $\mathcal{S}_3$. Therefore $|\alpha_\mathcal{A}| = k_1 + k_2 + k_3$ and we will show that $|\alpha_\mathcal{A}| < |\alpha_B|$.

The above are illustrated, among other, in Parts I, II of Figure 4.11.

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

Claim 1: $\alpha_\mathcal{A}^1 = (\alpha_\mathcal{A}\{b_0 : b_1\} \cup b_1)$ *contains only a few blocks.* Let $|\alpha_\mathcal{A}^1| = k_1$. We have defined the blocks $b_0 = LCA(\pi_\mathcal{A}, \pi_B)$ and $b_1 = LCA(\alpha_\mathcal{A}^\mathcal{S}, \mathsf{C}_B^S)$. First observe that because of the Corollary

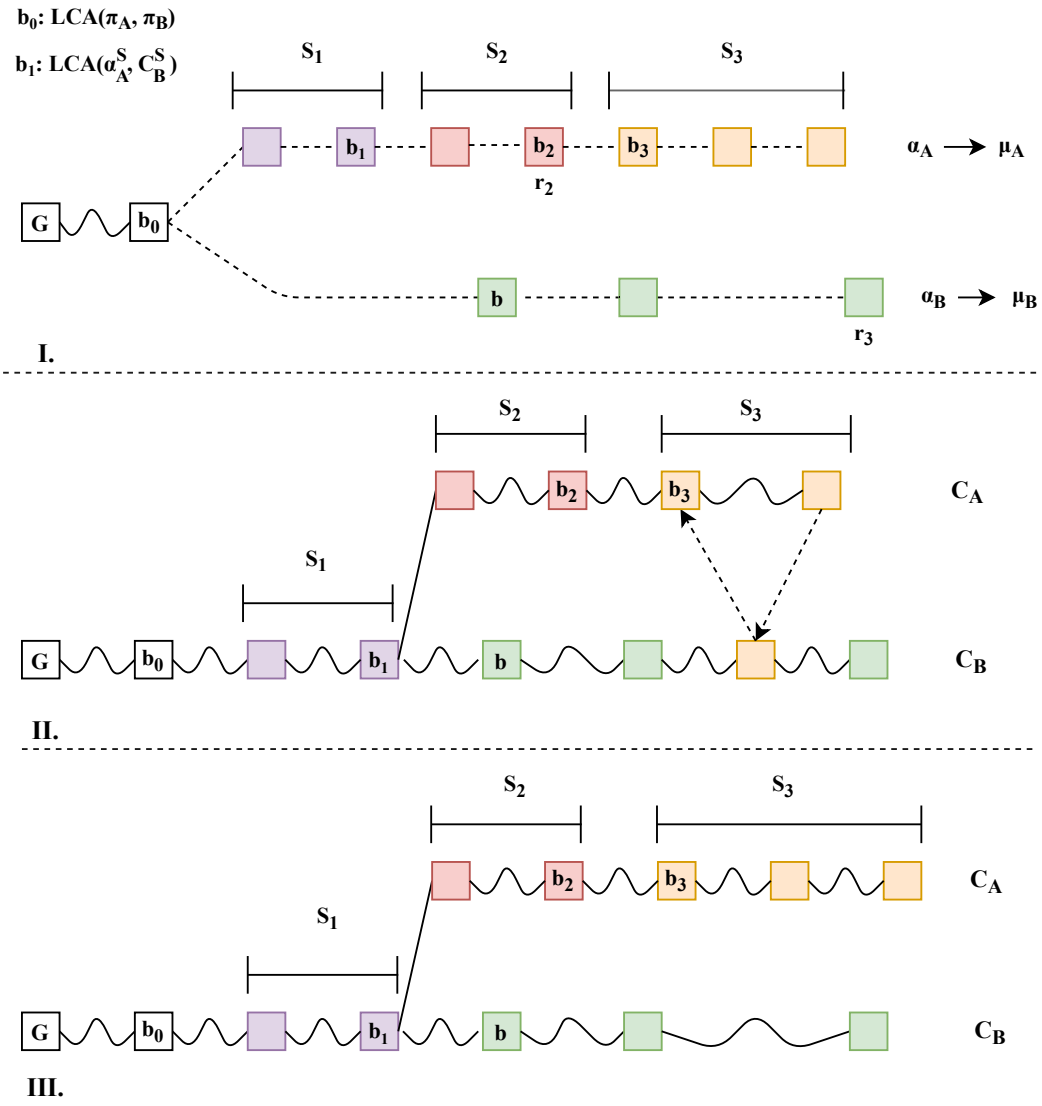

Figure 4.11: I. The three round sets in two competing proofs at different levels, II. the corresponding 0-level blocks implied by the two proofs, III: blocks in C_B and block set $\tilde{\mathsf{C}}_{\mathcal{A}}$ from the verifier's perspective.

1 there are no thorny blocks in $\alpha^1_{\mathcal{A}}$ since $\alpha^1_{\mathcal{A}}[-1] = b_1$ is a smooth block. This means that if b_1 was generated at round r_{b_1} and $\alpha^{\mathcal{S}}_{\mathcal{A}}[-1]$ in round r then $r \geq r_{b_1}$. Therefore, $\alpha^1_{\mathcal{A}}$ contains smooth blocks of C_B. We show the claim by considering the two possible cases for the relation of $\mu_{\mathcal{A}}, \mu_B$.

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

Claim 1b: If $\mu_B > \mu_{\mathcal{A}}$ then $k_1 \leq \dfrac{\delta_1 2^{-\mu_{\mathcal{A}}}}{(1+\epsilon)\dfrac{t}{n-t}\dfrac{f}{1-f}}$. In order to show this we consider block b the first block in α_B. Now suppose for contradiction that $k_1 > \dfrac{\delta_1 2^{-\mu_{\mathcal{A}}}}{(1+\epsilon)\dfrac{t}{n-t}\dfrac{f}{1-f}}$. Then from lemma 9 we have that block b is generated during S_1. But b is of lower level $\mu_{\mathcal{A}}$ and $\alpha^1_{\mathcal{A}}$ contains

smooth blocks of C_B. Therefore b is also included in $\alpha^1_{\mathcal{A}}$, which contradicts the minimality of block b_0.

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

Claim 2. *Part $\alpha^2_{\mathcal{A}} = (\alpha_{\mathcal{A}}\{b_1 : b_2\} \cup b_2)$ consists of only a few blocks.* Let $|\alpha^2_{\mathcal{A}}| = k_2$. We have defined $b_2 = \alpha^2_{\mathcal{A}}[-1]$ to be the last block generated by an honest party in $\alpha_{\mathcal{A}}$. Consequently no thorny block exists in $\alpha^2_{\mathcal{A}}$, so all blocks in this part belong in a proper zero-level chain $\mathsf{C}^2_{\mathcal{A}}$. Let r_{b_1} be the round at which b_1 was generated. Since b_1 is the last block in $\alpha_{\mathcal{A}}$ which belongs in C_B, then $\mathsf{C}^2_{\mathcal{A}}$ is a fork chain to C_B at some block b' generated at round $r' \geq r_{b_1}$. Let r_2 be the round when b_2 was generated by an honest party. Because an honest party has adopted chain C_B at a later round r_3 when the proof π_B is constructed and because of the Common Prefix property on parameter k_2, we conclude that $k_2 \leq 2^{-\mu_{\mathcal{A}}}k$.

Claim 3. *The adversary may submit a suffix proof such that $|\alpha_{\mathcal{A}}| \geq |\alpha_B|$ with negligible probability.* Let $|\alpha^3_{\mathcal{A}}| = k_3$. As explained earlier part $\alpha^3_{\mathcal{A}}$ consists only of adversarially generated blocks. Let S_3 be the set of consecutive rounds $r_2...r_3$. Then all k_3 blocks of this part of the proof are generated during S_3. Let α^3_B be the last part of the honest proof containing the interlinked μ_B superblocks generated during S_3. Then by applying Lemma 8 $\frac{m}{k}$ times we have that $2^{\mu_{\mathcal{A}}}|\alpha^3_{\mathcal{A}}| < 2^{\mu_B}(|\alpha_B^{S_3}\uparrow^{\mu_B}| + \frac{m\delta_3}{k})$. By substituting the values from all the above Claims and because every block of level μ_B in $a\alpha_B$ is of equal hashing power to $2^{\mu_B - \mu_{\mathcal{A}}}$ blocks of level $\mu_{\mathcal{A}}$ in the adversary's proof we have that: $2^{\mu_B}|\alpha^3_B| - 2^{\mu_{\mathcal{A}}}|\alpha^3_{\mathcal{A}}| > 2^{\mu_{\mathcal{A}}}(k_1 + k_2)$ or $2^{\mu_B}|\alpha^3_B| > 2^{\mu_{\mathcal{A}}}|\alpha^1_{\mathcal{A}} + \alpha^2_{\mathcal{A}} + \alpha^3_{\mathcal{A}}|$ or $2^{\mu_B}|\alpha_B| > 2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}|$ Therefore we have proven that $2^{\mu_B}|\pi_B\uparrow^{\mu_B}| > 2^{\mu_{\mathcal{A}}}|\pi^{\mu_{\mathcal{A}}}_{\mathcal{A}}|$. □

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 naively 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 4.12.

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

In order to construct secure infix proofs under velvet fork conditions, we suggest the following additional protocol patch: each upgraded miner constructs and updates an authenticated data structure for all the blocks in the chain. We suggest Merkle Mountain Ranges *(MMR)* for this structure. Now a velvet block's header additionally includes the root of this MMR.

After this additional protocol change the notion of a smooth block changes as well. Smooth blocks are now considered the blocks that contain truthful interlinks and valid MMR root too. A valid MMR root denotes the MMR that contains all the blocks in the chain of an honest full node.

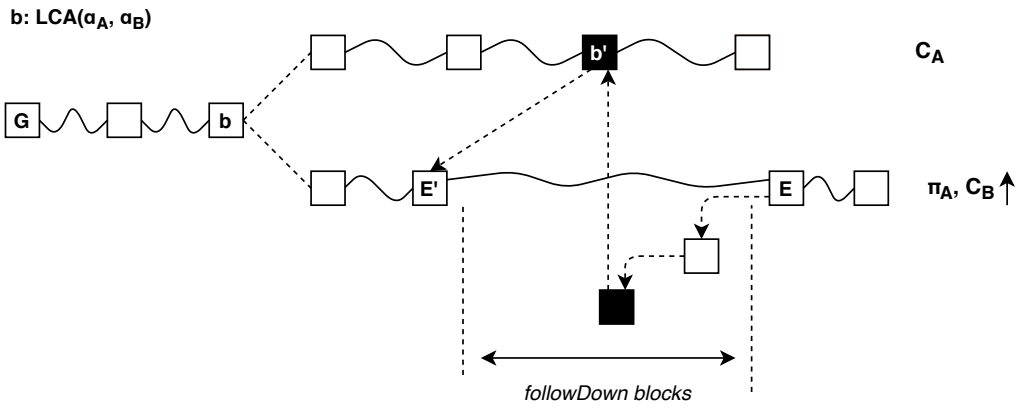

Figure 4.12: Adversarial fork chain C_A and an adversarial infix proof based on the chain adopted by an honest player. Wavy lines imply one or more blocks. Blocks generated by the adversary are colored black. Dashed arrows represent interlink pointers included in the proof as part of the *followDown* procedure. The adversary provides infix proof for a transaction in block b'.

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

The velvet infix prover and verifier

Considering this addtional patch we can now define the final algorithms for the honest miner, infix and suffix prover, as well as for the infix verifier. Because of the new notion of smooth block, the function *isSmoothBlock()* of Algorithm 11 needs to be updated, so that the validity of the included MMR root is also checked. The updated function is given in Algorithm 14. Considering that input C_S is computed using Algorithm 11 with the updated *isSmoothBlock'()* function, *Velvet updateInterlink* and *Velvet Suffix Prover* algorithms remain the same as described in Algorithms 12, 13 repsectively. The velvet infix prover given in Algorithms 15, 16 respectively. 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 that the specific predicate can be answered by a known algorithm given the block of interest. Given that the verifier is already synchronized to the longest valid chain, the infix verification algorithm only has to confirm the Merkle-Tree inclusion proof $\pi_{b'}$ for the block of interest b'.

Details about the construction and verification of an MMR and the respective inclusion proofs can be found in [21]. Note that equivalent solution could be formed by using any authenticated data structure that provides inclusion proofs of size logarithmic to the length of the chain. We suggest MMRs because of they come with efficient update operations.

Algorithm 14 Function isSmoothBlock'() for infix proof support

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

Algorithm 15 Velvet Infix Prover

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

Algorithm 16 Velvet Infix Verifier

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

Chapter 5

FlyClient under Velvet Fork

In the FlyClient paper [1] a velvet fork is suggested for the deployment of the protocol as-is, followed by a short argument for its respective security. In this document we describe an explicit attack against the FlyClient protocol under velvet fork deployment. This is essentially a kind of "Chainsewing Attack", a class of attacks that we have already described in our work on NIPoPoWs under velvet fork conditions.

5.1 The FlyClient Protocol

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

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

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

A high level description of the FlyClient is as follows. Suppose that the verifier, a superlight client, asks to synchronize to the current longest valid chain. Suppose that he receives different proofs from two provers. Each prover sends (the header) of the last block in the chain, $\mathsf{C}[-1]$, and a claim for the number of blocks in his chain, $|\mathsf{C}|$. If both proofs are valid, then the one claiming the greater block count is selected. The validity check of a proof goes as follows. The verifier has received $\mathsf{C}[-1]$, $|\mathsf{C}|$ and queries k random block headers from each prover based on a specific probabilistic sampling algorithm. For each queried block B_i the prover sends the header of B_i along with an MMR subtree inclusion proof Π_{B_i} that B_i is the i_{th} block in the chain. The verifier also checks that B_i is normally mined on the same chain as $\mathsf{C}[-1]$ by verifying that the root included in B_i is the MMR root of the first $(|\mathsf{C}| - 1)$ blocks' subtree. If the k random sampled blocks successfully pass through this verification procedure then the proof is considered valid, otherwise the proof is rejected by the verifier. The chain synchronization protocol is given in Algorithm 17.

For a single block query the protocol can be described as follows. The verifier is synchronized to a chain C and already has the tip of the chain $\mathsf{C}[-1]$. He then queries the prover and receives the header of the specific block of interest B in C and the inclusion proof $\Pi_{B \in \mathsf{C}}$. Then the verifier checks the validity of B in the same way as already described for the random sampled blocks in the synchronization protocol. The prover/verifier single query protocol is given in Algorithm 18.

Algorithm 17 FlyClient protocol [1]

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

1. The provers send to the verifier the last block header in their chain. Each header includes the root of an MMR created over the first n blocks of the corresponding chain.
2. The verifier queries k random block headers from each prover based on the described optimal probabilistic sampling algorithm.
3. For each queried block, B_i, te prover sends the header of B_i along with an MMR proof $\Pi_{B_i \in \mathsf{C}}$ that B_i is the $i-$th block in the chain.
4. The client performs the following checks for each block B_i according to Algorithm and rejects the proof if any checks fail
5. The client rejects the proof if any checks fail
6. Otherwise, the client accepts C as the valid chain

Algorithm 18 Prover/Verifier protocol for a single query [1]

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

Verifier

1. Has the root of the MMR of n blocks stored in the $n+1$ block's header
2. Queries prover for the header of block k and for $\Pi_{k \in n}$
3. Verifies that the hashes of $\Pi_{k \in n}$ hash up to the root of MMR_n
4. Calculates the root of the MMR of the $k-1$ blocks from $\Pi_{k \in n}$
5. Compares the calculated root with the root in the header of block k
6. If everything checks out, accepts the block proof

Prover

1. Has chain of $n+1$ blocks and the MMR of the first n blocks
2. Receives query for block k from verifier
3. Calculates $\Pi_{k \in n}$ from MMR_n
4. Sends header of k and $\Pi_{k \in n}$ to verifier

5.2 Velvet MMRs

A velvet fork suggests that any protocol changes are deployed in a backwards-compatible manner so that unupgraded players accept upgraded blocks and upgraded players accept unupgraded blocks too. In practice, the protocol changes are applied via some auxiliary data included in each block, which make sense and are used only by upgraded parties,while being omitted as comments by unupgraded parties.

In the context of FlyClient, velvet fork deployment implies that upgraded miners additionally

include an MMR root in each block's header. The claim made in the paper is that considering a constant fraction α of upgraded blocks in the chain, an honest prover could produce proofs by utilizing only these blocks and by joining the intermediary blocks together. This should result to less efficient proofs, bacause in order to random sample a sufficient number of upgraded blocks you need a larger underlying chain than in a hard or soft fork, since only a portion of the blocks are upgraded. The claim is that the velvet proofs remain secure. We show that this claim does no hold by presenting a specific attack.

Velvet fork requires any block to be accepted in the chain regardless the validity of the auxiliary data coming with the protocol update. In the case of FlyClient, an adversary may produce blocks which are compatible to the basic consensus rules but contain invalid MMR information. As an example, an invalid MMR may omit blocks existing in C or contain blocks which belong in temporary forks of C. We call adversarially generated blocks containing invalid MMRs *thorny* blocks. A specific case of thorny is illustrated in Figure 5.1.

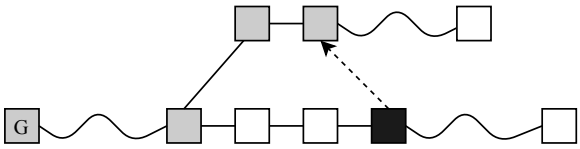

Figure 5.1: A thorny block colored black containing invalid MMR commitment to a block of a fork chain illustrated as a dashed arrow. With respect to the MMR commitments the black block along with the grey ones form a chain.

5.3 The Attack

The velvet FlyClient description does not deal with thorny blocks, meaning blocks that contain only seemingly valid auxiliary data. More specifically, it remains unspecified whether blocks containing an MMR root but not the correct one are considered valid upgraded blocks or unpupgraded blocks. We work on the hypothesis that honest miners validate the MMR root of the blocks and blocks containing invalid MMR roots are treated as unupgraded. This seems to be the only reasonable option. In the opposite case any block containing trash data in the place where the MMR root should be would completely destroy the protocol making it impossible to deliver a valid proof. We will now describe the chainsewing attack against the velvet FlyClient protocol.

Consider that the adversary utilizes more than one thorny blocks in order to cut-and-paste portions from the chain adopted by honest parties to his fork chain. Consider the attack illustrated in Figure 5.2. The adversary acts as follows. She first mines upgraded blocks on a fork chain C_A until she generates block b' containing a double spending attack. Afterwards she mines block a' in the honest chain C_B, which includes an MMR root for her fork chain, thus including the blocks from genesis and up to b'. After that she keeps mining blocks on C_B, which contain MMR root that builds on top of the root included in a', including only the following adversarially generated blocks in C_B and ignoring any intermediary honestly generated blocks while constructing the MMRs of her blocks. Additionally, during this period when she mines on C_B she tries to suppress any honest upgraded block in C_B. Towards this end she acts as follows. She regularly mines block on C_B as described ignoring honestly unupgraded blocks. When an honest upgraded block $\mathsf{C}[i]$ is appended she mines on top of block $\mathsf{C}[i-1]$. If she mines a block and the suppression fails she can still use her fresh block in her proof by continuing to construct consistent MMRs in the following blocks as described before. Figure 5.3 illustrates an example of the underlying suppression attack. From the verifier's perspective the ignored honest blocks are simply perceived as unupgraded blocks. At some later point, the adversary generates block a in the fork chain, which also contains an MMR root for all the grey and black blocks up to block b. Right afterwards the adversary produces a proof as described in the velvet FlyClient protocol giving block a as $\mathsf{C}[1]$ and the count number of all the

grey and black blocks. From the verifier's perspective the black and grey colored blocks form a valid chain, since the head of the chain a contains consistent MMR commitments with all these blocks. In addition, the random sampling performed by the FlyClient protocol will succeed because there are no invalid blocks in this chain.

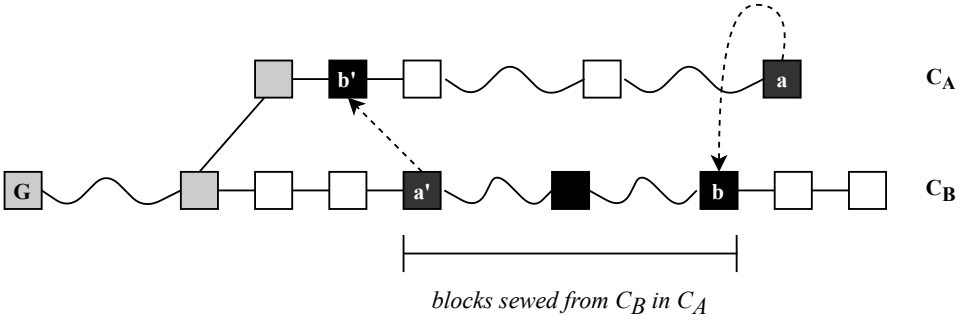

Figure 5.2: Chainsewing attack. Two thorny blocks a, a' are used to chainsew a portion of honest chain C_B to adversarial fork C_A. Black blocks imply adversarially generated blocks. Grey blocks are used in the adversarial proof along with the black ones. Wavy lines imply one or more blocks. Dashed arrows imply an MMR commitment for the destination block in the block of origin.

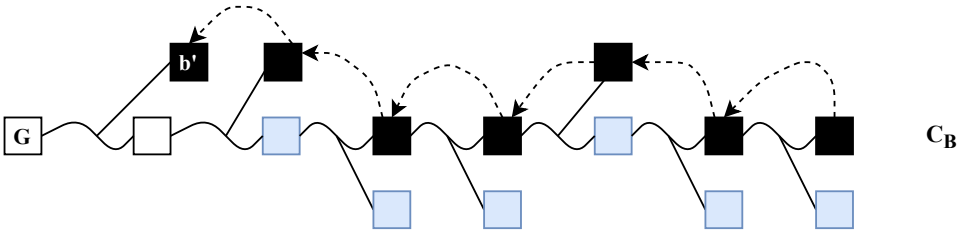

Figure 5.3: Chainsewing along with suppression attack. Black blocks imply adversarially generated blocks. Blue blocks imply honest upgraded blocks,which the adversary tries to suppress. Wavy lines imply one ore more blocks. Dashed arrows imply an MMR commitmens.

List of Figures

Bibliography

[1] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-light clients for cryptocurrencies. 2020.

[2] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.

[3] Alexander Chepurnoy, Charalampos Papamanthou, and Yupeng Zhang. Edrax: A cryptocurrency with stateless transaction validation. *IACR Cryptology ePrint Archive*, 2018:968, 2018.

[4] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, 2013.

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

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

[7] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[8] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[9] 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.

[10] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security Symposium*, pages 129–144, 2015.

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

[12] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Compact storage of superblocks for nipopow applications. In *The 1st International Conference on Mathematical Research for Blockchain Economy*. Springer Nature, 2019.

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

[14] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Smart contract derivatives, 2020.

[15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[16] Aggelos Kiayias, Peter Gaži, and Dionysis Zindros. Proof-of-stake sidechains. In *IEEE Symposium on Security and Privacy*. IEEE, IEEE, 2019.

[17] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In *International Conference on Financial Cryptography and Data Security*, pages 61–78. Springer, 2016.

[18] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-Interactive Proofs of Proof-of-Work. In *International Conference on Financial Cryptography and Data Security*. Springer, 2020.

[19] Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *International Conference on Financial Cryptography and Data Security*. Springer, Springer, 2019.

[20] Jae-Yun Kim, Jun-Mo Lee, Yeon-Jae Koo, Sang-Hyeon Park, and Soo-Mook Moon. Ethanos: Lightweight bootstrapping for ethereum. *arXiv preprint arXiv:1911.05953*, 2019.

[21] Ben Laurie, Adam Langley, and Emilia Kasper. Rfc6962: Certificate transparency. *Request for Comments. IETF*, 2013.

[22] Eric Lombrozo, Johnson Lau, and Pieter Wuille. BIP 0141: Segregated witness (consensus layer). Available at: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki, 2015.

[23] Izaak Meckler and Evan Shapiro. CODA: Decentralized Cryptocurrency at Scale. 2018.

[24] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.

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

[26] Ling Ren. Analysis of nakamoto consensus, 2019.

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

[28] Peter Todd. Merkle mountain ranges, October 2012. https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md.

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

[30] Karl Wüst and Arthur Gervais. Ethereum eclipse attacks. Technical report, ETH Zurich, 2016.

[31] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt. SoK: Communication across distributed ledgers, 2019.

[32] Alexei Zamyatin, Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Edgar Weippl, William Knottenbelt, and Alexei Zamyatin. A wild velvet fork appears! inclusive blockchain protocol changes in practice. In *International Conference on Financial Cryptography and Data Security*. Springer, 2018.

[33] Dionysis Zindros. *Decentralized Blockchain Interoperability*. PhD thesis, Apr 2020.