Theorem[section]
Corol-
lary[theorem]
[lemma]Lemma
Definition[section]

Non-interactive proofs of proof-of-work

Anonymous Author(s)

ABSTRACT

Blockchain protocols such as bitcoin provide decentralized consensus mechanisms based on proof-of-work (PoW). In this work we introduce and instantiate a new primitive for blockchain protocols called Noninteractive-Proofs-of-Proof-of-Work (NIPoPoWs) which can be adapted into existing proof-of-work-based cryptocurrencies. Unlike a traditional blockchain client which must verify the entire linearly-growing chain of PoWs, clients based on NIPoPoWs require resources only logarithmic in the length of the blockchain. NIPoPoWs solve two important open questions for PoW based consensus protocols. Specifically the problem of constructing efficient transaction verification clients, sometimes called "simple payment verification" or SPV, and the problem of constructing efficient sidechain proofs. We provide a formal model for NIPoPoWs. We prove our construction is secure in the backbone model and show that the communication is succinct in size. We provide simulations and experimental data to measure concrete communication efficiency and security. Finally, we provide two ways that our NIPoPoWs can be adopted by existing blockchain protocols, first via a soft fork, and second via a new update mechanism that we term a "velvet fork" and enables to harness some of the performance benefits of NIPoPoWs even with a minority upgrade.

1 INTRODUCTION

Cryptocurrencies such as Bitcoin

In this work we introduce, analyze and instantiate a new primitive, Noninteractive Proofs of Proof-of-Work (NIPoPoWs), which can be adapted into existing cryptocurrencies to support more efficient clients. A traditional blockchain client must verify the entire chain of proofs-of-work, which grows linearly over time. On the contrary, clients based on NIPoPoWs require resources only logarithmic in the length of the blockchain.

Motivation. There are three trends that motivate the need for more efficient clients, specifically clients that can verify transactions without having to process the whole blockchain. First, while Bitcoin remains by far the largest and most widely used cryptocurrency, the ecosystem has become significantly more diverse. Today there are hundreds of competitors, commonly called "altcoins", and Bitcoin's dominance in the market is at an all time low

Second, cryptocurrencies are increasingly used as components of larger systems (the cryptocurrency system is typically called a "blockchain" when its currency feature is not salient). As one example, a recent system called Catena

Finally, there has been significant interest in the development of "cross-chain" applications, i.e. logical transactions that span multiple separate cryptocurrencies. Simple cross-chain transactions are feasible today: the most well-known is the atomic exchange

These examples illustrate that our solution is a key component for two important pillars needed for next-generation blockchains: *interoperability* and *scalability*.

Our contributions. In summary, we make the following contributions in this paper.

Our main technical contribution is the introduction of a new cryptographic primitive called Non-Interactive Proofs of Proof of Work (NIPoPoW).

We present a formal model and a provably secure instantiation of NIPoPoWs in the model of

We provide concrete parameterization and empirical analysis focusing on showing the potential savings of our approach versus existing clients. Using actual data from the bitcoin blockchain, we quantify the actual savings of NIPoPoWs over the previous known techniques of constructing efficient SPV verifiers.

We describe two practical deployment paths for our NIPoPoWs that existing cryptocurrencies can adopt to support our efficient clients. Specifically, we first outline how NiPoPoW's can be deployed using either a "soft fork" or a "hard fork" ugprade procedure, both of which have been successfuly used by existing cryptocurrencies

2 RELATED WORK

Lightweight "SPV" Clients. Nakamoto's original Bitcoin whitepaper

As an alternative to downloading all block headers at first startup, the SPV client software could embed a hardcoded checkpoint (perhaps chosen conservatively, e.g. many months in the past), blocks prior to which are ignored.[1] The BitcoinJ software provides mechanisms for users to build "Checkpoint Files," and at least some applications, such as Multibit,[2] include checkpoints. Although this approach is very efficient, it introduces additional trust assumptions on software maintainers.

Proofs of proof-of-work. Kiayias et al. (KLS)

In an influential whitepaper

3 MODEL AND DEFINITIONS

We define our problem in the setting of a blockchain client which is completely stateless beyond the knowledge of a common reference string, the Genesis block. Without loss of generality, this single reference block could be any stable checkpoint block, either hard-coded in the client or obtained through a previous interaction with the network. However, importantly, the client does not maintain a blockchain. The challenge is, then, to build a client that is able to communicate with the network and, without downloading the whole chain headers, is convinced in a secure manner that a transaction

[1] See https://bitcoinj.github.io/speeding-up-chain-sync

[2] https://multibit.org/help/hd0.3/how-spv-works.html

has taken place. We term such a client a *verifier* and we term *provers* the full nodes it connects to.

The intuition behind these proofs is quite simple: Let T be the mining target for the network. Then because block ids are generated uniformly at random, half of the blocks will have an id less than $T/2$. By having the prover present to the verifier just these blocks which capture more proof-of-work than their neighbor counterparts, the verifier is convinced that the proof-of-work of the omitted blocks also took place. The same logic can be applied to any level μ, namely blocks with id less than $T2^{-\mu}$.

Before we present our construction in detail, we lay out the mathematical model in which we will be working in this section.

3.1 Overview of the backbone model

In order to make the analysis mathematically precise, we adopt the backbone model of GKL

The backbone model is parameterized by the parameters κ, n, t, p, q. The meaning of these parameters will become clear shortly.

Backbone models the blockchain as an execution of a single Interactive Turing Machine (ITM), the *environment*, which is responsible for coordinating the game. There are two more ITM specifications: The honest party and the adversary. The environment is responsible for spawning them and scheduling them for execution.

Two parameters of the protocol are n, the total number of parties, and t the number of adversarial parties. Therefore, $n-t$ is the number of honest parties. At the beginning of the game, the environment spawns $n-t$ honest parties which are termed $\Pi_1, \Pi_2, \cdots, \Pi_{n-t}$ and exactly one adversary $\mathcal{A}$. The environment then generates a common reference string, the *Genesis* block, which is provided to all of these players by invoking a certain initialization subroutine.

The execution of the game happens in discrete *rounds* which are used to model synchronous communication between the various players. The environment runs a main loop and maintains the current round count. Each iteration of this loop constitutes one round. At the beginning of each round, a *round execution subroutine* defined by Π_1 is invoked. When Π_1's subroutine is completed, the subroutine is invoked for Π_2 and so on until Π_{n-t} finishes executing their subroutine. In each of these executions, the player Π_i being invoked is not aware of their index i, but can maintain local state which persists across rounds. At the end of each round, the round execution subroutine of the adversary is executed. The adversary can also maintain local state across rounds. The maintenance of local state is formally modelled using Interactive Turing Machines. The fact that each honest party is unaware of its own index [Dionysis: XXX finish this section].

The environment also spawns a random oracle functionality $\mathcal{H}$ which produces truly random responses in the range $\{0,1\}^\kappa$ to queries.

3.2 New algorithms in the backbone model

We assume the verifier will always connect to at least one honest prover. We now move on to formally define these entities in terms of a mathematical model which can be used for our analysis.

In order to study the syntax and properties of NIPoPoWs we abstract their operation in a way that is akin to non-interactive zero-knowledge proofs (NIZK)

The backbone approach models proof-of-work discovery attempts by using a random oracle

We introduce the Prover and the Verifier in the Backbone model as follows. The Prover will extend the functionality of the miner node. The Verifier will be a completely new entity that can be spawned by the environment and process a NiPoPoW produced by the miner nodes running the Prover code.

The predicates of interest in our context are predicates on the active blockchain. Some of the predicates are more suitable for succinct proofs than others. We focus our attention in what we call *reliable* predicates; that is, predicates which are monotonic and stable notions that we define below. These predicates have the exceptional property that all honest miners share their view of them in a way that is updated in a predictable manner, with a truth-value that persists as the blockchain grows. Furthermore, we are interested in *succinct* predicates, predicates which can be proven using a succinct protocol, a protocol which requires proofs that are short in terms of the total blockchain size. Finally, we formalize the notion of non-interactivity.

The honest entities that live on the blockchain network are of three kinds: First, miners, who try to mine new blocks on top of the longest known blockchain and broadcast their blocks as soon as they are discovered. Second, full nodes, which maintain the longest blockchain but without mining. Full nodes also act as the provers in the network, producing proofs by calling a Prove function which we leave undefined for now, as it is part of the proof protocol. And third, verifiers or stateless clients, which connect to provers and ask for proofs in regards to which blockchain is the largest. The verifiers attempt to determine the value of a predicate on these chains.

A non-interactive proof of proof-of-work protocol is a game between a prover and a verifier parameterized by predicate Q which the prover tries to convince the verifier for. The predicate Q is a function of a chain $\mathcal{C}$, which the prover claims to be the longest chain.

When asked to do so by the environment, two provers generate proofs π_A, π_B claiming potentially different truth values for the predicate Q based on their claimed local longest chains. The verifier receives these proofs and accepts one of the two proofs, determining the truth value of the predicate. In this setting, one of the two proofs could be adversarial, as defined in the security section.

We define a *blockchain proof protocol* for a predicate Q as a pair (P, V) where P is called the *prover* and V is called the *verifier*. P is a PPT algorithm that is spawned by a full

node when they wish to produce a proof, accepts as input a full chain $\mathcal{C}$ and produces a proof π as its output. V is a PPT algorithm which is spawned by the environment at some round, receives a pair of proofs (π_A, π_B) from both an honest party and the adversary and returns its decision $d \in \{T, F, \perp\}$ before the next round and terminates. The honest parties produce proofs for V using P.

3.3 Blockchain addressing

Blockchains, or *chains*, are finite sequences of blocks such that they obey the *blockchain property* that for every block in the chain there exists a pointer to its previous block in the chain. A chain $\mathcal{C}$ is *anchored* if its first block is the designated *Genesis* block, a known reference string. We denote the genesis block as *Gen*.

For chain addressing we use the Python range notation of brackets [ ]. A positive number in a bracket indicates the indexed block in the chain, starting from zero. For example, if $\mathcal{C}$ is anchored, then $\mathcal{C}[0] = Gen$. A negative number indicates a block indexed from the end. For example $\mathcal{C}[-1]$ indicates the tip of the chain. A range of two numbers $\mathcal{C}[i:j]$ indicates a subarray starting on index i (inclusive) and ending on index j (exclusive), which can be negative. Omitting one of the range indices takes it to the beginning or end of the chain respectively. So, for example $\mathcal{C}[1:]$ indicates a chain with the first block omitted and $\mathcal{C}[:-k]$ indicates a chain with the last k blocks omitted, the "stable" part of the chain according to

Given two chains $\mathcal{C}_1$ and $\mathcal{C}_2$ we write $\mathcal{C}_1\mathcal{C}_2$ to indicate the concatenation of these two chains into a new chain. Then the $\mathcal{C}_2[0]$ must point back to $\mathcal{C}_1[-1]$. Similarly with a block B we write $\mathcal{C}_1 B$ to indicate the concatenation of block B to the tip of $\mathcal{C}_1$, implying that block B points back to $\mathcal{C}_1[-1]$ to preserve the blockchain property.

We will also use block-based addressing with the range notation of curly braces { }. Given a chain $\mathcal{C}$ and two blocks A, Z which are in $\mathcal{C}$, we denote $\mathcal{C}\{A:Z\}$ the subarray of the chain starting from block A (inclusive) and ending in block Z (exclusive). Again we can omit blocks from the range to take the chain to the beginning or end respectively. Hence, for example, $\mathcal{C}\{A:\}$ indicates that the subarray of the chain from block A and until the end of the chain is to be taken.

Some helper functions need to be defined. The *id* function calculates the id of a block given its data. This is done by applying the hashfunctions H and G in a manner similar to

The id of a block must always be less than or equal to the mining target T to satisfy the proof-of-work

In this paper, we will extend blocks to contain multiple pointers to previous blocks. Hence, certain blocks can be omitted from a chain, obtaining a subchain. However, the subchain must still maintain the blockchain property that each block must contain a pointer to its previous block in the sequence (except for the first block in the sequence). Subarrays of chains obtained by addressing with [ ] or { } are, naturally, subchains, as the only omit blocks from the beginning or the end of a chain. It will also become apparent that it is useful to omit certain blocks from the middle of a chain, while maintaining the blockchain property.

While chains are sequences and not sets, we will use some set notation for convenience. The notation $B \in \mathcal{C}$ will be used to indicate that a block B is part of a chain $\mathcal{C}$, while $B \notin \mathcal{C}$ will be used to indicate that it isn't. We will use $\emptyset$ to denote the empty blockchain. Finally, given a chain $\mathcal{C}$ and two subchains $\mathcal{C}_1$ and $\mathcal{C}_2$ of it, we will write $\mathcal{C}_1 \subseteq \mathcal{C}_2$ to indicate that $\mathcal{C}_1$ is a (potentially improper) subchain of $\mathcal{C}_2$, i.e. that for all blocks B of $\mathcal{C}_1$, we have that $B \in \mathcal{C}_2$, while $\mathcal{C}_1 \nsubseteq \mathcal{C}_2$ will be used to indicate that it isn't. Because both $\mathcal{C}_1 \subseteq \mathcal{C}$ and $\mathcal{C}_2 \subseteq \mathcal{C}$, we can define $\mathcal{C}_1 \cup \mathcal{C}_2$ to mean the chain obtained by sorting the blocks contained in both $\mathcal{C}_1$ and $\mathcal{C}_2$ into a sequence, provided that we can ensure the blockchain property is maintained. Hence, for example, if blocks (B_1, B_2, B_3) form a blockchain $\mathcal{C}$ such that B_2 points to B_1 and B_3 points to B_2, but also B_3 points to B_1, then letting $\mathcal{C}_1 = (B_1, B_3)$ and $\mathcal{C}_2 = (B_2)$, we note that both $\mathcal{C}_1$ and $\mathcal{C}_2$ are blockchains and that $\mathcal{C}_1 \cup \mathcal{C}_2$ is a blockchain. Additionally, take two anchored chains $\mathcal{C}_1$ and $\mathcal{C}_2$, and note that $\mathcal{C}_1$ and $\mathcal{C}_2$ belong to the same blocktree but can potentially have forked. Then we use the notation $\mathcal{C}_1 \cap \mathcal{C}_2$ to mean the anchored chain $\{B : B \in \mathcal{C}_1 \wedge B \in \mathcal{C}_2\}$ provided it satisfies the blockchain property. We also define the lowest common ancestor of these chains $\mathsf{LCA}(\mathcal{C}_1, \mathcal{C}_2) = (\mathcal{C}_1 \cap \mathcal{C}_2)[-1]$. We will also freely use set builder notation $\{B \in \mathcal{C} : p(B)\}$ to obtain a subchain $\mathcal{C}'$ from a chain $\mathcal{C}$ by filtering with a block predicate $p(B)$, provided the blockchain property is maintained. Note that in all these cases, because we are talking about blockchains and not blocktrees, any set of blocks has a unique order obtained by topologically sorting the blocks based on the order indicated by the underlying chain and so can be put into a unique sequence. For two chains $\mathcal{C}_1$ and $\mathcal{C}_2$, if $\mathcal{C}_1[0] = \mathcal{C}_2[0]$ and $\mathcal{C}_1[-1] = \mathcal{C}_2[-1]$, we will say that the two chains *span* the same range of blocks.

Given a chain $\mathcal{C}$ and a level μ, we define the *upchain operator* $\mathcal{C}{\uparrow^\mu}$ to mean the chain $\{B \in \mathcal{C} : level(B) \geq \mu\}$. We will call such a chain that contains only μ-superblocks a *μ-superchain*. Given a chain $\mathcal{C}'$ and a chain $\mathcal{C}$ such that $\mathcal{C}' \subseteq \mathcal{C}$, we define the *downchain operator* $\mathcal{C}'{\downarrow}^{\mathcal{C}}$ as the chain $\mathcal{C}[\mathcal{C}'[0] : \mathcal{C}'[-1]]$ and we will call $\mathcal{C}$ the *underlying chain* of $\mathcal{C}'$. If the underlying chain is implied by context, for example, if the chain $\mathcal{C}'$ was obtained by applying the upchain operator on a known underlying chain $\mathcal{C}$, we will simply write $\mathcal{C}'{\downarrow}$. Note as well that the $\mathcal{C}{\uparrow}$ operator is absolute, in the sense that $(\mathcal{C}{\uparrow^\mu}){\uparrow^{\mu+i}} = \mathcal{C}{\uparrow^{\mu+i}}$.

If a set of consecutive rounds S is given, we define $\mathcal{C}^S = \{B \in \mathcal{C} : B \text{ was generated during } S\}$.

3.4 Desired properties

We now define two desired properties of a non-interactive proof protocol, in particular *succinctness* and *security*.

Definition 3.1. (Security) A *blockchain proof protocol* is *secure* if for all environments and for all PPT adversaries $\mathcal{A}$ the output of V on round r is the same as the evaluation of $Q(\mathcal{C})$ on all honest parties' chains $\mathcal{C}$ as long as $Q(\mathcal{C})$ is the same for all honest parties after round r.

Definition 3.2. (Succinctness) A *blockchain proof protocol* is *succinct* if the maximum proof size $|\pi|$ across all honest provers for a given round r is $O(polylog(r))$.

Based on these definitions, it is easy to see that it is trivial to construct a secure but not succinct protocol: The prover provides the chain $\mathcal{C}$ itself as a proof and the verifier simply compares the proofs received by length: Since the longest proof is literally the longest chain, the protocol is trivially secure but not succinct. The challenge we will solve over the next sections is to provide a non-interactive protocol which at the same time achieves security and succinctness over a large class of predicates.

3.5 Provable chain predicates

Not all predicates can be proved. In this section we characterize the class of blockchain predicates that will be of interest to us for constructing proofs. A useful property of predicates in our context is *monotonicity*. Based on predicate monotonicity, we show that a predicate has *liveness* and *persistence*. Formalizing the security of our blockchain proofs will be preconditioned on such blockchain predicates. Given that we are in a decentralized setting it can be the case that for certain sensible blockchain predicate, there will be honest nodes that have not reached a conclusion about its value. For this reason we allow our predicates to have an undefined value $\perp$ before they take on a truth value of 0 or 1. We define a number of predicate properties and prove a simple but useful relation between them.

Definition 3.3. (Predicate monotonicity) A chain predicate $Q(\mathcal{C})$ is *monotonic* if for all chains $\mathcal{C}$ and for all blocks B we have that: $Q(\mathcal{C}) \neq \perp \Rightarrow Q(\mathcal{C}) = Q(\mathcal{C}B)$.

(Predicate switching position) A chain predicate $Q(\mathcal{C}) \neq \perp$ has a switching position $i \in \mathbb{N}$ in the chain $\mathcal{C}$ defined as the minimum i such that: $Q(\mathcal{C}[:i]) \neq \perp$.

(Predicate stability) Parameterized by $k \in \mathbb{N}$ and a chain $\mathcal{C}$, a chain predicate Q has *stability* if its switching position i satisfies $i < |\mathcal{C}| - k$.

(Predicate persistence) Parameterized by $k \in \mathbb{N}$ and a chain $\mathcal{C}$, a chain predicate Q has *persistence* if the following is true: When in a certain round the predicate is k-stable in some honest party's chain, then whenever it is k-stable in any honest party's chain, the truth value of the predicate is the same. Furthermore, the change of truth value from $\perp$ to its final truth value happened at the same blockchain depth.

Theorem 3.4 (Predicate persistence). *If a chain predicate is monotonic, then it satisfies persistence with parameter $k = 2\eta\kappa f$ with probability at least $1 - e^{-\Omega(\kappa)}$.*

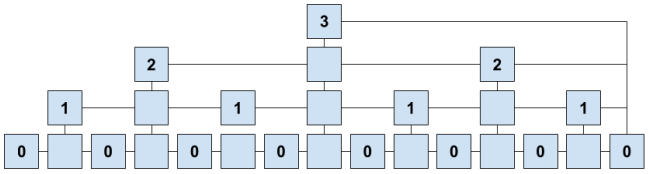
All full proofs are provided in the appendix.

4 CONSENSUS LAYER SUPPORT

4.1 The interlink pointers data structure

We will now start constructing a concrete blockchain proof protocol. Before we're able to define the prover and the verifier algorithms, we need to propose certain modifications to the consensus layer.

Figure 1: The hierarchical blockchain. Higher levels have achieved a lower target (higher difficulty) during mining.



We now observe that some blocks will achieve a much lower id. Observe then that in expectation, half of the blocks will be of level 1, 1/4 of the blocks will be of level 2, 1/8 will be of level 3 and $1/2^{\mu}$ blocks will be of level μ. In expectation, the number of superblock levels of a chain $\mathcal{C}$ will be $\log(\mathcal{C})$.

Figure 1 illustrates the blockchain superblocks starting from level 1 and going up to level 4 in case these blocks are distributed exactly according to expectation. Note that each level contains half the blocks of the level below.

The *interlink* data structure is proposed to be included in each block, replacing the existing pointer to the previous block with a list of pointers to a small number of previous blocks. For each superblock level μ, this data structure contains a pointer to the most recent preceding block of level μ. The algorithm for this construction is shown in Algorithm 1 and is borrowed from

The updateInterlink algorithm accepts a block B', which already has an interlink data structure defined on it. The function then evaluates the interlink data structure which needs to be included as part of the next block. The algorithm proceeds by copying the existing interlink data structure and then modifying some of its entries from level 0 to the level of block B' to point to the block B'. Observe that the pointers stored are block ids and hence block data needs to be looked up based on blockid. To this end, the full node maintains a *blockbyid* dictionary which, queried with the block id, returns the block data. Finally, the full node also maintains a *depth* data structure which, given a block, returns its distance from the Genesis block.

Algorithm 1 updateInterlink

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

4.2 Superchain quality

We now formally describe the distribution of superblocks within a blockchain.

Definition 4.1 (Locally good superchain). A superchain $\mathcal{C}'$ of level μ with underlying chain $\mathcal{C}$ is said to be μ-locally-good with respect to security parameter δ, written $\mathsf{local\text{-}good}_\delta(\mathcal{C}', \mathcal{C}, \mu)$, if $|\mathcal{C}'| > (1-\delta)2^{-\mu}|\mathcal{C}|$.

Lemma 4.2 (Local goodness). *Assume chain $\mathcal{C}$ contains only honestly-generated blocks and has been adopted by an honest party in an execution with random network scheduling. Then for all levels μ, for all constant $\delta > 0$, all continuous subchains $\mathcal{C}' = \mathcal{C}[i:j]$ with $|\mathcal{C}'| \geq m$ are locally good, $\mathsf{local\text{-}good}_\delta(\mathcal{C}', \mathcal{C}, \mu)$, with overwhelming probability in m.*

Definition 4.3 (Chain superquality). The (δ, m) superquality property Q^{μ}_{scq} of a chain $\mathcal{C}$ pertaining to level μ with security parameters $\delta \in \mathbb{R}$ and $m \in \mathbb{N}$ states that for all $m' \geq m$, it holds that $\mathsf{local\text{-}good}_\delta(C{\uparrow^\mu}[-m':], C{\uparrow^\mu}[-m':]{\downarrow}, \mu)$. That is, all suffixes that are sufficiently large are locally good.

Lemma 4.4 (Superquality). *For all levels μ, for all constant $\delta > 0$, a chain $\mathcal{C}$ containing only honestly-generated blocks adopted by an honest party in an execution with random network scheduling has (δ, m)-superquality at level μ with overwhelming probability in m.*

Definition 4.5 (Multilevel quality). A μ-superchain $\mathcal{C}'$ is said to have *multilevel quality*, written $\text{multi-good}_{\delta,k_1}(\mathcal{C}, \mathcal{C}', \mu)$ with respect to an underlying chain $\mathcal{C} = \mathcal{C}'{\downarrow}$ with security parameters k_1, δ if for all $\mu' < \mu$ it holds that for any $\mathcal{C}^* \subseteq \mathcal{C}$, if $|\mathcal{C}^*{\uparrow^{\mu'}}| = k_1$, then $|\mathcal{C}^*{\uparrow^{\mu}}| \geq (1-\delta)2^{\mu-\mu'}|\mathcal{C}^*{\uparrow^{\mu'}}|$

Note also that for $\delta < 0.5$, multilevel quality implies that $|\mathcal{C}^*{\uparrow^{\mu}}| \geq 1$.

Lemma 4.6 (Multilevel quality). *For all levels μ, for all constant $0 < \delta \leq 0.5$, a chain $\mathcal{C}$ containing only honestly-generated blocks adopted by an honest party in an execution with random scheduling has (δ, k_1) multilevel quality at level μ with overwhelming probability in k_1.*

Definition 4.7 (Good superchain). A μ-superchain $\mathcal{C}'$ is said to be *good*, written $\text{good}_{\delta,k_1}(\mathcal{C}, \mathcal{C}', \mu)$, with respect to an underlying chain $\mathcal{C} = \mathcal{C}'{\downarrow}$ if it has both superquality and multilevel quality with the same parameters (δ, m).

Corollary 4.8 (Optimistic superchain distribution). *For all levels μ, for all constant $0 < \delta < 0.5$, a chain $\mathcal{C}$ containing only honestly-generated blocks adopted by an honest party in an execution with random scheduling is (δ, m)-good at level μ with overwhelming probability in m.*

5 AN ATTACK AGAINST PROOFS OF PROOF-OF-WORK

In this section, we revisit the construction for interactive proofs of proof-of-work from

We first show that a powerful attacker can break chain superquality with non-negligible probability and we construct a concrete attack based on this observation. Maintaining chain superquality is not in the original security model. However, we show how the property affects the security of the underlying proofs. We do this by presenting a concrete attack against their scheme that allows an adversary to double spend with non-negligible probability. This attack works under the assumption that the adversary has sufficient hashing power, which is still below 50%.

5.1 The interactive proofs of proof-of-work protocol

The interactive proof of proof-of-work of

5.2 Attacking chain superquality

We present next a general attack against chain superquality that works against any blockchain protocol that incorporates the interlink data structure, cf. Section 4.1. Observe that in expectation, in a completely honest setting with random network scheduling, the honestly adopted chain will be μ-locally-good for sufficiently long subchains everywhere, as proven in Lemma 4.2.

Let us now construct an adversary to break this property at level μ. Suppose the adversary $\mathcal{A}$ controls a portion of the hashing power equal to t/n. $\mathcal{A}$ then works as follows. Assume she wants to attack the honest party B in order to have him adopt a chain $\mathcal{C}_B$ which has a bad distribution of superblocks, i.e. such that local goodness is violated in some sufficiently long subchain. She continuously determines the current chain $\mathcal{C}_B$ adopted by B (as honest chain adoption is deterministic, this can be easily done by inspecting the messages exchanged by the honest parties). The adversary starts playing after $|\mathcal{C}_B| \geq 2$. If $\mathit{level}(\mathcal{C}_B[-1]) < \mu$, then $\mathcal{A}$ remains idle. However, if $\mathit{level}(\mathcal{C}_B[-1]) \geq \mu$, then $\mathcal{A}$ attempts to mine an adversarial block b on top of $\mathcal{C}_B[-2]$. If she is successful, then she attempts to mine another block b' on top of b. If she is successful, she broadcasts both blocks b, b'. The adversarial mining continues until party B adopts a new chain, which can be due to two reasons: Either the adversary managed to successfully mine b, b' on top of $\mathcal{C}_B[-2]$ and succeeded in having B adopt it; or one of the honest parties (potentially B) was able to mine a block which was subsequently adopted by B. In either case, the adversary continues with the strategy by inspecting $\mathcal{C}[-1]$ and acting accordingly.

Assume now that an honestly-generated μ-superblock has been adopted by B at position $\mathcal{C}_B[i]$ at some round r. Let us now examine the probability that $\mathcal{C}_B[i]$ will remain a μ-superblock in the long run. Suppose $r' > r$ is the first round after r during which any block is generated. It is clear that $\mathcal{A}$ will succeed in this attack with non-negligible probability and cause B to abandon the μ-superblock from their adopted chain. Therefore, there will be some δ such that the adversary will be able to cause such variance with non-negligible probability in m. This suffices to show that superquality is harmed by this attack. To avoid repeating ourselves, the full understanding of the probabilities for this

attack will become clear in the next section where the double-spending attack is explored in detail.

5.3 From chain superquality attacks to double-spending

Extending the above attack, we now modify the superquality attacker into an attacker, $\mathcal{A}$, that can cause a double spending attack in the proof of proof-of-work construction. As before, $\mathcal{A}$ targets the proofs generated by the honest party B by violating μ-superquality in B's adopted chain. $\mathcal{A}$ begins by remaining idle until a certain chosen block b. After block b is produced, $\mathcal{A}$ starts mining a secret chain which forks off from b akin to a selfish mining attacker

$\mathcal{A}$ keeps extending their own secret chain as usual. However, whenever a μ-superblock is adopted by B, the adversary temporarily pauses mining in their secret chain and devotes her mining power to harm the μ-superquality of B's adopted chain. Intuitively, for large enough μ, the time spent trying to harm superquality will be limited, because the probability of a μ-superblock occurring will be small. Therefore, the adversary's superchain quality will be larger than the honest parties' superchain quality (which will be harmed by the adversary) and therefore, even though the adversary's 0-level blockchain will be shorter than the honest parties' 0-level blockchain, the adversary's μ-superchain will be longer than the honest parties' μ-superchain.

We now proceed to calculate the attack probabilities precisely and in formal detail. We simplify the above attack to ease the formal probabilistic analysis. The attack is parameterized by two parameters r and μ which are picked by the adversary. μ will be the superblock level at which the adversary will attempt to produce a proof longer than the honest proof. The modified attack works precisely as follows: Without loss of generality, we fix block b to be the Genesis block. The adversary always mines on the secret chain which forks off from genesis, unless a *superblock generation event* occurs. If a superblock generation event occurs, then the adversary pauses mining on the secret chain and attempts a *block suppression attack* on the honest chain. The adversary devotes exactly r rounds to this suppression attack and then resumes mining on their secret chain. Our goal is to show that, despite this simplification (of fixing r) which is harmful to the adversary, the probability of a successful attack is non-negligible for certain (reasonable) values of the protocol parameters.

A superblock generation event is detected by the adversary by monitoring the network. Whenever an honest party diffuses an honestly-generated μ-superblock at the end of a given round r_1, then the superblock generation event will have occurred and the adversary will starting devoting their mining power to block suppression starting from the next round.

A block suppression attack works as follows. Let B be the honestly generated μ-superblock which was diffused at the end of the previous round. If the round was not uniquely successful, let B be any of the diffused honestly-generated μ-superblocks. Let B be the tip of an honest chain $\mathcal{C}_B$. The adversary first mines on top of $\mathcal{C}_B[-2]$. If she is successful in mining a block B', then she continues extending the chain ending at B' (to mine B'' and so on). The value r is fixed, and so the adversary devotes exactly r rounds to this whole process; the adversary will keep mining on top of $\mathcal{C}_B[-2]$ (or one of the adversarially-generated extensions of it) for exactly r rounds, regardless of whether B' or B'' have been found. At the same time, the honest parties will be mining on top of B (or a competing block in the case of a non-uniquely successful round). Again, further successful block diffusion by the honest parties shall not affect that the adversary is going to spend exactly r rounds for suppression.

Having laid out the attack scenario precisely, we are ready to prove that it will succeed with non-negligible probability. In fact, as we will see, perhaps surprisingly, it will succeed with overwhelming probability for the right choice of protocol values.

Theorem 5.1 (Double-spending attack). *There exist parameters p, n, t, q, μ, δ, with $t \leq (1-\delta)(n-t)$, and a double spending attack against KLS PoPoW that succeeds with overwhelming probability.*

Remark. It is worth isolating the mistake in the security proof from the interactive construction paper

Regardless, their basic argument and construction is what we will use as a basis for constructing a system that is both provably secure and succinct under the same assumptions, albeit requiring a more complicated proof structure to obtain security.

6 BLOCKCHAIN SUFFIX PROOFS

We now provide a concrete non-interactive PoPoW construction which allows proving certain predicates Q of the chain $\mathcal{C}$. The construction will be done in two steps: First, we will build a construction which is limited to a class of predicates that is easy to prove. Later, we will extend this class of predicates by augmenting the construction.

Among the predicates which are stable, we now limit ourselves to the class of predicates which are functions of only the chain suffix $\mathcal{C}[-k-1:]$. We call these predicates $(k+1)$-*suffix sensitive*. As we will illustrate, these predicates are significantly easier to prove, but at the same time allow us to use the construction as a scaffold for the more generic case.

Definition 6.1 (Suffix sensitivity). A chain predicate Q is called k-suffix sensitive if for all chains $\mathcal{C}, \mathcal{C}'$ with $|\mathcal{C}| \geq k$ and $|\mathcal{C}'| \geq k$ such that $\mathcal{C}[-k:] = \mathcal{C}'[-k:]$ we have that $Q(\mathcal{C}) = Q(\mathcal{C}')$.

Given this suffix-sensitivity definition, we deduce that for each suffix-sensitive predicate Q there must exist a "short version" predicate $\tilde{Q}$ which can deduce the value of $Q(\mathcal{C})$ by only examining the suffix of the chain, that is $Q(\mathcal{C}) = \tilde{Q}(\mathcal{C}[-k:])$.

Suffix-sensitive predicates allow proving that a recent transaction occurred in the longest chain and in a block buried under at most k blocks. Because the predicates we are interested in are stable, they must also pertain to properties that

are independent of the last k blocks. Therefore, combining the two, the predicate can only describe something that is visible at the exact block $\mathcal{C}[-k-1]$ and that will also remain true as the chain grows in perpetuity (due to monotonicity). These predicates may seem contrived for blockchains like Bitcoin. However, they make sense for blockchains such as Ethereum which maintain state

6.1 Construction

We will now describe the details of the NIPoPoW construction by presenting a specific prover and verifier. We present a generic form of the verifier first and the prover afterwards. The generic form of the verifier will work with any practical suffix proof protocol (we make this more exact below). Therefore, it makes sense to describe the generic verifier first before we talk about the specific instantiation of our protocol. The generic verifier is given access to call a proof comparison operator $\leq_m$ which is protocol-specific. We therefore begin the description of our protocol by first illustrating the generic verifier. Next, we describe the prover specific to our protocol. Finally, we show the instantiation of the $\leq_m$ operator, which plugs into the generic verifier to make a concrete verifier for our protocol.

6.1.1 The generic verifier

The Verify function of our NIPoPoW construction is described in Algorithm 2. The verifier algorithm is parameterized by a chain predicate Q and security parameters k and m. The parameter k pertains to the amount of proof-of-work needed to bury a block so that it is believed to remain stable; in bitcoin's case, we often set $k = 6$. The parameter m is a security parameter pertaining to the prefix of the proof, which connects the Genesis block to the k-sized suffix. The verifier receives several proofs by different provers in a collection of proofs $\mathcal{P}$, at least one of which must be generated by an honest prover. Iterating over these proofs, it extracts the best one by comparing these proofs.

Each of these proofs is a chain. For honest provers, these chains are subchains of the currently adopted chain. The proofs consist of two parts, π and χ, such that $\pi\chi$ is a valid chain. χ is called the proof suffix, for which we require that $|\chi| = k$, while π is called the proof prefix. For honest provers, χ will be the last k blocks of their adopted chain, while π will consist of a selected subset of blocks from the rest of their chain preceding χ. The exact method of choice of this subset will become clear when we describe the concrete prover in the next section.

The verifier compares the various proofs provided to it by calling the $\geq_m$ comparison operator, which is defined by the protocol. We will get to the operator's definition after we have examined the prover algorithm. All proofs are checked for validity before comparison by checking that $|\chi| = k$ and subsequently using the *validChain* function. This function simply checks that $\pi\chi$ is an anchored blockchain by ensuring that the first block in the proof is the common reference string (Genesis) and that each block in the proof contains a pointer to the previous block within the proof.

Algorithm 2 The Verify algorithm for the NIPoPoW protocol

```
1:  function Verify^Q_{m,k}(P)
2:      π̃ ← (Gen)                          ▷ Trivial anchored blockchain
3:      for (π, χ) ∈ P do                  ▷ Examine each proof (π, χ) in P
4:          if validChain(πχ) and |χ| = k and π ≥_m π̃ then
5:              π̃ ← π
6:              χ̃ ← χ                      ▷ Update current best
7:          end if
8:      end for
9:      return Q̃(χ̃)
10: end function
```

At each loop iteration, the verifier compares the next candidate proof prefix π against the currently best known proof prefix $\tilde{\pi}$ by calling $\pi \geq_m \tilde{\pi}$. If the candidate prefix is better than the currently best known proof prefix, then the currently known best prefix is updated by setting $\tilde{\pi} \leftarrow \pi$. When the best known prefix is updated, the suffix $\tilde{\chi}$ associated with the best known prefix is also updated to match the suffix χ of the candidate proof by setting $\tilde{\chi} \leftarrow \chi$. While $\tilde{\chi}$ is needed for the final predicate evaluation, it is not used as part of any comparison, as it has the same size k for all proofs. The best known proof prefix is initially set to (Gen), the trivial anchored chain containing only the genesis block. Any well-formed proof compares favourably against the trivial chain.

After the end of the for loop, the verifier will have determined the best proof $(\tilde{\pi}, \tilde{\chi})$. We will later prove that this proof will necessarily belong to an honest prover with overwhelming probability. Since the proof has been generated by an honest prover, it is associated with an underlying honestly adopted chain $\mathcal{C}$. The verifier then evaluates the predicate Q on the underlying chain associated with the proof by invoking $\tilde{Q}$.

This generic form of the verifier will work for any protocol for suffix proofs, as long as the underlying operator $\leq_m$ is defined in a manner that is transitive and returns the same result if invoked multiple times with the same inputs with overwhelming probability.

6.1.2 The concrete prover

The NIPoPoW honest prover construction is shown in Algorithm 3. The honest prover is supplied with an honestly adopted chain $\mathcal{C}$ and security parameters m, k, δ and returns a proof $\pi\chi$, which is a chain. The proof consists of a suffix χ and a prefix π. The suffix χ is simply the last k blocks of $\mathcal{C}$. The prefix π is constructed by selecting various blocks and adding them to π. which consists of a number of blocks for every level μ. At the highest possible level at which at least m blocks, all these blocks are presented. Then, inductively, for every superchain of level μ that is included in the proof, the

suffix of length m is taken. Then the underlying superchain of level $\mu - 1$ covering the same span as that suffix is also included, until level 0 is reached. This underlying superchain will have $2m$ blocks in expectation and always at least m blocks.

Algorithm 3 The Prove algorithm for the NIPoPoW protocol

```
1: function Prove_{m,k,δ}(C)
2:     B ← C[0]                                   ▷ Genesis
3:     for μ = |C[−k].interlink| down to 0 do
4:         α ← C[: −k]{B :}↑^μ
5:         π ← π ∪ α
6:         if good_{δ,m}(C, α, μ) then
7:             B ← α[−m]
8:         end if
9:     end for
10:    χ ← C[−k :]
11:    return πχ
12: end function
```

The algorithm returns a chain $\pi\chi$. In this chain, χ is the suffix of the blockchain containing the most recent k blocks. π is a subchain of the underlying blockchain with the last k blocks removed, $\mathcal{C}[: -k]$. In each iteration of the for loop, blocks of level μ are considered, starting from the top-most level $|\mathcal{C}[-k].\mathsf{interlink}|$ and descending down to level 0.

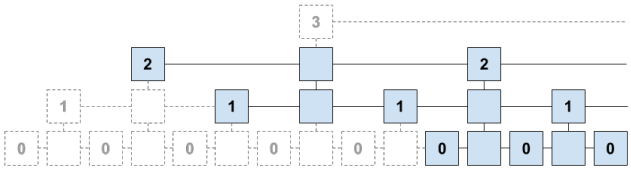
When we take a μ-superchain and are interested in its last m blocks, we fill the same range of blocks with blocks from the superchain of level $\mu - 1$ below. All the blocks of level μ which are within this last m blocks range will also be superblocks of level $\mu - 1$ and so we do not want to keep them in the proof structure twice. Note furthermore that no check is necessary to make sure the top-most level has at least m blocks, even though the verifier requires this. The reason is the following: Assume the blockchain has at least m blocks in total. Then when a superchain of level μ has less than m blocks in total, these blocks will all be necessarily included into the proof by a lower-level superchain $\mu - i$ for some $i > 0$. Therefore, it does not hurt to add them to the set π earlier.

Figure 2 contains 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 superblock level 2 are included for support spanning the same range. For the last 3 blocks of the level 2 superchain, blocks of level 1 are included for support.

6.1.3 The concrete verifier

Finally, the $\geq_m$ operator which performs the actual comparison of proofs is presented in Algorithm 4. It takes two proofs π_A and π_B as parameters and returns true if the first proof is winning, otherwise it return false to indicate the second proof is winning. The comparison operator computes the lowest common ancestor block b between the two proofs as

Figure 2: Non-interactive proof-of-work prefix π for $m = 3$ on a chain good everywhere.



$(\pi_A \cap \pi_B)[-1]$. As parties A and B agree that the blockchain is the same up to block b, arguments will then be taken for the diverging chains after block b. The best possible argument from each player's proof is extracted by calling the best-arg$_m$ function. We call the willingness of the verifier to allow each prover to be evaluated based on their best argument the *principle of charity*. To find the best argument of a proof π given an LCA block b, best-arg$_m$ first collects all the μ indices which point to superblock levels that contain valid arguments after block b. Argument validity requires that there are at least m μ-superblocks following block b, which is captured by the comparison $|\pi\uparrow^\mu \{b :\}| \geq m$. 0 is always considered a valid level, regardless of how many blocks are present there. These level indices are collected into a set M. For each of these levels, the score of their respective argument is evaluated by weighting the number of blocks by the exponent of the level as $2^\mu|\pi\uparrow^\mu \{b :\}|$. The highest possible score across all levels is then returned. Once the score of the best argument of both players A and B is known, they are then directly compared and the winner is returned. Note that an advantage is given to the adversary in case of a tie by using the $\geq$ operator which favours player A.

Algorithm 4 The algorithm implementation for the $\geq$ operator to compare two proofs in the NIPoPoW protocol parameterized with security parameter m. Returns True if the underlying chain of player A is deemed longer than the underlying chain of player B

```
1: function best-arg_m(π, b)
2:     M ← {μ : |π↑^μ {b :}| ≥ m} ∪ {0}
3:     return max_{μ∈M}{2^μ|π↑^μ {b :}|}
4: end function
5: operator π_A ≥_m π_B
6:     b ← (π_A ∩ π_B)[−1]                        ▷ LCA
7:     return best-arg_m(π_A, b) ≥ best-arg_m(π_B, b)
8: end operator
```

6.2 Security

We will call a query to the random oracle *μ-successful* if the random oracle returns a value h such that $h \leq 2^{-\mu}T$.

In order to prove our construction secure, we first define the boolean random variables X_r^μ, Y_r^μ and Z_r^μ, akin to the random variables X_r, Y_r and Z_r of the backbone setting

We now extend the definition of a *typical execution* from the backbone setting

Definition 6.2. (Typical execution) An execution of the protocol is (ϵ, η)-*typical* if:

Block counts don't deviate. For all $\mu \geq 0$ it holds that for any set S of consecutive rounds with $|S| \geq 2^{\mu}\eta\kappa$, the following hold:

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

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

Chain regularity. No insertions, no copies, and no predictions

Theorem 6.3 (Typicality). *Executions are (ϵ, η)-typical with overwhelming probability in κ.*

The following lemma is at the heart of the security proof that will follow.

Lemma 6.4. *Suppose S is a set of consecutive rounds $r_1 \ldots r_2$ and $\mathcal{C}_B$ is a chain adopted by an honest party at round r_2 of a typical execution. Let $\mathcal{C}_B^S = \{b \in \mathcal{C}_B : b$ was generated during $S\}$. Let $\mu_{\mathcal{A}}, \mu_B \in \mathbb{N}$. Suppose $\mathcal{C}_B^S{\uparrow}^{\mu_B}$ is good. Suppose $\mathcal{C}'_{\mathcal{A}}$ is a $\mu_{\mathcal{A}}$-superchain containing only adversarially generated blocks generated during S and suppose that $|\mathcal{C}'_{\mathcal{A}}| \geq k$. Then:*

$$2^{\mu_{\mathcal{A}}}|\mathcal{C}'_{\mathcal{A}}| < \frac{1}{3}2^{\mu_B}|\mathcal{C}_B^S{\uparrow}^{\mu_B}|$$

Definition 6.5 (Adequate level of honest proof). Let π be an honestly generated proof constructed upon some adopted chain $\mathcal{C}$ and let $b \in \pi$.

Then μ' is defined as follows:

$$\mu' = \max\{\mu : |\pi\{b:\}{\uparrow}^{\mu}| \geq \max(m+1, (1-\delta)2^{-\mu}|\pi\{b:\}{\uparrow}^{\mu}{\downarrow}|)\}$$

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 6.6. *Let π be some honest proof generated with security parameters δ, m. Let $\mathcal{C}$ be the underlying chain, $b \in \mathcal{C}$ be any block and μ' be the adequate level of the proof with respect to b and the same security parameters.*

Then $\mathcal{C}\{b:\}{\uparrow}^{\mu'} = \pi\{b:\}{\uparrow}^{\mu'}$.

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 $b = \mathsf{LCA}(\pi_{\mathcal{A}}, \pi_B)$ and let μ'_B be the adequate level of π_B with respect to b. Then $\mu'_B \geq \mu_B$.

Theorem 6.7. *The non-interactive proofs-of-proof-of-work construction for k-stable suffix-sensitive predicates is secure with overwhelming probability in κ.*

Remark (Variance attacks). The critical issue addressed by this security proof is to avoid Bahack-style attack

6.3 Succinctness

We now prove that our construction produces succinct proofs.

Observe that full succinctness in the standard honest majority model is impossible to prove for our construction. To see why, recall that an adversary with sufficiently large mining power can significantly harm superquality as described in Subsection 5.2. By causing this harm in superquality of a sufficiently low level μ, for example $\mu = 3$, the adversary can cause the honest prover to digress in their proofs from high-level superchains down to low-level superchains, causing a linear proof 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, in expectation, will be of size $|\mathcal{C}|/2$.

Having established security in the general case of the standard honest majority model, we therefore concentrate our succinctness claims to the particular case where there is no adversary. This case, called the "optimistic succinctness" case following the lead of

Theorem 6.8 (Number of levels). *The number of superblock levels which have at least m blocks are at most $\log(|S|)$, where S is the set of all blocks produced, with overwhelming probability in m.*

The above theorem establishes that the number of superchains is small. What remains to be shown is that few blocks will be included at each superchain level.

Theorem 6.9 (Large upchain expansion). *Let $\mathcal{C}$ be an honestly generated chain and let $\mathcal{C}' = \mathcal{C}{\uparrow}^{\mu-1}[i : i+\ell]$ with $\ell \geq 4m$. Then $|\mathcal{C}'{\uparrow}^{\mu}| \geq m$ with overwhelming probability in m.*

Corollary 6.10 (Small downchain support). *Assume an honestly generated chain $\mathcal{C}$ and let $\mathcal{C}' = \mathcal{C}{\uparrow}^{\mu}[i : i+m]$. Then $|\mathcal{C}'{\downarrow}{\uparrow}^{\mu-1}| \leq 4m$ with overwhelming probability in m.*

This last theorem establishes the fact that the support of blocks needed under the m-sized chain suffix to move from one level to the one below is small. Based on this, we can establish our theorem on succinctness:

Theorem 6.11 (Optimistic succinctness). *Non-interactive proofs-of-proof-of-work produced by honest provers on honestly generated chains with random scheduling are succinct with the number of blocks bounded by $4m\log(|\mathcal{C}|)$, with overwhelming probability in m.*

Full proofs of these facts are provided in Appendix B.

6.4 Certificates of Badness

To recap what we have achieved, note that we have proved security in the general case, but succinctness only in the optimistic case where there is no adversary present and network scheduling is random.

Examine the non-optimistic case where the adversary is allowed to provide proofs, yet, as in the optimistic case, they have no mining power or adversarial scheduling capabilities. It is then still possible for that adversary to produce large incorrect proofs that use up the resources of a verifier with logarithmic time. It is important to note that a verifier knowing ahead of time that a proof will be large cannot conclude that the prover providing it is malicious, as an attack on superquality of the blockchain could be taking place, requiring an honest prover to provide long proofs.

To avoid the above undesired scenario, we now offer a generalization of our above construction which maintains the same security. However, succinctness is improved in that the protocol remains succinct in the optimistic case where the adversary has no mining power, network scheduling is random, yet the adversary *is able* to produce proofs.

Our extended construction will allow the verifier to stop processing input early, in a streaming fashion, thereby only requiring logarithmic communication complexity even when an adversary is pushing a lot of data. To achieve this, we observe that the honest proofs only need to be large if there is a violation of *goodness* of the blockchain. However, goodness is not harmed when the chain is not under attack by the adversarial computational power or network.

Having made this observation, we now require the prover to produce a *certificate of badness* in case there is a violation of *goodness* in the blockchain.

Definition 6.12 (Badness). A chain $\mathcal{C}$ is called *bad* if there is some level μ such that for some chain $\mathcal{C}' \subseteq \mathcal{C}\uparrow^{\mu}$ with $|\mathcal{C}'| \geq m$, it holds that $\neg good_{\delta,m}(\mathcal{C}, \mathcal{C}', \mu)$.

Definition 6.13 (Badness certification). A certificate of badness protocol is a proof protocol (P, V) consisting of an honest prover P and an honest verifier V with the following properties:

- **Correctness.** Given a bad honestly adopted chain $\mathcal{C}$, the honest prover $P(\mathcal{C})$ produces a certificate of badness s such that $V(s)$ outputs 1;
- **Security.** For any adversary $\mathcal{A}$ with no mining power in an environment of random network scheduling, if the adversary produces some certificate of badness s, then $V(s)$ outputs 1 with only negligible probability;
- **Succinctness.** For any adversary $\mathcal{A}$ with no mining power in an environment of random network scheduling, if the adversary produces some certificate of badness s, then V returns after reading only the first $O(polylog(|\mathcal{C}|))$ bits of the input, where $\mathcal{C}$ is the maximum honestly adopted blockchain.

The prover for our certificates of badness protocol is shown in Algorithms 5. The prover looks for two adjacent levels μ and $\mu - 1$ which violate the goodness property with sufficient error $(1-\delta)\rho$ so that it can be possible for them to contribute to an error of more than $(1-\delta)$ across non-adjacent superblock levels.

In order to make its decision, the verifier examines the provided chain of level $\mu - 1$ and counts the number of μ-superblocks and $\mu - 1$ superblocks to see if the claimed error is attained.

Algorithm 5 The badness prover which generates a succinct certificate of badness

```
1: function badness_{m,δ}(C)
2:     M ← {μ : |C↑^μ| ≥ m} \ {0}
3:     ρ ← 1/max(M)
4:     for μ ∈ M do
5:         for B ∈ C↑^μ do
6:             C' ← C↑^μ {B :}[: m]
7:             if |C'| = m then                ▷ Sliding m-sized window
8:                 C* ← C↓↑^{μ−1}
9:                 if 2|C'| < (1 − δ)^ρ |C*| then
10:                    return C*                ▷ Chain is bad
11:                end if
12:            end if
13:        end for
14:    end for
15:    return ⊥                                 ▷ Chain is good
16: end function
```

Theorem 6.14. *Algorithm 5 is a correct, secure and succinct certificate of badness protocol.*

Proof. **Correctness.** XXX
Security. XXX
Succinctness. XXX □

7 BLOCKCHAIN INFIX PROOFS

7.1 Construction

While the previous proofs we constructed were proving predicates on the blocks and transactions pertaining to the chain suffix of the last k blocks, similar to the kind of proofs possible with

We will now extend our prover to support this generalization of predicates. The generalized prover for *infix proofs* allows proving any predicate $Q(\mathcal{C})$ that depends on a number of blocks that can appear anywhere within the chain. These blocks constitute a *subset* $\mathcal{C}'$ of blocks which may not necessarily be a stand-alone blockchain. For our proofs to work, the blocks contained within the set have to be k-stable (recall that this means that they must not be one of the most recent k blocks).

The proofs are able to prove any predicate of the general class of predicates that depend on the inclusion of a constant number of blocks of the chain. This allows proving powerful statements such as, for example, whether a transaction took place at any point in history. While, as illustrated in the previous section, it is possible to build an SPV client which is aware of the latest blocks using suffix proofs, a client which needs to learn about a transaction buried deep in the blockchain cannot expect suffix proofs to be efficient. Hence, the infix proofs construction is necessary.

Definition 7.1 (Infix sensitivity). A chain predicate Q is called k-infix sensitive if it can be written in the form

$$Q(\mathcal{C}) = \begin{cases} \text{true, if } \exists \mathcal{C}' \subseteq \mathcal{C} : |\mathcal{C}'| \leq k \wedge P_T(\mathcal{C}') \\ \text{false, if } \exists \mathcal{C}' \subseteq \mathcal{C} : |\mathcal{C}'| \leq k \wedge P_F(\mathcal{C}') \\ \text{undefined, } otherwise \end{cases}$$

Where P_F and P_T are arbitrary predicates such that it is impossible for an honestly-adopted chain $\mathcal{C}$ to have a $\mathcal{C}' \subseteq \mathcal{C} : P_T(\mathcal{C}') \wedge P_F(\mathcal{C}')$. Note that $\mathcal{C}'$ is a blockset and may not necessarily be a blockchain.

We require that all infix-sensitive predicates can never be both satisfied and unsatisfied simultaneously. This can be enforced by the honest parties through their application-layer verification function, $\mathcal{V}$ which is purposefully left undefined by the backbone protocol and we will also leave undefined.

Similarly to suffix-sensitive predicates, infix-sensitive predicates Q can be evaluated by a short evaluation. In particular, here, because of the form of infix-sensitive predicates, to evaluate $Q(\mathcal{C})$ it suffices to pass to Q any short blockset $\mathcal{C}'' \subseteq \mathcal{C}$ such that there is a $\mathcal{C}' \subseteq \mathcal{C}''$ which satisfies the condition that $|\mathcal{C}'| \leq k \wedge P(\mathcal{C}')$ where P is one of P_T or P_F.

The manner to construct these proofs is shown in Algorithm 7. The infix prover accepts two parameters: The chain $\mathcal{C}$ which is the full blockchain and $\mathcal{C}'$ which is a subchain of the blockchain whose blocks are of interest for the predicate in question. This prover simply calls the previous suffix prover to produce a proof as usual. Then, having the prefix π and suffix χ of the suffix proof in hand, the infix prover adds a few auxiliary blocks to the prefix π. The prover ensures that these auxiliary blocks form a verifiable chain with the rest of the proof π in a manner that the blocks can be arranged in topological order such that a pointer exists from each non-Genesis block to a preceding block and each preceding block is pointed to by a unique block. Such auxiliary blocks are collected as follows: For every block B of the subchain $\mathcal{C}'$, the immediate previous (E') and next (E) blocks in the proof π are found. Then, a chain of blocks R which connects E back to B' is found by the algorithm followDown. Finally, observe that if E' is of level μ, then there can be no other μ level superblock between E' and B', otherwise it would have been included in π. Therefore, block B' already contains a pointer to E' in its interlink, completing the chain.

Algorithm 6 The followDown function which produces the necessary blocks to connect a superblock hi to a preceeding regular block lo, enabling infix proofs.

```
1: function followDown(hi, lo, depth)
2:     aux ← [hi]
3:     B ← hi
4:     aux ← []
5:     μ ← level(hi)
6:     while B ≠ lo do
7:         B' ← blockById[B.interlink[μ]]
8:         if depth[B'] < depth[lo] then
9:             μ ← μ − 1
10:        else
11:            aux ← aux ∪ aux'
12:            B ← B'
13:        end if
14:    end while
15:    return aux
16: end function
```

The way to connect a superblock to a previous lower-level block is implemented in Algorithm 7 as the followDown algorithm. Notice that block B' cannot be of higher or equal level than block E, otherwise it would be equal to E and the followDown algorithm would return immediately. The algorithm then proceeds as follows: Starting at block E which here is marked as hi, it tries to follow a pointer to as far as possible. If following the pointer surpasses block B', which here is marked as lo, then the following is aborted and a lower level is tried, which will cause a smaller step within the skiplist. If a pointer was followed without surpassing B', the operation continues from the new block, until eventually B' will be reached, which concludes the algorithm.

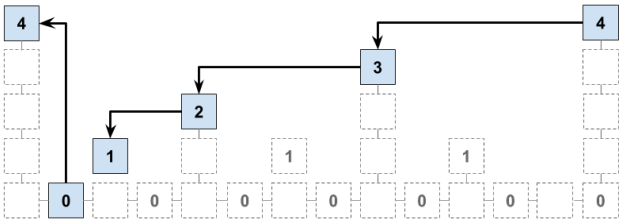
Algorithm 7 The Prove algorithm for infix proofs

```
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
8:                 break
9:             end if
10:            E' ← E
11:        end for
12:    end for
13:    return (aux ∪ π, χ)
14: end function
```

An example of the kind of auxiliary sequences that followDown produces is seen in Figure 3. This is only a portion of the proof shown at the point where the superblock levels are at level 4, but a descend to level 0 was necessary so that

a regular block would be included in the chain. Note that the level 0 block can jump immediately back up to level 4 because it has a high-level pointer.

Figure 3: An infix proof descending down to level 0 and back up to level 4. Only blue blocks are included in the proof. Blue blocks of level 4 are part of π, while the blue blocks of level 1 and 3 are produced by followDown to get to the block of level 0 which is part of $\mathcal{C}'$. Notice how in this proof the blocks happened to be distributed slightly different from expectation and there are two consecutive 0-level blocks.



The verification algorithm must then be modified as in **??**.

Algorithm 8 The verify algorithm for the NIPoPoW infix protocol

```
1:  function ancestors(B, blockById)
2:      if B = Gen then
3:          return {B}
4:      end if
5:      C ← ∅
6:      for B' ∈ B.interlink do
7:          C ← C ∪ ancestors(B') ▷ Collect all ancestors into a blockDAG
8:      end for
9:      return C
10: end function
11: function verify^Q_{m,k}(P)
12:     π̃ ← (Gen)                    ▷ Trivial anchored blockchain
13:     blockById ← ∅                ▷ Initialize empty hashmap
14:     for (π, χ) ∈ P do            ▷ Examine each proof (π, χ) in P
15:         for B ∈ π do
16:             blockById[B.id] ← B
17:         end for
18:         if validChain(πχ) and |χ| = k and π ≥_m π̃ then
19:             π̃ ← π
20:             χ̃ ← χ                ▷ Update current best
21:         end if
22:     end for
23:     return Q(ancestors(π̃[−1]))
24: end function
```

The algorithm works as before by comparing competing chain prefixes π using the standard $\geq_m$ operator and extracts a best chain prefix $\tilde{\pi}$ and $\tilde{\chi}$. However, it also maintains a blockDAG consisting of blocks from all proofs. Note that, unlike traditional blockchains, here we have an *interlink* structure which can potentially be adversarially defined, it is possible that this collection does not form a blocktree, but a blockDAG with diamond-shaped topologies. This blockDAG is maintained in the form of a hashmap in the blockById data structure. Using this, the set of ancestors of a block present in any of the proofs can be extracted, which is done in the ancestors method. In the final predicate evaluation, the set of ancestors of the best blockchain tip is passed to the predicate.

7.2 Security

Theorem 7.2. *The infix NIPoPoW construction is secure for all infix-sensitive stable predicates Q, except with negligible probability in κ.*

Proof. Assume a typical execution. To prove the security of the construction, it suffices to show that if all honest verifiers agree on the value of $Q(\mathcal{C})$, then the verifier will output the same value. Assume that all verifiers agree on the value v.

By Theorem 6.7 and because the evaluation of $\tilde{\pi}$ is identical in the suffix-sensitive and in the infix-sensitive case, we deduce that $b = \tilde{\pi}[-1]$ will be an honestly adopted block. Furthermore, due to the Common Prefix property of backbone, b will belong to all honest parties' chains and in the same position, as it is buried under $|\tilde{\chi}| = k$ blocks.

By assumption, at least one honest party B has provided an honest NIPoPoW π_B. Let the chain adopted by that honest party during the round in which the NIPoPoW was produced be $\mathcal{C}$. Because the predicate Q is infix-sensitive and stable, this means that $\exists \mathcal{C}' \subseteq \mathcal{C}[:-k] : P_v(\mathcal{C}')$. Due to B being honest, $\mathcal{C}' \subseteq \pi_B$. Let $S = \mathsf{ancestors}(b)$ be the ancestors evaluated by the verifier. Clearly $\mathcal{C}' \subseteq S$ and therefore $Q(S) = Q(\mathcal{C}') = v$. □

7.3 Succinctness

As long as the number of blocks on which the predicate depends is polylogarithmic with respect to the chain length, our proofs remain succinct. Specifically, the proof size for the suffix has exactly the same size. Then the part of the proof that is of interest is the output of the followDown algorithm. However, notice that this algorithm will on average produce as many blocks as the difference of levels between B' and E, which is at most logarithmic in the chain size. Hence the proof sizes will be in expectation $(m + |\mathcal{C}'|)\log(|\mathcal{C}|)$, which remains succinct if $|\mathcal{C}'| \in O(polylog(|\mathcal{C}|))$.

8 GRADUAL DEPLOYMENT PATHS

Our construction requires an upgrade to the cryptocurrency consensus layer. We envision that new cryptocurrencies will adopt our construction in order to support efficient light clients.

However, existing cryptocurrencies could also benefit by adopting our construction as an upgrade. In this section we outline several possible upgrade paths. We also contribute a

novel upgrade approach, called a "velvet fork," which allows for gradual deployment without harming unupgraded nodes or miners at all.

8.1 Hard Forks and Soft Forks

First of all, keep in mind that the interlink data structure will be useful to be included in the form of a Merkle tree of interlink pointers, as discussed in the next section on implementation optimizations.

The most obvious way to upgrade a cryptocurrency to support our protocol is a hard fork: the block header could be modified to include our additional interlink structure, and the validation rules would be modified to require that new blocks (after a "flag day") contain a correctly-formed interlink hash.

The safety of the hard fork approach is often debated: it is not "forward compatible," in the sense that nodes failing to upgrade in time would not be able to process newly created blocks. Hard forks are also associated with "schisms": in one notable case, Ethereum applied a hard fork in order to reverse the effects of an exploited vulnerability; this decision was controversial, and dissenters splintered off into a new network, Ethereum Classic

A soft fork construction in practice would require including the interlink data structure not in the block header, but in the data of the coinbase transaction. Clearly it is enough to store a single hash of the whole interlink structure in the coinbase data. The only requirement for the PoPoWs to work is that the PoW commits to all the pointers within interlink so that the adversary cannot cause a chain reorganization except with negligible probability. That way, each superchain is really a chain which goes all the way back to genesis. If we take that route, then each PoPoW will be required to present not only the block header, but also a proof-of-inclusion path within the Merkle tree of transactions proving that the coinbase transaction is indeed part of the block. Once that is established, the coinbase data can be presented, and the verifier will thereby know that the hash of the interlink data structure is correct. Given the fact that in the current Bitcoin implementation there is a fixed limit for block sizes, we observe that including such proofs-of-inclusion will only increase the PoPoW sizes by a constant factor per block, allowing for the communication complexity to remain at $\Theta(polylog(|\mathcal{C}|))$.

8.2 Velvet Forks

We now describe a novel upgrade path that avoids the need for a fork at all. The key idea is that clients can make use of our scheme, even if only some blocks in the blockchain include the interlink structure. Given that intuitively these changes constitute rule modifications to the consensus layer, we call this technique a *velvet fork*.

The way to achieve this is by requiring upgraded miners to include the interlink data structure in the form of a new Merkle tree root hash in their coinbase data, similar to a soft fork. An unupgraded miner will as usual ignore this data as comments. However, we further require the upgraded miners to accept all previously accepted blocks, regardless of whether they have included the interlink data structure or not. In fact, even if the interlink data structure is included and contains invalid data, we require the upgraded miners to accept their containing blocks. Malformed interlink data could be simply of the wrong format, for example pointers of particular level appearing in the wrong position in the Merkle tree, or the pointers could be pointing to superblocks of incorrect levels. Furthermore, the pointers could be pointing to superblocks of the correct level, but not to the most recent block. By requiring upgraded miners to accept all such blocks, we do not modify the set of accepted blocks: Upgraded and unupgraded miners accept the exact same set of blocks. Therefore, the upgrade is simply a "recommendation" for miners and not an actual change in the consensus rules. Hence, while a hard fork makes new upgraded blocks invalid to unupgraded clients and a soft fork makes new unupgraded blocks invalid to upgraded clients, the velvet fork has the effect that blocks produced by either upgraded or unupgraded clients are valid for either. Hence, the blockchain is never forked. Only the codebase is upgraded, and the existing data on the blockchain is interpreted differently.

The reason this can work is because provers and verifiers of our protocol can check the validity of the claims of miners who make false interlink chain claims. An upgraded prover can check whether a block contains correct interlink data and use it. If a block does not contain correct interlink data, the prover can opt not to use those pointers at all in their proofs. As the Verifier verifies all claims of the prover, adversarial miners cannot cause harm by including invalid interlink data. The only thing that the Verifier cannot verify in terms of interlink claims is whether the claimed superblock of a given level is in fact the most recent previous superblock of that level. However, an adversarial prover cannot make use of that to construct winning proofs, as they are only able to present shorter chains in that case. The honest prover can simply ignore such pointers as if they were not included at all.

The velvet prover works as usual, but additionally maintains a *realLink* data structure as part of its full node, which stores the correct interlink for each block. Whenever a new winning chain is received from the network, the prover checks it for blocks that it hasn't seen before. For those blocks, it maintains its own realLink data structure which it updates accordingly to make sure it is correct regardless of what the interlink data structure of the received block claims.

The velvet constructInnerChain is implemented identically as before, except that instead of following the interlink pointer blindly it now calls the helper function called *followUp*. The followUp algorithm is shown in Algorithm 9. It accepts a block B and a level μ and creates a connection from B back to the most recent preceding level-μ block. It does this by following the interlink pointer if it is available and correct. Otherwise, it follows the previd link which is available in unupgraded blocks, and tries to follow the interlink pointer again from there. Finally, the velvet prover simply calls the

velvet constructInnerChain function and includes the auxiliary connecting nodes within the final proof. No changes in the verifier are needed.

Algorithm 9 The followUp function which produces the necessary blocks to connect two superblocks in velvet forks.

```
1: function followUp(B, μ, realLink, blockById)
2:     aux ← [B]
3:     while B ≠ Gen do
4:         if B.interlink[μ] = realLink[id(B)][μ] then
5:             id ← B.interlink[μ]
6:         else                                   ▷ Invalid interlink
7:             id ← B.interlink[0]
8:         end if
9:         B ← blockById[id]
10:        aux ← aux ∪ B
11:        if level(B) = μ then
12:            return B, aux
13:        end if
14:    end while
15:    return B, aux
16: end function
```

It is straight-forward to see that velvet NIPoPoWs preserve security. But observe that if a constant minority of miners has upgraded their nodes to add NIPoPoW support, then succinctness is also preserved, as there is only a constant factor penalty as proven in the following theorem.

THEOREM 8.1. *Velvet non-interactive proofs-of-proof-of-work on honest chains by honest provers remain succinct as long as a constant percentage g of miners has upgraded, with overwhelming probability.*

The reason we were able to upgrade using a velvet fork was because the changes we made were helpful but verifiable by those looking at the chain. Note that we would not have been able to pull off this upgrade without modifications to the consensus layer in the sense that the interlink data structure could not have been maintained somewhere independently of the blockchain: It is critical that the proof-of-work commits to the interlink data structure. However, interestingly, the interlink data structure does not even need to be part of the coinbase transaction and could even be produced and included in regular transactions by users (such as OP_RETURN transactions). In that manner, the miners can be completely oblivious to it, while users and provers make use of the feature. The way to realize this would be to ask users to regularly create transactions containing the most recent interlink pointers so that they are included in the next block. If the transaction makes it to the next block, it can be used by the prover who keeps track of these. Otherwise, if it becomes part of a subsequent block, in which case some of the pointers it contains are invalid, it can be simply be ignored or only partially used.

Table 1: Estimated size of NIPoPoWs, applied to the Bitcoin blockchain today (≈450k blocks) for various values of the security parameter m setting $k = 6$.

m	NIPoPoW size	Blocks	Hashes
6	70 kB	108	1440
15	146 kB	231	2925
30	270 kB	426	5400
50	412 kB	656	8250
100	750 kB	1206	15000
127	952 kB	1530	19050

Supporting clients with different beliefs. We note that the interlink format does not depend at all on the parameters m and k. Therefore, regardless of the upgrade procedure, it is not necessary to reach consensus agreement on a particular choice of these parameters. Instead, the choice of m and k can be exposed as a user-configurable parameter to clients.

9 IMPLEMENTATION & PARAMETERS

Interlink Optimizations. We now discuss the exact manner in which the interlink data structure should be hashed to provide a single hash within the coinbase transaction. A Merkle tree should be used to hash the interlink data structure into a single hash

By organizing the interlink pointers into a Merkle tree of $\log(|\mathcal{C}|)$ items, a proof-of-inclusion in the pointers Merkle tree can be provided in $\log\log(|\mathcal{C}|)$ space, noting that the 0-level pointer need not be included in this Merkle tree, but the genesis block needs to be. Then the root of the pointers Merkle tree can be proved to have been included in the block header in $\log(|\overline{x}|)$ using the standard Merkle tree of transactions, where $\overline{x}$ is the vector of all transactions in the given block. This makes the proof size require $\log(|\overline{x}|) + \log\log(|\mathcal{C}|)$ hashes per block for a total of $m(\log(|\mathcal{C}|) - \log(m))(\log(|\overline{x}|) + \log\log(|\mathcal{C}|))$ hashes. In addition, $m(\log(|\mathcal{C}|) - \log(m))$ block headers and coinbase transactions are needed. More concretely, given that currently $|\mathcal{C}| = 464,185$, we have that $\log(|\mathcal{C}|) = 18$ and $\log\log(|\mathcal{C}|) = 5$. Typically, $|\overline{x}| = 2000$ in the current bitcoin setting, which makes $\log(|\overline{x}|) = 11$. For the k-suffix, only k block headers need to be included. We set $k = 6$ as is traditional in the bitcoin setting and see that block headers are 80 bytes and hash outputs are 32 bytes. However, we note that block headers do not need to include the previous block header hash for the k-suffix, as each previous block header hash can be calculated by the Verifier, limiting header sizes to 48 bytes for the k last blocks. Furthermore, the k-suffix does not require the presentation of the coinbase transaction data. These are also true for the $2m$ blocks in $\overline{\Pi}[0]$. Note also that for the proof prefix $\overline{\Pi}$, the coinbase transaction hash need not be included: It can be evaluated by the Verifier when building the Merkle tree during the proof-of-inclusion. Similarly, the root of the pointers Merkle tree can be omitted from the coinbase transaction data when transmitting the proof. Additionally, the leaf of the pointers Merkle tree can

also be omitted, as it can be obtained by the Verifier by simply hashing the previous block in the proof. In fact, no block ids need to be transmitted at all, as they can all be built from the Verifier starting from the knowledge of Genesis.

From these observations, we estimate our scheme's proof sizes for various parameterizations of m in Table 1.

9.1 Choosing Secure Parameters

In order to determine concrete values for the security parameter m, we now turn our attention to a particular adversarial strategy of interest and analyze its probability of success under specific conditions. It is important to keep in mind that this adversarial strategy is only one possible strategy and an advanced adversary can perform more sophisticated attacks. However, the fact that this attack is reasonable and possible to simulate allows us to extract specific values for m. The attack is an extension of the stochastic processes originally described in

The experiment works as follows: Initially, the value m is fixed and some adversarial computational power percentage q over the total network computational power is chosen. k is chosen based on q according to Nakamoto

Based on the above experiment, we measure the probability of success of the adversary. We experiment with various values of m for $y = 100$, indicating 100 blocks of secret parallel mining. We make the simplifying assumption that honest party communication is perfect and immediate, meaning that all honest party rounds that are successful are also uniquely successful. We ran $1,000,000$ Monte Carlo executions [3] of the experiment for each value of m from 1 to 30. We ran the simulation for values of computational power percentage $q = 0.1$, $q = 0.2$ and $q = 0.3$. The results are plotted in Figure 4.

Based on this data, we conclude that $m = 5$ is sufficient to achieve a 0.001 probability of failure against an adversary with 10% mining power. To secure against a powerful adversary who controls more than 30% of the network's mining power, a choice of $m = 15$ is needed.

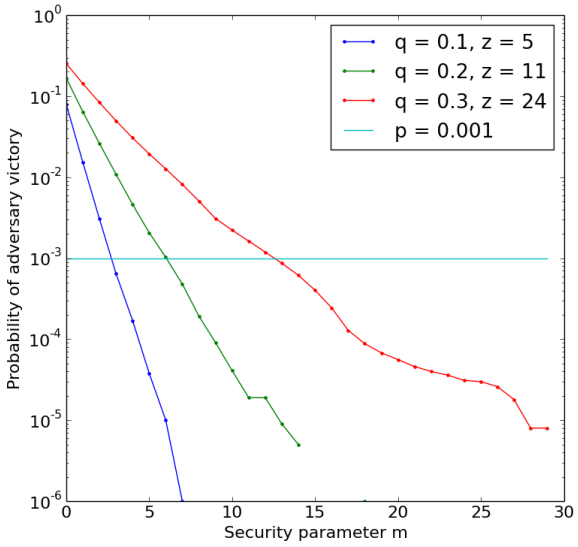
10 APPLICATIONS AND EVALUATION

10.1 Multi-blockchain wallets.

One potential application of our technique is an efficient multi-cryptocurrency client. Consider for example a merchant that wishes to accept payments in any cryptocurrency, not just the most popular ones. A naïve approach would be for the merchant to simply install an SPV client for each known cryptocurrency. However, this approach would entail downloading the header chain for each cryptocurrency, and periodically (e.g., each day) syncing up by fetching any newly generated block headers. Another alternative would be to use an online service supporting multiple currencies, such as an exchange, although this introduces reliance on a third party. Similarly, existing multi-cryptocurrency apps, such as

[3]The URL to the GitHub repository of this MIT-licensed experiment has been redacted for anonymity and will be provided in the proceedings version of this paper.

Figure 4: Simulation results for a private mining attacker with $q \in \{0.1, 0.2, 0.3\}$ computational power, suffix security k according to Nakamoto and parallel mining parameter $y = 100$. Probabilities are plotted on a logarithmic scale. The threshold probability of



Jaxx and Coinomi rely on third party networks rather than peer-to-peer networks.

A multi-blockchain wallet based on NIPoPoWs. The NIPoPoW-based client maintains a most recent k-stable block hash for each of its supported cryptocurrencies, initially the genesis block for each. Each time a payment is received, the client connects to peers on the corresponding network and asks for a NIPoPoW proof relative to the most recently stored block hash. For cryptocurrencies where payments are received very frequently, the NIPoPoW-based client might download nearly every block header, just like an ordinary SPV client; however, for cryptocurrencies used infrequently, the NIPoPoW-based client would be able to skip over many of the blocks.

Simulation. We developed a simulation to evaluate the potential resources savings resulting from the use of a NIPoPoW-based client. We model the arrival of payments in each cryptocurrency as a Poisson process, and assume that the market capitalization of a cryptocurrency is a proxy for usage, i.e. in our model most payments received are Bitcoin transactions, with transactions from the remaining constituting a long tail.

At the current time of writing, a total of 731 cryptocurrencies are listed on coin market directories[4]. This list includes many defunct cryptocurrencies that are no longer functioning at all. We narrow our focus to the 80 cryptocurrencies that have their own proof-of-work blockchains (i.e., no proof of stake) and that have a market cap of over USD $100,000.

[4]https://coinmarketcap.com/

Table 2: Cost of header chains for all active Proof-of-Work-based cryptocurrencies (collected from coinwarz.com)

Hash	Coins	Size today	Growth rate
Scrypt	44	4.3 GB	5.5 MB / day
SHA-256	15	1.4 GB	937.0 kB / day
X11	5	581.1 MB	556.3 kB / day
Quark	3	647.9 MB	518.4 kB / day
CryptoNight	2	199.0 MB	115.2 kB / day
EtHash	2	588.6 MB	921.6 kB / day
Groestl	2	300.3 MB	184.2 kB / day
NeoScrypt	2	310.6 MB	153.6 kB / day
Equihash	2	17.7 MB	92.2 kB / day
Keccak	1	161.1 MB	115.2 kB / day
X13	1	30.0 MB	57.6 kB / day
Lyra2REv2	1	57.4 MB	46.1 kB / day
Total	80	8.5 GB	9.2 MB / day

Table 3: Simulated bandwidth of multi-blockchain clients after two months (Averaged over 10 trials each)

Daily	Naive SPV		NIPoPoW		
tx	Total	(Daily)	Total	(Daily)	Savings
100	5.5 GB	(5.5 MB)	31.7 MB	(507 kB)	99% (91%)
500	5.5 GB	(5.7 MB)	68.2 MB	(1.1 MB)	99% (81%)
1000	5.5 GB	(6.0 MB)	99.1 MB	(1.6 MB)	98% (73%)
3000	5.6 GB	(7.0 MB)	192 MB	(3.1 MB)	97% (56%)

In Table 2 we show aggregate statistics about these 80 cryptocurrencies, grouped according to the their proof-of-work puzzle. Bitcoin's 10 minute block time is relatively slow; other cryptocurrencies typically use faster block times, e.g. 12 seconds for Ethereum and 2 minutes for Litecoin. Thus while the entire chain of proof-of-work puzzles in Bitcoin today only amounts to about 40 MB, taken together, the 80 cryptocurrencies comprise nearly 10 GB of proofs-of-work, and generate nearly 10 MB more each day.

In our simulation we choose NIPoPoW parameters based as in the previous section, with $m = 24$ and $k = 6$. In Table 3 we show the resulting bandwidth costs from simulating a period of 60 days, with varying rates of payments received.

For the naïve SPV client, the total bandwidth cost is dominated by the need to fetch the entire chain of block headers, which the NIPoPoW client does not need to do. The marginal cost for naïve SPV depends on the number of blocks generated each day, as well as the transaction inclusion proofs associated with each payment. The NIPoPoW based client provides the most savings when the number of transactions per day is smallest, reducing the necessary bandwidth per day (not including the initial sync up) by 90%.

10.2 Certificate Transparency

In Catena

10.3 Sidechains

At the time of writing, it is widely known that Bitcoin faces significant scaling hurdles

In the following, we explain sidechains in an incremental fashion, by first providing some context on other protocols involving cross-blockchain interactions. The most well known is the "atomic swap"

As a more complicated example, BTCRelay is an Ethereum

Building on this idea, the sidechains protocol can be seen as a pair of blockchains, the sidechain and the parentchain, that each act as an SPV client of each other. It is then straightforward to cross-chain applications, such as a virtual asset on the sidechain that is backed by a quantity of currency on the parent chain (i.e., a two-way peg). This is accomplished by defining two new transaction types: "deposit" transactions, which have the effect of locking coins on the parent chain and creating new assets on the side chain, as well as "withdrawal" transactions that have the opposite effect, unlocking coins on the parent chain and destroying coins on the sidechain. Deposit transactions are committed first on the parent chain, and then delivered to the sidechain using an SPV proof; vice versa for withdrawal transactions. Note that the parent chain would not need to know about most transactions on the sidechain except for the withdrawal transactions — hence this provides a potential scalability solution if significant transaction volume can be delegated to sidechains.

In the simplest case, the naïve SPV approach would work; each blockchain header of the parent chain would be submitted in a transaction payload to the sidechain (c.f. Back et al.

11 FUTURE WORK

Our security proof was constructed in the Backbone

We worked with a construction that allows a certain class of chain predicates to be proved: In particular, stable predicates which are functions only of the suffix of the chain or depend on a small segment within the chain. However, slightly more general predicates can also be proved succinctly. We leave the complete characeterization of succinctly provable predicates for future work.

We have constructed Proofs-of-Proofs-of-Work. An open question remains whether the same problem of Proof-of-Proofs is solvable in Proof-of-Stake settings, a Proof-of-Proof-of-Stake (PoPoS) protocol. It is unknown whether this problem can be solved interactively or non-interactively.

Finally, while our construction hints at a direction of sidechains, no formalism exists that allows us to reason about whether NIPoPoW protocols are appropriate as sidechain solutions. A proper security definition for the desirable properties of sidechains would allow this evaluation to take place.

Algorithm 10 The backbone protocol

$\mathcal{C} \leftarrow \varepsilon$
$st \leftarrow \varepsilon$
$round \leftarrow 1$
while TRUE **do**
  $\tilde{\mathcal{C}} \leftarrow \mathsf{maxvalid}(\mathcal{C}, \text{any chain } \mathcal{C}' \text{ found in } \text{RECEIVE}())$
  if INPUT() contains READ **then**
    write $R(\mathcal{C})$ to OUTPUT()
  end if
  $\langle st, x\rangle \leftarrow I(st, \tilde{\mathcal{C}}, round, \text{INPUT}(), \text{RECEIVE}())$
  $\mathcal{C}_{\mathsf{new}} \leftarrow \mathsf{pow}(x, \tilde{\mathcal{C}})$
  if $\mathcal{C} \neq \mathcal{C}_{\mathsf{new}}$ **then**
    $\mathcal{C} \leftarrow \mathcal{C}_{\mathsf{new}}$
    DIFFUSE(*broadcast*)
  else
    DIFFUSE($\perp$)
  end if
  $round \leftarrow round + 1$
end while

Algorithm 11 Our generic honest Prover entity addition to the backbone model. It is augmented with a *proof generating function* $\mathsf{Prove}^{Q}(\cdot)$ parameterized by a predicate Q

$\mathcal{C} \leftarrow \varepsilon$
while TRUE **do**
  $\tilde{\mathcal{C}} \leftarrow \mathsf{maxvalid}(\mathcal{C}, \text{any chain } \mathcal{C}' \text{ found in } \text{RECEIVE}())$
  if INPUT() contains READPROOF **then**
    $\pi \leftarrow \mathsf{Prove}^{Q}(\mathcal{C})$
    DIFFUSE(π)
  else
    DIFFUSE($\perp$)
  end if
end while

Algorithm 12 The generic proof verifier augmented with a *proof verification function* $\mathsf{Verify}^{Q}(\cdot)$ parameterized by a predicate Q

$\Pi \leftarrow$ all proofs found in RECEIVE()
if $\Pi \neq \emptyset$ **then**
  $\mathcal{C} \leftarrow \mathsf{Verify}^{Q}(\Pi)$
  write $\mathcal{C}$ to OUTPUT()
end if
if INPUT() contains CHALLENGE **then**
  DIFFUSE(READPROOF)
end if

A BACKBONE INTEGRATION

In this section, we illustrate the detailed formalisms to integrate our proof systems with the backbone model

The backbone protocol of a miner node is shown in Algorithm 10. The honest miner maintains the longest chain from the network and tries to mine on top of it. In Algorithm 11, we illustrate our new entity for the model, the full node or *prover*. While the full node is not mining, it maintains a state with the longest chain from the network. Furthermore, whenever it is asked to prove a predicate Q about its local chain $\mathcal{C}$, it calls a Prove function to provide the proof. We leave this Prove function undefined here, as it is part of the concrete protocol construction. On the other hand, in Algorithm 12 we illustrate another new entity to the model, the generic *verifier*, which is stateless, but receives proofs from the provers on the network (via the environment) and takes a decision about a predicate on the blockchain it believes to be the longest.

The prover which additionally maintains the *blockById*, *depth* and *realLink* data structures is illustrated in Algorithm 13.

B FULL PROOFS

THEOREM 3.4 (PREDICATE PERSISTENCE). *If a chain predicate is monotonic, then it satisfies persistence with parameter* $k = 2\eta\kappa f$ *with probability at least* $1 - e^{-\Omega(\kappa)}$.

PROOF. Assume a typical execution as defined in

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 $b = \mathsf{LCA}(\pi_{\mathcal{A}}, \pi_B)$ and let μ'_B be the adequate level of π_B with respect to b. Then $\mu'_B \geq \mu_B$. [Local goodness] Assume chain $\mathcal{C}$ contains only honestly-generated blocks and has been adopted by an honest party in an execution with random network scheduling. Then for all levels μ, for all constant $\delta > 0$, all continuous subchains $\mathcal{C}' = \mathcal{C}[i:j]$ with $|\mathcal{C}'| \geq m$ are locally good, $\mathsf{local\text{-}good}_{\delta}(\mathcal{C}', \mathcal{C}, \mu)$, with overwhelming probability in m.

PROOF. **(Sketch)** Observing that for each honestly generated block the probability of being a μ-superblock for any level μ follows an independent Bernoulli distribution, we can apply a Chernoff bound to show that the number of superblocks within a chain will be close to its expectation, which is what is required for local goodness. □

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 $b = \mathsf{LCA}(\pi_{\mathcal{A}}, \pi_B)$ and let μ'_B be the adequate level of π_B with respect to b. Then $\mu'_B \geq \mu_B$. [Superquality] For all levels μ, for all constant $\delta > 0$, a chain $\mathcal{C}$ containing only honestly-generated blocks adopted by an honest party in an execution with random network scheduling has (δ, m)-superquality at level μ with overwhelming probability in m.

Algorithm 13 The modified prover that allows for a velvet fork.

```
1: C ← [Gen]
2: realLink ← {}
3: blockById ← {}
4: genId ← id(Gen)
5: blockById[genId] ← Gen
6: realLink[genId] ← [Gen]
7: while TRUE do
8:     C̃ ← maxvalid(C, any chain C′ found in RECEIVE())
9:     new ← []
10:    for i = |C| − 1 down to 0 do
11:        if C[i] ∈ realLink then
12:            B′ ← C[i]
13:            interlink ← realLink[B′]
14:            break
15:        end if
16:        new.append(C[i])
17:    end for
18:    for B ∈ new do
19:        for μ = 0 to level(B′) do
20:            interlink[μ] ← blockid(B′)
21:        end for
22:        B′ ← B
23:        blockById[blockid(B)] ← B
24:        realLink[B] ← interlink
25:    end for
26:    if INPUT() contains READPROOF then
27:        Π̃ ← Prove′_q(C, realLink, blockById)
28:        DIFFUSE(Π̃)
29:    else
30:        DIFFUSE(⊥)
31:    end if
32: end while
```

PROOF. Let $\mathcal{C}' = \mathcal{C}\uparrow^{\mu}$ and let $\mathcal{C}^* = \mathcal{C}'[-m' :]$ for some $m' \geq m$. Then let $B \in \mathcal{C}^*\downarrow$ and let X_B be the random variable equal to 1 if $level(B) \geq \mu$ and 0 otherwise. $\{X_B : B \in \mathcal{C}^*\}$ are mutually independent Bernoulli random variables with expectation $E(X_B) = 2^{-\mu}|\mathcal{C}^*\downarrow|$. Let $X = \sum_{B\in\mathcal{C}^*\downarrow} X_B$. Then X follows a Binomial distribution with parameters $(m', 2^{-\mu})$ and note that $|\mathcal{C}^*| = X$. Then $\mathbb{E}(|\mathcal{C}^*\downarrow|) = 2^{-\mu}|\mathcal{C}^*|$. Applying a Chernoff bound on $|\mathcal{C}^*\downarrow|$ we obtain that:

$$\Pr[|\mathcal{C}^*\downarrow| \leq (1-\delta)2^{-\mu}|\mathcal{C}^*\downarrow|] \leq \exp(-\delta^2 2^{-\mu-1}|\mathcal{C}^*|)$$

□

THEOREM 5.1 (DOUBLE-SPENDING ATTACK). *There exist parameters p, n, t, q, μ, δ, with $t \leq (1-\delta)(n-t)$, and a double spending attack against KLS PoPoW that succeeds with overwhelming probability.*

PROOF. Recall that in the backbone notation n denotes the total number of parties, t denotes the number of adversarial parties, q denotes the number of the random oracle queries allowed per party per round and p is the probability that one random oracle query will be successful and remember that $p = T/2^{\kappa}$ where T is the mining target and κ is the security parameter (or hash function bit count). Then f denotes the probability that a given round is successful and we have that $f = 1-(1-p)^{q(n-t)}$. Recall further that a requirement of the backbone protocol is that the honest majority assumption is satisfied, that is that $t \leq (1-\delta)(n-t)$ were $\delta \geq 2f + 3\epsilon$, where $\epsilon \in (0,1)$ is an arbitrary small constant describing the quality of the concentration of the random variables.

Denote $\alpha_{\mathcal{A}}$ the secret chain generated by the adversary and α_B the honest chain belonging to any honest party. We will show that for certain protocol values we have that $\Pr[|\alpha_{\mathcal{A}}\uparrow^{\mu}| \geq |\alpha_B\uparrow^{\mu}|]$ is overwhelming.

Assume that, to the adversary's harm and to simplify the analysis, the adversary plays at beginning of every round and does not perform adversarial scheduling. At the beginning of the round when it is the adversary's turn to play, she has access to the blocks diffused during the previous round by the honest parties.

First, observe that at the beginning of each round, the adversary finds herself in one of two different situations: Either she has been forced into an r-round-long period of suppression, or she is not in that period. If she is within that period, she blindly performs the suppression attack without regard for the state of the world. If she is not within that period, then she must initially observe the blocks diffused at the end of the previous round by the honest parties. Call these rounds during which the diffused data must be examined by the adversary *decision rounds*. Let there be ω decision rounds in total. In each such decision round, it is possible that the adversary discovers a diffused μ-superblock and therefore decides that a suppression attack must be performed starting with the current round. Call these rounds during which this discovery is made by the adversary *migration rounds*. Let there be y migration rounds in total. The adversary devotes the migration round to performing the suppression attack as well as $r-1$ non-migration rounds after the migration round. Call these rounds, including the migration round, *suppression rounds*. In the rest of the decision rounds, the adversary will not find any μ-superblocks diffused. Call these *secret chain rounds*; these are rounds where the adversary devotes her queries to mining on the secret chain. Let there be x secret chain rounds. If the adversary devotes ω decision rounds to the attack in total, then clearly we have that $\omega = x + y$. If the total number of rounds during which the attack is running is s then we also have that $s = x + ry$, because for each migration round there are $r-1$ non-decision rounds that follow.

We will analyze the honest and adversarial superchain lengths with respect to ω, which roughly corresponds to time (because note that $\omega \geq s/r$, and so ω is proportional to the number of rounds).

Let us calculate the probability p_{SB} ("superblock probability") that a decision round ends up being a migration round. Ignoring the negligible event that there will be random oracle collisions, we have that $p_{SB} = (n-t)qp2^{-\mu}$.

Given this, note that the decision taken at the beginning of each decision round follows independent Bernoulli distributions with probability p_{SB}. Denote z_i the indicator random variable indicating whether the decision round was a migration round. Therefore we can readily calculate the expectations for the random variables x and y, as $x = \omega - y$, $y = \sum_{i=1}^{\omega} z_i$. We have $E[x] = (1-p_{SB})\omega$ and $E[y] = p_{SB}\omega$. Applying a Chernoff bound to the random variables x and y, we observe that they will attain values close to their mean for large ω and in particular $\Pr[y \geq (1+\delta)E[y]] \leq \exp(-\frac{\delta^2}{3}E[y])$ and similarly $\Pr[x \leq (1-\delta)E[x]] \leq \exp(-\frac{\delta^2}{2}E[x])$, which are negligible in ω.

Given that there will be x secret chain rounds, we observe that the random variable indicating the length of the secret adversarial superchain follows the binomial distribution with xtq repetitions and probability $p2^{-\mu}$. We can now calculate the expected secret chain length as $E[|\alpha_{\mathcal{A}}\uparrow^{\mu}|] = xtqp2^{-\mu}$. Observe that in this probability we have given the adversary the intelligence to continue using her random oracle queries during a round even after a block has been found during a round and not to wait for the next round. Applying a Chernoff bound, we obtain that $\Pr[|\alpha_{\mathcal{A}}\uparrow^{\mu}| \leq (1-\delta)E[|\alpha_{\mathcal{A}}\uparrow^{\mu}|]] \leq exp(-\frac{\delta^2}{2}E[|\alpha_{\mathcal{A}}\uparrow^{\mu}|])$, which is negligible in ω (because we know that with overwhelming probability $x > (1-\delta)(1-p_{SB})\omega$).

It remains to calculate the behavior of the honest superchain.

Suppose that a migration round occurs during which at least one superblock B is diffused. We will now calculate the probability p_{sup} that the adversary is able to suppress that block after r rounds by performing the suppression attack and cause all honest parties to adopt a chain not containing B.

One way for this to occur is if the adversary has generated exactly 2 shallow blocks (blocks which are not μ-superblocks) after exactly r rounds and the honest parties having generated exactly 0 blocks after exactly r rounds. This provides a lower bound for the probability, which is sufficient for our purposes. Call ADV-WIN the event where the adversary has generated exactly 2 shallow blocks after exactly r rounds since the diffusion of B and call HON-LOSE the event where the honest parties have generated exactly 0 blocks after exactly r rounds since the diffusion of B.

The number of blocks generated by the adversary after the diffusion of B follows the binomial distribution with r repetitions and probability p_{LB}, where p_{LB} denotes the probability that the adversary is able to produce a shallow block ("low block probability") during a single round. We have that $p_{LB} = tqp(1-2^{-\mu})$. To evaluate Pr[ADV-WIN], we evaluate the binomial distribution for 2 successes to obtain $\Pr[\text{ADV-WIN}] = \frac{r(r-1)}{2}p_{LB}^2(1-p_{LB})^{r-2}$. The number of blocks generated by the honest parties after the diffusion of B follows the binomial distribution with r repetitions and probability f. To evaluate Pr[HON-LOSE], we evaluate the binomial distribution for 0 successes to obtain $\Pr[\text{HON-LOSE}] = (1-f)^r$. Note that this is an upper bound in the probability, in particular because there can be multiple blocks during a non-uniquely successful round during which a μ-superblock was generated.

Then observe that the two events ADV-WIN and HON-LOSE are independent and therefore $p_{sup} = \Pr[\text{ADV-WIN}]\Pr[\text{HON-LOSE}] = \frac{r(r-1)}{2}p_{LB}^2(1-p_{LB})^{r-2}(1-f)^r$.

Now that we have evaluated p_{sup}, we will calculate the honest chain length in two chunks: The superblocks generated and adopted by the honest parties during secret chain rounds, $\mathcal{C}_1$, and the superblocks generated and adopted by the honest parties during suppression rounds, $\mathcal{C}_2$ (and note that these sets of blocks are not blockchains on their own).

$|\mathcal{C}_1|$ is a random variable following the binomial distribution with $s(n-t)q$ repetitions and probability $p2^{-\mu}(1-p_{sup})$. In the evaluation of this distribution, we give the honest parties the liberty to belong to a mining pool and share mining information within a round, an assumption which only makes matters for the adversary worse. We can now calculate the expected length of $\mathcal{C}_1$ to find $E[|\mathcal{C}_1|] = s(n-t)qp2^{-\mu}(1-p_{sup})$. Applying a Chernoff bound, we find that $\Pr[|\mathcal{C}_1| \geq (1+\delta)E[|\mathcal{C}_1|]] \leq exp(-\frac{\delta^2}{3}E[|\mathcal{C}_1|])$, which is negligible in s.

Finally, some additional μ-superblocks could have been generated by the honest parties while the adversary is spending r rounds attempting to suppress a previous μ-superblock. These μ-superblocks will be adopted in the case the adversary fails to suppress the previous μ-superblock. As the adversary does not devote any decision rounds to these new μ-superblocks, they will never be suppressed if the previous μ-superblock is not suppressed. We collect these in the set $\mathcal{C}_2$. To calculate $|\mathcal{C}_2|$, observe that the number of unsuppressed μ-superblocks which caused an adversarial suppression period is $|\mathcal{C}_1|$. For each of these blocks, the honest parties spend r rounds attempting to form further μ-superblocks on top. The total number of such attemps is $r|\mathcal{C}_1|$. Therefore, the number of further honestly generated μ-superblocks attained during the $|\mathcal{C}_1|$ different r-round periods follows a binomial distribution with $|\mathcal{C}_1|rq(n-t)$ repetitions and probability $p2^{-\mu}$. Here we allow the honest parties to use repeated queries within a round even after a shallow success and to work in a pool to obtain an upper bound for the expectation. Therefore $E[|\mathcal{C}_2|] = |\mathcal{C}_1|rq(n-t)p2^{-\mu}$ and applying a Chernoff bound we obtain that $\Pr[|\mathcal{C}_2| \geq (1+\delta)E[|\mathcal{C}_2|]] \leq exp(-\frac{\delta}{3}E[|\mathcal{C}_2|])$, which is negligible in s and has a quadratic error term. We deduce that $|\mathcal{C}_2|$ will have a very small length compared to the rest of the honest chain, as it is a vanishing term in μ.

Concluding the calculation of the adversarial superchain, we get $E[|\alpha_B\uparrow^{\mu}|] = E[|\mathcal{C}_1|] + E[|\mathcal{C}_2|]$.

Finally, it remains to show that there exist values $p, n, t, q, r, \mu, \delta$ such that a $E[|\alpha_{\mathcal{A}}\uparrow^{\mu}|] \geq (1+\delta)E[|\alpha_B\uparrow^{\mu}|]$.

Using the values $p = 10^{-5}, q = 1, n = 1000, t = 489, \mu = 25, r = 200$, we observe that the honest majority assumption

is preserved. Replacing these values into the expectations formulae above, we obtain $E[|\alpha_{\mathcal{A}}\uparrow^{\mu}] \approx 1.457 * 10^{-10} * \omega$ and $E[|\alpha_B\uparrow^{\mu}] \approx 1.424 * 10^{-10} * \omega$, which result to a constant gap δ. Because the adversarial chain grows linearly in ω, the adversary only has to wait sufficient rounds for obtaining m blocks to create a valid proof. Therefore, for these values, the adversary will be able to generate a convincing PoPoW at some level μ which is longer than the honest parties' proof, even when the adversary does not have a longer underlying blockchain. □

THEOREM B.1 (TYPICALITY). *Executions are (ϵ, η)-typical with overwhelming probability in κ.*

PROOF. **Block counts and regularity.** For the blocks count and regularity, we refer the reader to

Round count. First, observe that $Z^{\mu}_{ijk} \sim \mathsf{Bern}(2^{-\mu}p)$ and these are jointly independent. Therefore $Z^{\mu}_{S} \sim \mathsf{Bin}(tq|S|, 2^{-\mu}p)$ and $|S| \sim \mathsf{NB}(Z_S, 2^{-\mu}p)$. So $\mathbb{E}(|S|) = 2^{\mu}\frac{Z_S}{pqt}$. Applying a tail bound to the negative binomial distribution, we obtain that $\Pr[|S| < (1-\epsilon)\mathbb{E}(|S|)] \in \Omega(\epsilon^2 m)$. □

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 $b = \mathsf{LCA}(\pi_{\mathcal{A}}, \pi_B)$ and let μ'_B be the adequate level of π_B with respect to b. Then $\mu'_B \geq \mu_B$. Suppose S is a set of consecutive rounds $r_1 \dots r_2$ and $\mathcal{C}_B$ is a chain adopted by an honest party at round r_2 of a typical execution. Let $\mathcal{C}^S_B = \{b \in \mathcal{C}_B : b \text{ was generated during } S\}$. Let $\mu_{\mathcal{A}}, \mu_B \in \mathbb{N}$. Suppose $\mathcal{C}^S_B\uparrow^{\mu_B}$ is good. Suppose $\mathcal{C}'_{\mathcal{A}}$ is a $\mu_{\mathcal{A}}$-superchain containing only adversarially generated blocks generated during S and suppose that $|\mathcal{C}'_{\mathcal{A}}| \geq k$. Then:

$$2^{\mu_{\mathcal{A}}}|\mathcal{C}'_{\mathcal{A}}| < \frac{1}{3}2^{\mu_B}|\mathcal{C}^S_B\uparrow^{\mu_B}|$$

PROOF. From $|\mathcal{C}'_{\mathcal{A}}| \geq k$ we know that $Z_S \geq k$. Applying Theorem B.1, we conclude that $|S| \geq (1-\epsilon')2^{\mu_A}\frac{1}{pqt}|\mathcal{C}'_{\mathcal{A}}|$.

Applying the chain growth theorem

$$2^{\mu_{\mathcal{A}}}|\mathcal{C}'_{\mathcal{A}}| < \frac{pqt}{(1-\delta)(1-\epsilon')(1-\epsilon)f}2^{\mu_B}|\mathcal{C}^S_B\uparrow^{\mu_B}|$$

□

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 $b = \mathsf{LCA}(\pi_{\mathcal{A}}, \pi_B)$ and let μ'_B be the adequate level of π_B with respect to b. Then $\mu'_B \geq \mu_B$. Let π be some honest proof generated with security parameters δ, m. Let $\mathcal{C}$ be the underlying chain, $b \in \mathcal{C}$ be any block and μ' be the adequate level of the proof with respect to b and the same security parameters.

Then $\mathcal{C}\{b:\}\uparrow^{\mu'} = \pi\{b:\}\uparrow^{\mu'}$.

PROOF. $\pi\{b:\}\uparrow^{\mu'} \subseteq \mathcal{C}\{b:\}\uparrow^{\mu'}$ is trivial.

For the converse, we will show that for all $\mu^* > \mu'$, we have that in the iteration of the Prove `for` loop with $\mu = \mu^*$, the block stored in variable B preceeds block b in $\mathcal{C}$.

Suppose $\mu = \mu^*$ is the first `for` loop iteration during which the property is violated. Clearly this cannot be the first iteration, as there $B = \mathcal{C}[0]$ and Genesis preceeds all blocks, including itself. By the induction hypothesis we see that during the iteration $\mu = \mu^* + 1$, block B preceeded block b. But from the definition of μ' we know that μ' is the highest level for which $|\pi\{b:\}\uparrow^{\mu'}[1:]| \geq \max(m, (1-\delta)2^{-\mu'}|\pi\{b:\}\uparrow^{\mu'}[1:]\downarrow|)$.

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

In case $\mathsf{local\text{-}good}$ is violated, variable B remains unmodified and the induction step holds. If $\mathsf{local\text{-}good}$ is not violated, then $|\pi\{b:\}\uparrow^{\mu^*}[1:]| < m$ and so $\pi\uparrow^{\mu^*}[-m]$ preceeds b, and so we are done. □

THEOREM 6.7. *The non-interactive proofs-of-proof-of-work construction for k-stable suffix-sensitive predicates is secure with overwhelming probability in κ.*

PROOF. We will prove security by contradiction. Let $m = k_1 + k_2 + k_3$ and let k_1, k_2, k_3 be polynomial functions of the security parameter κ. Let Q be a k-stable suffix sensitive chain predicate. Assume the NIPoPoWs construction with parameters m, k is not secure with respect to Q. Then, during an execution at some round r_3, $Q(\mathcal{C})$ is defined and has the same value for all honest miners. Assume the execution is typical. The verifier V communicates with at least two provers, $\mathcal{A}$, the adversary, and B, an honest prover. The verifier receives the proofs $\pi_{\mathcal{A}}, \pi_B$ from the adversary and the honest prover respectively. Because B is honest, π_B will be a proof constructed based on an underlying blockchain $\mathcal{C}_B$ (with $\pi_B \subseteq \mathcal{C}_B$) which is the adopted blockchain of B during round r_3 at which π_B was generated. Furthermore, $\pi_{\mathcal{A}}$ will have been generated at some round $r'_3 \leq r_3$.

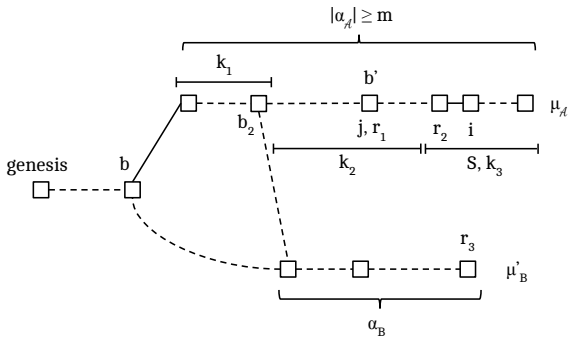
The verifier then outputs $\neg Q(\mathcal{C}_B)$, and so $\mathsf{Verify}^{Q}_{m,k} = \neg Q(\mathcal{C}_B)$. Thus it is necessary that $\pi_{\mathcal{A}} \geq \pi_B$, otherwise, because Q is suffix sensitive, Verify^{Q} would have returned $Q(\mathcal{C}_B)$. We will now show that $\pi_{\mathcal{A}} \geq \pi_B$ is a negligible event.

Let $b = \mathsf{LCA}(\pi_{\mathcal{A}}, \pi_B)$ and let b^* be the most recent honestly generated block in $\mathcal{C}_B$ preceeding b (and note that b^* necessarily exists because Genesis was 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. Call $\alpha_{\mathcal{A}} = \pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}\{b:\}$, $\alpha'_B = \pi_B\uparrow^{\mu'_B}\{b:\}$.

We will proceed in the proof by showing three successive claims: First, $\alpha_{\mathcal{A}}$ and $\alpha'_B\downarrow$ are mostly disjoint. Second, $\alpha_{\mathcal{A}}$ contains mostly adversarially-generated blocks. And third, the adversary is able to produce this $\alpha_{\mathcal{A}}$ with negligible probability.

Figure 5: Two competing proofs at different levels.



Claim 1a: If $\mu'_B \leq \mu_{\mathcal{A}}$ then $\alpha_{\mathcal{A}}[1:]$ and $\alpha_B[1:]\downarrow$ are completely disjoint.

Applying Lemma 6.6 to $\mathcal{C}_B\{b:\}\uparrow^{\mu'_B}$ we see that $\mathcal{C}_B\{b:\}\uparrow^{\mu'_B} = \pi_B\uparrow^{\mu'_B}\{b:\}$ and so indeed $\pi_B\uparrow^{\mu'_B}\{b:\}[1:] \cap \pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}\{b:\}[1:] = \emptyset$.

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

First, observe that, because the adversary is winning, therefore $|\alpha_{\mathcal{A}}| > 2^{\mu'_B-\mu_{\mathcal{A}}}m$.

Suppose for contradiction that $|\alpha_{\mathcal{A}}[1:] \cap \alpha_B\downarrow\ [1:]| > 2^{\mu'_B-\mu_{\mathcal{A}}}k_1$. This means that there are indices $1 \leq i < j$ such that $|\mathcal{C}_B\uparrow^{\mu_{\mathcal{A}}}[i:j]| > 2^{\mu'_B-\mu_{\mathcal{A}}}k_1$ but $|\mathcal{C}_B\uparrow^{\mu_{\mathcal{A}}}[i:j]\downarrow\uparrow^{\mu'_B}| = 0$. But this contradicts the goodness of $\mathcal{C}_B\uparrow^{\mu'_B}$. Therefore there are more than $2^{\mu'_B-\mu_{\mathcal{A}}}(k_2+k_3)$ blocks in $\alpha_{\mathcal{A}}$ that are not in α_B, and clearly also more than k_2+k_3 blocks.

From Claim 1a and Claim 1b, we conclude that there are at least k_2+k_3 blocks after block b in $\alpha_{\mathcal{A}}$ which do not exist in α_B. We now set $b_2 = \mathsf{LCA}(\mathcal{C}_B, \alpha_{\mathcal{A}})$.

Claim 2: At least k_3 superblocks of $\alpha_{\mathcal{A}}$ are adversarially generated.

We will show this by showing that $\alpha_{\mathcal{A}}[k_2+1:]$ contains no honestly mined blocks. By contradiction, assume that the block $\alpha_{\mathcal{A}}[i]$ for some $i \geq k_1+k_2+1$ was honestly generated. But this means that an honest party had adopted the chain $\alpha_{\mathcal{A}}[i-1]$ at some round $r_2 \leq r_3$. Because of the way the honest parties adopt chains, this means that the superchain $\alpha_{\mathcal{A}}[:i-1]$ has an underlying properly constructed 0-level anchored chain $\mathcal{C}_{\mathcal{A}}$ such that $\mathcal{C}_{\mathcal{A}} \subseteq \alpha_{\mathcal{A}}[:i-1]$. Let j be the index of block b_2 within $\mathcal{C}_{\mathcal{A}}$. As $\alpha_{\mathcal{A}} \subseteq \mathcal{C}_{\mathcal{A}}$, observe that $|\mathcal{C}_{\mathcal{A}}[j+1:]| > i-1 \geq k_2+k_1$. Therefore $\mathcal{C}_{\mathcal{A}}[:-(k_2+k_1)] \not\preccurlyeq \mathcal{C}_B$. But $\mathcal{C}_{\mathcal{A}}$ was adopted by an honest party at round r_2 which is prior to round r_3 during which $\mathcal{C}_B$ was adopted by an honest party. This contradicts the Common Prefix

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

Let b' be the latest honestly generated block in $\alpha_{\mathcal{A}}$, or b^* if no honest block exists in $\alpha_{\mathcal{A}}$. Let r_1 be the round during which b^* was generated. Let j be the index of block b^*. Consider now the set S of consecutive rounds $r_1 \ldots r_3$. Every block in $\alpha_{\mathcal{A}}[-k_3:]$ has been adversarially generated during S and $|\alpha_{\mathcal{A}}[-k_3:]| = k_3$. $\mathcal{C}_B$ 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 $\mathcal{C}_B^S$, we see that $\mathcal{C}_B^S = \mathcal{C}_B\{b^*:\}$. But chain $\mathcal{C}_B^S\uparrow^{\mu'_B}$ is good with respect to $\mathcal{C}_B^S$. Applying Lemma 6.4, we obtain that with overwhelming probability

$$2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}\{b':\}| < \frac{1}{3}2^{\mu'_B}|\mathcal{C}_B^S\uparrow \mu'_B|$$

But $|\alpha_B| \geq |\mathcal{C}_B^S\uparrow \mu'_B|$ and $|\alpha_{\mathcal{A}}\{b':\}| \geq |\alpha_{\mathcal{A}}| - k_2$, therefore:

$$2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}| - k_2 < \frac{1}{3}2^{\mu'_B}|\alpha_B|$$

But $|\alpha_{\mathcal{A}}| - k_2 \geq k_3$, therefore $\frac{1}{3}2^{\mu'_B}|\alpha_B| > k_3$ and so $2^{\mu'_B}|\alpha_B| > 3k_3$

Taking $k_2 = k_3$, we obtain:

$$2^{\mu_{\mathcal{A}}}|\alpha_{\mathcal{A}}| < \frac{1}{3}3k_3 + k_3 = 2k_3 < 2^{\mu'_B}|\alpha_B|$$

But this contradicts the fact that $\pi_{\mathcal{A}} \geq \pi_A$, and so the claim is proven.

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 $2^{\mu_B}|\pi_B\uparrow^{\mu_B}| \geq 2^{\mu'_B}|\pi_B\uparrow^{\mu'_B}|$, and therefore we conclude that $2^{\mu_B}|\pi_B\uparrow^{\mu_B}| > 2^{\mu_{\mathcal{A}}}|\pi_{\mathcal{A}}\uparrow^{\mu_{\mathcal{A}}}|$, which completes the proof. □

Theorem 6.8 (Number of levels). *The number of superblock levels which have at least m blocks are at most* $\log(|S|)$*, where S is the set of all blocks produced, with overwhelming probability in m.*

Proof. Let S be the set of all blocks successfully produced by the honest parties or the adversary. Because each block id is generated by the random oracle, the probability that it is less than $T2^{-\mu}$ is $2^{-\mu}$. These are independent Bernoulli trials. For each block $B \in S$, define $X_B^\mu \in \{0,1\}$ to be the random variable indicating whether the block belongs to superblock level μ and let D_μ indicate their sum, which is a Binomial distribution with parameters $(|S|, 2^{-\mu})$ and expected value $E[D_\mu] = |S|2^{-\mu}$.

For level μ to exist in any valid proof, at least m blocks of level μ must have been produced by the honest parties or the adversary. We will now show that m blocks of level $\mu = \log(|S|)$ are produced with negligible probability in m.

As all of the X^μ are independent, we can apply a Binomial Chernoff bound to the probability of the sum. Therefore we have

$\Pr[D_\mu \geq (1+\Delta)E[D_\mu]] \leq \exp(-\frac{\Delta^2}{2+\Delta}E[D_\mu])$. But for this μ we have that $E[D_\mu] = 1$. Therefore $\Pr[D_\mu \geq 1+\Delta] \leq \exp(-\frac{\Delta^2}{2+\Delta})$. Requiring $1+\Delta = m$, we get $\Pr[D_\mu \geq m] \leq \exp(-\frac{(m-1)^2}{m+1})$, which is negligible in m. □

Theorem 6.9 (Large upchain expansion). *Let $\mathcal{C}$ be an honestly generated chain and let $\mathcal{C}' = \mathcal{C}\uparrow^{\mu-1}[i : i + \ell]$ with $\ell \geq 4m$. Then $|\mathcal{C}'\uparrow^{\mu}| \geq m$ with overwhelming probability in m.*

Proof. Assume the $(\mu-1)$-level superchain has $4m$ blocks. Because each block of level $\mu - 1$ was generated as a query to the random oracle, it constitutes an independent Bernoulli trial and the number of blocks in level μ, namely $\pi\uparrow^{\mu}$, is a Binomial distribution with parameters $(4m, 1/2)$. Clearly $\Pr[|\pi\uparrow^{\mu}| = m] \leq \Pr[|\pi\uparrow^{\mu}| \leq m]$. Observing that $E[\pi\uparrow^{\mu}] = 2m$ and applying a Chernoff bound, we get $\Pr[|\pi\uparrow^{\mu}| \leq (1 - \frac{1}{2})2m] \leq \exp(-\frac{(1/2)^2}{2}2m)$ which is negligible in m.

This probability bounds the probability of fewer than m blocks occurring in the μ level restriction of $(\mu - 1)$-level superchains of more than $4m$ blocks. □

Corollary B.2 (Small downchain support). *Assume an honestly generated chain $\mathcal{C}$ and let $\mathcal{C}' = \mathcal{C}\uparrow^{\mu}[i : i + m]$. Then $|\mathcal{C}'\downarrow\uparrow^{\mu-1}| \leq 4m$ with overwhelming probability in m.*

Proof. Assume the $(\mu - 1)$-level superchain had at least $4m$ blocks. Then by Theorem 6.9 it follows that more than m blocks exist in level μ with overwhelming probability in m, which is a contradiction. □

Theorem 6.11 (Optimistic succinctness). *Non-interactive proofs-of-proof-of-work produced by honest provers on honestly generated chains with random scheduling are succinct with the number of blocks bounded by $4m\log(|\mathcal{C}|)$, with overwhelming probability in m.*

Proof. Assume $\mathcal{C}$ is the honest parties chain. From Theorem 6.8, the number of levels in the NIPoPoW is at most $\log(|\mathcal{C}|)$ with overwhelming probability in m.

First, observe that the count of blocks in the highest level will be less than $4m$ from Theorem 6.9; otherwise a higher superblock level would exist.

From Corollary 4.8, we know that at all levels μ the chain will be good. Therefore, for each μ superchain $\mathcal{C}$ the supporting $(\mu - 1)$-superchain will only need to span the m-long suffix of the μ-superchain above.

Then for the m-long suffix of each superchain of level μ, the supporting superchain of level $\mu - 1$ will have at most $4m$ blocks from Corollary 6.10.

Therefore the size of the proof is $4m\log(|\mathcal{C}|)$, which is succinct. □

Theorem 8.1. *Velvet non-interactive proofs-of-proof-of-work on honest chains by honest provers remain succinct as long as a constant percentage g of miners has upgraded, with overwhelming probability.*

Proof. From Theorem 6.11 we know that the proofs π contain only a $O(polylog(m))$ amount of blocks. For each of these blocks, the velvet client needs to include a followUp tail of blocks. Assume a percentage $0 < g \leq 1$ of miners have upgraded with NIPoPoW support. Then the question of whether each block in the honest chain is upgraded follows a Bernoulli distribution. If the velvet proof were to be larger than Δ times the soft fork proof in the number of blocks included, then this would require at least one of the followUp tails to include at least Δ sequential unupgraded blocks. But since the upgrade status of each block is independent, the probability of this occurring is g^{Δ}, which is negligible in Δ. □

REFERENCES

[1] Andreas M Antonopoulos. 2014. *Mastering Bitcoin: unlocking digital cryptocurrencies.* " O'Reilly Media, Inc.".

[2] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. 2014. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains* (2014).

[3] Lear Bahack. 2013. Theoretical Bitcoin Attacks with less than Half of the Computational Power (draft). *arXiv preprint arXiv:1312.7013* (2013).

[4] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-interactive zero-knowledge and its applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 103–112.

[5] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. 2016. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*. Springer, 106–125.

[6] Quinn DuPont. 2017. Experiments in Algorithmic Governance: A history and ethnography of "The DAO," a failed Decentralized Autonomous Organization. *Bitcoin and Beyond: Cryptocurrencies, Blockchains and Global Governance. Routledge* (2017).

[7] Mark Friedenbach. 2014. Compact SPV proofs via block headercommitments. https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg04318.html. (2014).

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

[9] Juan A Garay, Aggelos Kiayias, and Nikos Leonardos. 2016. The Bitcoin Backbone Protocol with Chains of Variable Difficulty. (2016).

[10] Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. 2016. Proofs of Proofs of Work with Sublinear Complexity. In *International Conference on Financial Cryptography and Data Security*. Springer, 61–78.

[11] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[12] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. 2016. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press.

[13] Tier Nolan. 2013. Alt chains and atomic transfers. bitcointalk.org. (May 2013).

[14] Andrew Poelstra. 2015. On stake and consensus. https://download.wpsoftware.net/bitcoin/pos.pdf. (2015).

[15] Meni Rosenfeld. 2014. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009* (2014).

[16] Laura Shin. 2017. For First Time Bitcoin Accounts for Less Than Half of Market Cap of All Cryptocurrencies. https://www.forbes.com/sites/laurashin/2017/05/16/for-first-time-bitcoin-accounts-for-less-than-half-of-market-cap-of-all-cryptocurr (May 2017).

[17] Riccardo Spagni. 2015. A Formal Approach Towards Better Hard Fork Managemenet. https://forum.getmonero.org/4/academic-and-technical/303/a-formal-approach-towards-better-hard-fork-management. (May 2015).

[18] Alin Tomescu and Srinivas Devadas. 2017. Catena: Efficient non-equivocation via Bitcoin. In *IEEE Symp. on Security and Privacy*.

[19] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014).