Proofs of Proofs of Work with Sublinear Complexity

Aggelos Kiayias, Nikolaos Lamprou(✉), and Aikaterini-Panagiota Stouka

National and Kapodistrian University of Athens, Athens, Greece
aggelos@di.uoa.gr, nikolaoslabrou@yahoo.gr, katerinastou21@yahoo.gr

Abstract. In the setting of blockchain based transaction ledgers we study the problem of "simplified payment verification" (SPV) which refers to the setting of a transaction verifier that wishes to examine the last k blocks of the blockchain (e.g., for the purpose of verification of a certain transaction) using as only advice the genesis block (or some "checkpoint" block that is known to it).

The straightforward solution to this task requires the delivery of the blockchain, the verification of the proof of work it contains, and subsequently the examination of the last k blocks. It follows that the communication required to complete this task is linear in the length of the chain.

At first thought the above seems the best one can hope: a sublinear in the length of the chain solution to the problem will be susceptible to an attacker that, using precomputation, can fool the verifier.

Contrary to this intuition, we show that with a suitable modification to the current Bitcoin blockchain protocol (that incurs a single hash expansion in each block and gives rise to the notion of an *interconnected* blockchain) we can produce *proofs of proof of work* with sublinear complexity in the length of the chain hence enabling SPV to be performed much more efficiently.

1 Introduction

Bitcoin, introduced by Nakamoto [10], and other numerous decentralized cryptocurrencies that were developed using the same codebase, have at their core a blockchain-based ledger of transactions. In these systems the ledger is a distributed data structure where transactions are organized into blocks. The blocks themselves form a hash chain so that each block is associated with a proof of work puzzle [1,4,7,11] and it points to a single previous block. A valid blockchain is rooted at a genesis block that is hard-coded into the client software that supports the distributed ledger.

The blockchain is maintained by a dynamically changing set of players that are called miners. The main task of each miner is to solve a proof of work and thus produce the next block. A transaction is validated when it is added to the blockchain. The certainty placed upon a certain transaction is associated to

This research was supported by ERC project CODAMODA, # 259152.

J. Clark et al. (Eds.): FC 2016 Workshops, LNCS 9604, pp. 61–78, 2016.
DOI: 10.1007/978-3-662-53357-4_5

the depth that is found in the blockchain. The deeper a transaction is placed in the blockchain the more certain it becomes that it will remain there. This was originally argued in [10] in a simplified model where the honest players are assumed to act in unison and the adversary follows a specific strategy. Security in the setting where the honest players are distributed and the adversary may exploit this was subsequently formally considered and proven in [6]. In this latter work two properties are introduced: common prefix and chain quality, and it is shown that with overwhelming probability in a parameter k, honest players will agree on the same prefix of the blockchain (after k blocks are pruned) and such chain will contain a certain percentage of blocks produced by honest players. These two properties were shown to imply that transactions in the ledger are "persistent" and that the ledger itself has "liveness" i.e., it is impossible for the adversary to stifle new transactions indefinitely.

In this work we study the problem of simplified payment verification or SPV. Introduced in [10], this problem considers a verifier that wishes to examine the ledger for a recent transaction. The verifier has as input a transaction identifier, say tx as well as the genesis block.[1] The verifier, with only this information, wishes to verify with high probability that the transaction has been included in the ledger and be sure that it will remain there with high probability. Based on the results stated above it is simple to implement such SPV verification as follows: the verifier will query the network and receive various blockchains (possibly some generated by an adversary that wishes to fool him) containing only the block headers for most blocks except the last k ones that are provided with all transactions (such communications have been referred as "SPV proofs"); the verifier will verify the integrity of the received chains and will select the one with the most proof of work. Finally, if the transaction with identifier tx is found at a depth say k it will conclude that the transaction is valid (with a probability of error as detailed by the persistence property of [6]). This SPV operation is more efficient than running a "full node" since not all transaction history needs to be received and verified.

An important observation regarding the above solution is that it is seemingly impossible to improve to below linear complexity in the length of the blockchain. Indeed, if a verifier is only allowed sublinear complexity it will not be able to verify that all the proofs of work in the received blockchains are valid. In this way it will only be able to verify fragments of given blockchains at best and this may open the door to potential attacks by an adversary that prepares ahead of time suitable blockchain fragments that are unrelated to the genesis block but are otherwise seemingly valid portions of the main blockchain.

Our Results. In this work, we present a method to construct *proofs of proof of work* that have sublinear complexity in the blockchain length. These proofs are capable of enabling "lite" SPV verification that is substantially more efficient compared to the full SPV verification described above. Our solution requires a modification in the current Bitcoin codebase that incurs a small overhead per each block that never exceeds a logarithmic function in the length of the

[1] Or a checkpoint block, if the verifier is in possession of such a block.

blockchain and can be compressed to a single hash value; this gives rise to a special type of blockchain that we call an *interconnected* blockchain.

In our solution the lite verifier receives a pair $(\mathcal{X}, \pi)$, where $\mathcal{X}$ is a blockchain fragment corresponding to the rightmost k blocks of the senders' chain and π is a proof of the proof of work that the pruned chain (denoted by $\mathcal{C}^{\lceil k}$) represents. Constructing the proof π is achieved via the following mechanism.

Recall that each block in a blockchain is associated with a proof of work which corresponds to a suitably formed value w that satisfies the inequality $H(w) < T$ where H is a hash function (e.g., SHA-256 in the case of Bitcoin) and T is a target value which is determined via a target calculation function (this function takes into account the rate of growth of the blockchain and reflects the size of the set of miners that participate in the protocol).

Our new mechanism operates as follows: whenever a block with a lower than usual hash is created we mark this in the next block as a continuation of a "deeper" chain that we call the inner chain of depth i where i is the greatest integer for which it holds $H(w) < T/2^i$. Specifically, each block carries a vector of pointers (which can be thought of expanding the standard reverse pointing link in a blockchain across multiple levels). In this way, in our modified blockchain, a block will have a vector of pointers denoted as interlink $= \langle s_0, \ldots, s_l \rangle$ such that s_0 points to the genesis block and for $i = 1, \ldots, l$, s_i points to the previous block with hash value smaller than $T/2^i$. Note that l would be the largest integer for which a hash in the blockchain is less than $T/2^l$ (and s_l is a pointer to the most recent such hash).

The construction of the proof π is as follows: the sender will remove the k-suffix from its local chain $\mathcal{C}$ and denote it as $\mathcal{X}$. Then, in the remaining prefix denoted as $\mathcal{C}^{\lceil k}$, he will attempt to find the deepest inner chain π that is of length at least m (the value m is a security parameter). The pair $(\mathcal{X}, \pi)$ will be the proof and will be transmitted to the lite verifier. In the optimistic scenario where the adversary does not actively interfere there is no further interaction between the lite verifier and the prover. In the general case, the adversary may invest hashing power in order to produce blocks with very low target, with the only purpose to increase the communication complexity between a lite verifier and a prover. In such case, the lite verifier engages in further interaction with the provers in order to be fully convinced.

Finally, we present a formal treatment of security for lite SPV proofs. Our argument is a *simulation-based* one. Security for a lite verifier is captured by the following statement: for any adversary that produces an SPV proof directed to a lite verifier there is an adversary producing an SPV proof directed to a regular SPV verifier that produces the same outcome. We establish the above security condition with overwhelming probability in m where m is a parameter of the lite verification protocol.

In our construction the complexity of the lite verifier will be shown to be $O(m \log n)$ in the optimistic case which can be improved in a straightforward manner to be $O(m \log \log n)$ where n is the blockchain length using Merkle trees.

Related Work. The first suggestion we are aware of[2] regarding the use of low hash values that appear naturally in the blockchain as an indicator of total proof of work was in a post in the Bitcoin forum [9]. A suggestion for a modification of the Bitcoin protocol was made in this post to include in each block a single "back-link" to the most recent block with a hash value less than half that of the previous block. Potential benefits of this modification were discussed including the possibility of using such pointers in SPV proofs.

In a short article posted in the Bitcoin-development list [5] this idea was taken further by suggesting to include a data structure containing various such back-links to previous blocks. An exact form of the data structure was not described and it was suggested that further research would be required to determine the most suitable data structure. A number of use-cases were discussed including the possibility of constructing compact SPV proofs as well as the design of "symmetrical two-way pegging schemes" between Bitcoin and side-chains. This latter concept, formulated in [2], enables the transfer of ledger assets from one main chain (say Bitcoin) to pegged side-chains. It is argued that such side-chains enable experimentation with new features in blockchain design and hence pegging them to, say, the Bitcoin blockchain enables the fluid transition of assets to these alternative blockchains (that potentially offer enhanced functionality or robustness features that are difficult to be assessed ahead of time). The pegging operation itself requires the main blockchain to enable transactions that move assets to special outputs that can only be "unlocked by an SPV proof of possession in the side-chain." This effectively enables the transfer of assets from the main chain to the side-chain as well as their return to the main chain in case the owner of the assets wishes to do that. Building efficient SPV proofs is an important aspect of this mechanism and a suggestion along the lines of [5] is presented in [2]. The possibility of exploiting the SPV proof mechanism by an adversary is recognized and some countermeasures are briefly discussed however without any formal analysis or the conclusion to an explicit data structure and a proof construction algorithm.

Finally, we note that the Bitcoin modifications related to SPV node verification do not affect the operation of the full nodes of the blockchain protocol and thus are of a different nature to chain selection and reward mechanism modifications such as those suggested in the GHOST rule for blockchain selection [12] or the inclusive blockchain protocols of [8].

2 Preliminaries

We follow the same notation as the Bitcoin backbone protocol, [6]. Below we introduce some basic notation and terminology.

- $G(.), H(.)$ are cryptographic hash functions with output in $\{0,1\}^\kappa$.
- A block B has the following form: $B = \langle s, x, ctr \rangle$ where $s \in \{0,1\}^\kappa, x \in \{0,1\}^*, ctr \in \mathbb{N}$.

[2] We thank the anonymous reviewers of the 3rd Workshop on Bitcoin and Blockchain Research for providing pointers to the relevant forum posts.

- A round is the period during which all the parties in the network are able to synchronize and obtain each other's messages. The scheduling of messages is controlled by the adversary. Furthermore, in each round, the adversary is able to introduce arbitrary number of messages and deliver them selectively to the parties.
- The rightmost block of the chain $\mathcal{C}$ is the head $(\mathcal{C})$ and $\mathcal{C}^{\lceil k}$ is the chain $\mathcal{C}$ without the rightmost k blocks. If we suppose that head $(\mathcal{C}) = \langle s, x, ctr\rangle$ and the previous block is $\langle s', x', ctr'\rangle$ then it holds $s = H(ctr', G(s', x'))$; in general every block has a reference to the previous block and thus all the blocks form a chain.
- The block header can be defined as $\langle ctr, G(s, x)\rangle$.
- A proof of work is finding a value $ctr : 0 \leq ctr < 2^{32}$ so that $H(ctr, G(s, x)) < T$ where $T \in \{0, 1\}^{\kappa}$ is the target of the block.
- The value x is the information is stored in the a block. In the case of the Bitcoin protocol this information is a sequence of transactions (organized in the form of a Merkle tree).

3 Interconnected Blockchains

In order to produce a proof of proof of work, the prover with local chain $\mathcal{C}$ will produce the pair $(\mathcal{X}, \pi)$ by setting the $\mathcal{X}$ to be the k-suffix of its local chain $\mathcal{C}$ and computing the proof π. The proof π constitutes a collection of blocks that are part of chain $\mathcal{C}^{\lceil k}$ and are collected in a specific way detailed below.

A proof π is associated with an integer $i \in \mathbb{N}$ which is the *depth* of the proof. The blocks contained in the proof are determined by a special type of chain that we will call innerchain$_i$.

Definition 1. *An* innerchain$_i$ *parameterized by an index* $i > 0$ *is a valid chain derived from a chain* $\mathcal{C}$ *that has the feature that each block* $B = \langle s, x, ctr\rangle$ *satisfies* $H(ctr, G(s, x)) < T/2^i$.

In an innerchain$_i$ we observe that, intuitively, each block represents as much proof of work as 2^i blocks with target T of the parent chain $\mathcal{C}$. As a result, if the proof π consists of m blocks, then the innerchain$_i$ represents proof of work as much as $m \cdot 2^i$ blocks of target T.

In our system, in order to produce the proof, provers should extract innerchain$_i$ for some $i > 0$ from $\mathcal{C}^{\lceil k}$. This means that for every $i \in \mathbb{N}$ all blocks with hash value smaller than $T/2^i$ should form a chain. This leads to the notion of an *interconnected blockchain*.

Every block with hash value smaller than $T/2^i$ needs a pointer to the previous block with hash value smaller than $T/2^i$. This does not exist in regular blockchains of Bitcoin, so a suitable modification with a sequence of pointers in each block in $\mathcal{C}$ is needed. The addition of this data structure inside each block, that we will call interlink[] will give rise to an "interconnected blockchain." A graphical description of an interconnected chain is shown in Fig. 1.

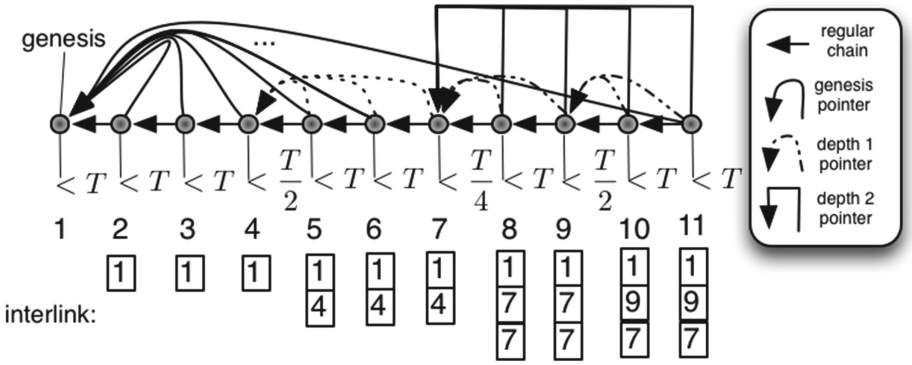

Fig. 1. A graphical depiction of an interconnected blockchain of 11 blocks that contains an inner chain of depth 1 (comprised of blocks $(1, 4, 7, 9)$) and an inner chain of depth 2 (comprised of blocks $(1, 7)$. The value of the interlink vector for each block is also shown.

The interlink data structure which should be included in each block B is *dynamic* and we formally define it below. Note that a block will be defined as $B = \langle s, x, ctr, \text{interlink}\rangle$ and the blockheader as $\langle ctr, G(s, x, \text{interlink})\rangle$.

Definition 2. interlink *is a vector, which is included in each block* B *and for which it holds that, for all* $i > 0$, interlink$[i]$ *is the hash of the previous block of* B *in chain* $\mathcal{C}$ *with hash value smaller than* $T/2^i$. interlink$[0]$ *is the hash of the genesis block.*

Note that the length of the vector depends on the type of blocks that exist in chain $\mathcal{C}$. Suppose that $B = \langle s, x, ctr, \text{interlink}\rangle$ is the head of the chain and $B' = \langle s', x', ctr', \text{interlink}'\rangle$ is the previous block; then interlink is equal to interlink$'$ after being updated with the algorithm we describe next.

3.1 Description of the Interlink-Update Algorithm

The purpose of this algorithm is to determine the operation that is needed in order to properly form an interconnected chain. When mining a new block, we must determine the appropriate set of pointers that will be used. Given the hash of the previous block denoted by s, the algorithm performs the following.

- Finds $max\ i$, so that $s = H(ctr', G(s', x', \text{interlink}')) < T/2^i$.
- Extends the size of interlink$'$ by adding $i - i'$ elements where the value i' equals $sizeof(\text{interlink}')$ (only if $i' < i$).
- Assign $H(ctr', G(s', x', \text{interlink}')) = s$ to interlink$[1], \ldots,$ interlink$[i]$.

Algorithm 1. Interlink-Update

Input : $B' = \langle s', x', ctr', \text{interlink}' \rangle$
Output: interlink

```
1  dynamic data structure interlink;
2  int entry, i = 0;
3  interlink = interlink';
4  while (H(ctr', G(s', x', interlink')) < T/2^i) do ;      // finds vector's max
   length
5
6      entry = i;
7      i = i + 1;
8  if entry = 0 then
9      return interlink;
10 else
11     for (i = 1, i ≤ entry, i++) do
12         if i > sizeof(interlink) then
13             sizeof(interlink)+ = 1;
14         interlink[i] = H(ctr', G(s', x', interlink'));
15     return interlink;
```

4 Proving Proof of Work with Sublinear Complexity

4.1 Description of the Prover

When a prover with a local chain $\mathcal{C}$ receives a request from a lite verifier that asks for the rightmost k blocks, then it constructs a proof π of the proof of work in $\mathcal{C}^{\lceil k}$ using the algorithm ConstructProof.

This algorithm's input is $\mathcal{C}^{\lceil k-1}$ and its output is innerchain$_i = \pi$, where i is the max i so that there are at least m (security parameter) blocks with hash value smaller than $T/2^i$ in $\mathcal{C}^{\lceil k}$. The algorithm ConstructProof (which we will describe below) calls the next algorithm, which is ConstructInChain.

This algorithm uses a hash table, which is a data structure that stores (key, value) pairs. In our case, the hash table stores blocks with their hash values.

The algorithm ConstructInChain has as input a chain $\mathcal{C}$ and an i. Its output is a chain with all the blocks with hash value smaller than $T/2^i$ in $\mathcal{C}^{\lceil 1}$.

4.2 Description of the Lite Verifier

We consider the case when a lite verifier has received $(\mathcal{X}_A, \pi_A)$ and $(\mathcal{X}_B, \pi_B)$ from provers A, B respectively that supposedly hold chains $\mathcal{C}_A$ and $\mathcal{C}_B$. Its purpose is to find which proof represents the chain with the most proof of work.

– Without loss of generality, let $\pi_A = \text{innerchain}_\mu$, so its blocks have hash value smaller than $T' = T/2^\mu$ and $\pi_B = \text{innerchain}_{i+\mu}$, so its blocks have hash value smaller than $T/2^{i+\mu} = T'/2^i$ with $i \geq 0$.

Algorithm 2. ConstructInChain

Input : $\mathcal{C}$,i
Output: $InnerChain[\,]$

```
1  data structure hashtable;
2  int x, y = 0;
3  int j = C.length;
4  int inner = 0;
5  block B ;                                   // as it is defined above
6  B = C[j];
7  x = B.interlink[i] ;                        // B.interlink is interlink in block B
8  y = B.interlink[i];
9  initialize hashtable(·) with all pairs (s, B) from C;
10 while (x!= 0) do
11     x = B.interlink[i];
12     if x!= 0 then
13         B = hashtable(x);
14         inner = inner + 1;
15 int c = inner;
16 chain InnerChain[c] ;                       // data structure which stores blocks
17 while (y!= 0) do
18     y = B.interlink[i];
19     if y!= 0 then
20         B = hashtable(y);
21         InnerChain[c] = B;
22         c = c − 1;
23 return InnerChain;
```

Firstly the lite verifier examines whether the length of π_A and π_B is more than m without the genesis and whether the length of the suffixes is k respectively. If a proof does not satisfy the above properties, it is rejected.

Next, the lite verifier examines whether there is a common block x in $\mathcal{X}_A$ and $\mathcal{X}_B$, because in this case the lite verifier can find which chain represents more proof of work easily. Specifically this means that there is a fork between $\mathcal{C}_A$ and $\mathcal{C}_B$ in the last k blocks. So the lite verifier chooses the suffix that represents the most proof of work (more blocks after x since we assume the same T).

If there is no common block in the suffixes then the lite verifier will execute the algorithm MaxChain$[\pi_A, \pi_B]$ which will decide which proof represents the chain with the most proof of work. This algorithm may require additional interaction with A, B and operates as follows.

MaxChain uses two sub-procedures called RemoveCPhigh and RemoveCPlow. The RemoveCPlow algorithm with input (π_A, π_B) just prunes the common blocks,

sets π'_A, π'_B to be the proofs without these common blocks and sets b to be the most recent common block in π_A, π_B.

Algorithm 3. ConstructProof

```
   Input  : C^⌈k−1
   Output: Proof[ ]
 1 int size = C^⌈k−1.length;
 2 int maxtarget = 0;
 3 chain Proof[ ];
 4 while (C[size].interlink[maxtarget + 1] != 0) do
 5 |  maxtarget = maxtarget + 1 ;
 6 int i = maxtarget;
 7 if maxtarget > 0 then
 8 |  Proof[ ] = ConstructInChain [C^⌈k−1, i];
 9 |  while (Proof.length < m ∧i > 0) do
10 |  |  i = i − 1;
11 |  |  Proof[ ] = ConstructInChain [C^⌈k−1, i];
12 |  if (i > 0) then
13 |  |  return genesis‖Proof;
14 |  else
15 |  |  return C^⌈k−1;
16 return C^⌈k−1;
```

The RemoveCPhigh on input (π_A, π_B) will actively query B for the chain with blocks with hash value smaller than $T/2^\mu$ that is omitted in π_B. Formally, it will return (π'_A, π'_B, b), where π'_A, π'_B are the proofs without the common prefix and b is the most recent common block with hash value smaller than $T/2^\mu$ in $\mathcal{C}_A$ and $\mathcal{C}_B$. In more detail, it operates as follows:

- We suppose that the proofs are stored in two arrays respectively. The algorithm looks for block $\pi_B[1]$ in π_A and it continues until it finds a $\pi_B[i']$ that it is not in π_A. As $\pi_B[i'-1]$ is included in π_A, there is a j, so that $\pi_A[j] = \pi_B[i'-1]$.
- It asks B for an array V with blocks with hash value smaller than $T/2^\mu$ between $\pi_B[i'-1]$ and $\pi_B[i']$. RemoveCPhigh will fail in case the array V is not returned by B.
- It finds min $j' \geq j+1$ so that $\pi_A[j']$ differs from $V[j'-j]$.
- $\pi_{B'}$ is π_B without the first $i'-1$ blocks and $\pi_{A'}$ is π_A without the first $j'-1$ blocks.
- $b = \pi_A[j'-1]$.
- Return (b, π'_A, π'_B).

Next we describe the algorithm MaxChain. Given diverging π_A, π_B, the algorithm will select the proof with the most proof of work as long as the diverging suffix is long enough (as determined by a parameter m). In case the algorithm

cannot make a decision it will recurse, requesting proofs with lower depths from A, B as needed, until it reaches level 0 where a decision will be made independently of the parameter m. During these recursion steps if one of the communicating nodes, A, B, fails to support its proof (by providing the extra blocks that the lite node requests) the MaxChain algorithm will conclude that the opposing chain is the correct one (or it will inevitably fail in case no node is responding to its requests). In more detail the algorithm operates as follows:

- Firstly, the algorithm calls RemoveCPlow to obtain the pruned suffixes (this does not require interaction). Then, it checks whether $i > 0$. In this case, the proofs have different depths and the algorithm checks whether $\pi'_B.pow \geq \pi'_A.pow$ and simultaneously $\pi'_B.length \geq m$. If these two conditions hold, the lite verifier will choose π_B. Otherwise the algorithm uses RemoveCPhigh in order to discover the common prefix from the proofs π_A, π_B (this will require interacting with B).
- Secondly the algorithm checks which of the proofs represents the most proof of work. The proof with the most proof of work is returned if it has length at least m for π'_B and $2^i m$ for π'_A. Note that in this case a decision is made whose security hinges on the parameter m.
- If the proof with the most proof of work is not long enough the algorithm asks B or both A, B for a proof with a lower depth of the part of the chain ($\mathcal{C}_A^{\lceil k}$ or $\mathcal{C}_B^{\lceil k}$) without the common prefix and continues recursively. We use $\mathsf{Request_B}[b, y]$ to denote a request from B for a proof with hash value smaller than $T/2^y$ of the chain $\mathcal{C}_B^{\lceil k}$ that is rooted at block b. Similarly, $\mathsf{Request_A}[b, y]$ functions in the same way for player A.[3]

Eventually, the algorithm will either obtain diverging suffixes that are long enough or will reach the depth 0 (where the actual target T is used) where a decision will be made based solely on the amount of proof of work. This will determine the winning proof and the lite verifier may proceed to execute another comparison or conclude the process.

5 Efficiency Analysis

In this section we present the efficiency analysis of the proof system: first we discuss space complexity, i.e., the expansion that is required in the local storage of the full nodes due to the data structure of the interconnected blockchain. Then, we analyze the communication that is required to send the proof and the verification complexity of the lite verifier.

5.1 Space Complexity

We first show a suitable upper bound on the vector interlink that is the only addition in each block of the interconnected blockchain.

[3] We note that there is no provision for authenticated channels in the Bitcoin setting; hence when we refer to a request for information from a certain player this is not performed in an authenticated fashion.

Algorithm 4. MaxChain

Input : π_A, π_B chains consisted of blocks with hash value smaller than $T/2^{\mu}, T/2^{i+\mu}$ resp., s.t. $i \geq 0$, or *fail*
Output: $Max[\,]$

```
1  chain Max[ ],Proof_A[ ], Proof_B[ ];
2  block b;
3  if either of π_A, π_B equals fail then
4      return the other one ;
5  else
6      (b, π'_A, π'_B) = RemoveCPlow[π_A, π_B];
7      if (i > 0) then
8          if ((π'_B.pow ≥ π'_A.pow) ∧(π'_B.length ≥ m)) then
9              return π_B
10         else
11             response:= RemoveCPhigh[π_A, π_B];
12             if response = fail then
13                 return π_A
14             else
15                 parse response as (b, π'_A, π'_B);
16     if (π'_B.pow ≥ π'_A.pow) then
17         if ((π'_B.length ≥ m) ∨ (i + μ = 0)) then
18             return π_B
19         else
20             if (i > 0) then
21                 Proof_B = Request_B[b, μ];
22                 Proof_A = b‖π'_A;
23             else
24                 Proof_A = Request_A[b, μ − 1];
25                 Proof_B = Request_B[b, μ − 1];
26     else
27         if ((π'_A.length ≥ 2^i · m) ∨ (i + μ = 0)) then
28             return π'_A
29         else
30             if (i > 0) then
31                 Proof_B = Request_B[b, μ];
32                 Proof_A = b‖π'_A;
33             else
34                 Proof_A = Request_A[b, μ − 1];
35                 Proof_B = Request_B[b, μ − 1];
36     return MaxChain[Proof_A, Proof_B];
```

Theorem 1. *Let n be the length of a chain $\mathcal{C}$ that is consisted of blocks with hash value smaller than $T = 2^f$. Then the expected size of the dynamic vector* interlink, *is* $f - \sum_{i=1}^{f}(1 - \frac{1}{2^i})^n$.

Proof. We define a discrete random variable $X_j \in \{0, \ldots, f\}$ associated with each block $\mathcal{C}[j]$ so that

$$X_j = i \Longleftrightarrow \frac{T}{2^{i+1}} \leq H_B < \frac{T}{2^i}, i \in \{0, \ldots, f-1\}$$

$$X_j = f \Longleftrightarrow 0 \leq H_B < \frac{T}{2^f}$$

(H_B is the hash value of $\mathcal{C}[j]$).

The hash value of each chain's block H_B follows the uniform discrete distribution on $\{0, \ldots, T-1\}$. So

$$\Pr(X_j = i) = \Pr(\frac{T}{2^{i+1}} \leq H_B < \frac{T}{2^i}) = \frac{1}{2^{i+1}}, i \in \{0, \ldots, f-1\}$$

$$\mathsf{Pr}(X_j = f) = \Pr(0 \leq H_B < \frac{T}{2^f}) = \frac{1}{2^f}$$

It holds:

$$\sum_{i=0}^{f} \mathsf{Pr}(X_j = i) = 1$$

Then the size of the interlink follows $Y = max\{X_1, \ldots, X_n\}$ distribution.

If $0 \leq y < f$ then:

$$\mathsf{Pr}(Y \leq y) = (\mathsf{Pr}(X_j \leq y))^n = (\sum_{i=0}^{y} \frac{1}{2^{i+1}})^n = (1 - \frac{1}{2^{y+1}})^n$$

$$\mathsf{Pr}(Y = y) = \mathsf{Pr}(Y \leq y) - \mathsf{Pr}(Y \leq y-1) = (1 - \frac{1}{2^{y+1}})^n - (1 - \frac{1}{2^y})^n$$

It also holds: $\mathsf{Pr}(Y \leq f) = 1$ and $\mathsf{Pr}(Y = f) = 1 - (1 - \frac{1}{2^f})^n$

We have:

$$E(Y) = \sum_{y=0}^{f-1} y \cdot [(1 - \frac{1}{2^{y+1}})^n - (1 - \frac{1}{2^y})^n] + f \cdot [1 - (1 - \frac{1}{2^f})^n]$$

$$= (f-1) \cdot (1 - \frac{1}{2^f})^n - \sum_{i=1}^{f-1}(1 - \frac{1}{2^i})^n + f \cdot [1 - (1 - \frac{1}{2^f})^n]$$

$$= f - \sum_{i=1}^{f}(1 - \frac{1}{2^i})^n$$

□

In Fig. 2 we demonstrate a graph that shows that the size of interlink is logarithmic in n when n ranges in the current Bitcoin blockchain length and the target is kept stable at 2^{200}.

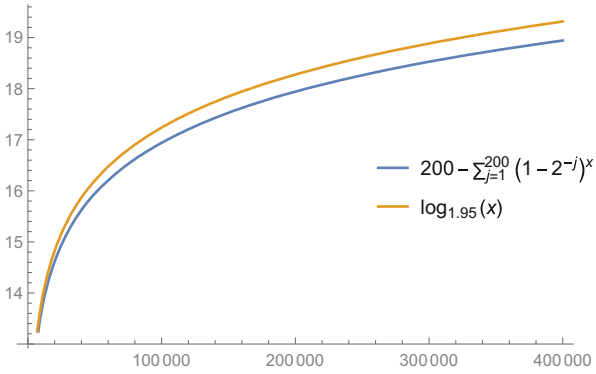

Fig. 2. Size of interlink as a function of blockchain length when target is $T = 2^{200}$.

Compressing the Interlink Vector Using Merkle Trees. To reduce the storage per block it is possible to compress the interlink vector using a Merkle tree. In more detail, instead of storing the whole interlink in each block we can organize the vector in a Merkle tree and store only the root hash in each blockheader. This demands the addition of only a hash value in each block instead of the sequence of hash values in interlink. The modifications needed in the ConstructProof algorithm are straightforward and we omit them.

5.2 Communication and Time Complexity

We will analyze now the size of the proof π. We will focus in the optimistic scenario, where the adversary does not create deep forks that cut into the proofs of the honest parties, i.e., when the k-suffix that the adversary sends has a common block with the suffix of the proofs sent by the honest provers. Note that the honest parties will not fork in the part of the chain before the k-suffix with overwhelming probability in k [6]. In such case the lite verifier chooses the chain with the most proof of work without having to perform any extra interaction with the provers (performing the Request steps in the MaxChain algorithm). Therefore the size of the proof will be the output of the ConstructProof algorithm.

Let $\mathcal{C}^{\lceil k}$ be the pruned local chain without the k-suffix of a prover that a lite verifier has asked, n the length of the chain and m the security parameter. Firstly we will prove that the probability with which a block of $\mathcal{C}^{\lceil k}$ has hash value smaller than $\frac{T}{2^i}$ is $\frac{1}{2^i}$.

If H_B is the hash value of a block B and $j \in \mathbb{N}$, $j < T$ then $\Pr(H_B = j \mid H_B < T) = 1/T$. It follows,

$$\Pr(H_B < T/2^i \mid H_B < T) = \sum_{j=0}^{T/2^i - 1} \Pr(H_B = j \mid H_B < T) = \frac{T/2^i}{T} = \frac{1}{2^i}$$

The number of blocks in $\mathcal{C}^{\lceil k}$ with hash value smaller than $T/2^i$ is a discrete random variable D_i that follows the Binomial distribution with parameters (n, $p_i = 1/2^i$) and its expected value is $E(D_i) = n \cdot p_i$.

Recall that the ConstructProof algorithm has output the innerchain$_{i_0} = \pi$, where i_0 is the maximum i so that there are at least m blocks with hash value smaller than $T/2^i$ in $\mathcal{C}^{\lceil k}$. As a result we must examine what is the depth i_0 of the proof that the algorithm returns and how many blocks (denoted by D_{i_0}) the proof π will contain.

In the next lemma we establish that the depth of the inner chain that the ConstructProof algorithm returns is quite close to the optimal value (which is roughly $\log(n/m)$).

Lemma 1. *Let n be the size of the local pruned chain $\mathcal{C}^{\lceil k}$ of the prover. Assume that $n < Tm$ and define i so that $2^i m \leq n < 2^{i+1}m$. Then it holds* $\Pr(D_{i-1} \leq m-1) \leq \exp(-\Omega(m))$.

Proof. Observe that $n \cdot p_{i-1} = n/2^{i-1} \geq 2^i m/2^{i-1} = 2m > m-1$. So according to the Chernoff bound[4] for the Binomial distribution it holds that:

$$\begin{aligned}\Pr(D_{i-1} \leq m-1) &\leq \exp(-(np_{i-1} - (m-1))^2/2np_{i-1}) \\ &\leq \exp(-1/(2/(2^{i-1})) \cdot (n(1/(2^{i-1})) - (m-1))^2/n)) \\ &\leq \exp(-(2m - m + 1)^2/2^3 m) \leq \exp(-\Omega(m))\end{aligned}$$

This completes the proof. □

Armed with this lemma we next observe that the length of the inner chain for the suitable index is not going to be substantially larger than m.

Lemma 2. *Let $n < Tm$ and define i so that $2^i m \leq n < 2^{i+1}m$. It holds that* $\mathsf{Pr}(D_{i-1} \geq 5m) \leq \exp(-\Omega(m))$.

Proof. Observe first that $2m/n \leq p_{i-1} = 1/2^{i-1} < 4m/n$. Consider the Chernoff bound on the upper tail that states $\mathsf{Pr}[X \geq (1+\delta)\mu] \leq \exp(-\delta^2\mu/3)$ when X is a Binomial distribution with mean μ and $\delta \in (0, 1]$. It follows that $\mathsf{Pr}[D_{i-1} \geq 5m] \leq \mathsf{Pr}[D_{i-1} \geq (1 + 1/4)p_{i-1}n] \leq \exp(-p_{i-1}n/48) \leq \exp(-m/24)$. □

We are now ready to state the theorem that establishes the efficiency of the proof that is constructed and communicated to the lite verifier.

Theorem 2. *The size of the proof π that the prover sends in response to a lite verifier in the optimistic case is $O(m)$ with overwhelming probability in m.*

Proof. In the optimistic case the proof π that the prover sends to the lite verifier is the output of the ConstructProof algorithm. If n is the length of the local chain from which the prover constructs the proof and we have $2^i m \leq n < 2^{i+1}m$ for an $i \geq 1$ then it holds that: The ConstructProof algorithm will return

[4] Here we use the following variant: $\Pr(X \leq k) \leq \exp(-(np-k)^2/2np)$, where $k \leq np$.

a proof of depth $i-1$ with overwhelming probability in m, as we proved in Lemma 1. Furthermore, the size D_{i-1} of the proof π will be bounded by $5m$ with overwhelming probability in m, as we proved in Lemma 2. This completes the proof. □

The above completes the argument for the optimistic case, where the adversary does not explicitly interfere and attempts to increase the complexity of the lite verifier. We note that in the case that the adversary interferes and makes the lite node to engage in extra communication by issuing the Request commands, he can only succeed in this with significant effort (by mining very low target blocks) and with bounded, albeit non-negligible, probability. It seems unlikely that an adversary will engage in this effort for the sole purpose of delaying a lite verifier and for this reason, we consider the optimistic efficiency analysis performed above to be quite indicative of the actual performance of the protocol.

Finally with respect to time complexity observe that in the optimistic case, the verifier will have to perform a number of verification steps that are proportional to the size of the proof that is received. It follows that the complexity of the verifier is also $O(m \log n)$.

Complexity When Using a Compressed Interlink Vector. The communication and time complexity in this case can be improved since in each block from the interlink vector committed in the Merkle root hash only a path in the tree needs to be transmitted. It follows easily that the complexity of the lite verifier in the optimistic case will be $O(m \log \log n)$.

6 Security Analysis

A successful attack against our lite verification mechanism suggests that a lite verifier reaches a different conclusion about a certain transaction compared to a full verifier. The proof argument for security is as follows: given an adversary $\mathcal{A}$ that responds to a lite verifier we construct an adversary $\mathcal{A}^*$ that responds to a full verifier. We will argue that with high probability the full verifier operating with $\mathcal{A}^*$ reaches the same conclusion as the lite verifier operating with $\mathcal{A}$.

Intuitively the above means that for any proof that a lite verifier accepts and processes there exists a full chain that can be recovered and produces the same output behavior for a regular SPV verifier.

The description of $\mathcal{A}^*$ is as follows:

1. $\mathcal{A}^*$ simulates the operation of $\mathcal{A}$ while additionally in each round acts as a full verifier and requests the chains from all the honest nodes denoted by $\mathcal{C}_1, \ldots, \mathcal{C}_e$ for some integer e. It maintains a "block tree" BT containing all blockchains and adds there any blocks that are produced by the adversary. Note that it is possible to $\mathcal{A}^*$ to perform this since in the random oracle model (that we adopt from [6]) it is possible for $\mathcal{A}^*$ to monitor all queries of $\mathcal{A}$ to the hash function $H(\cdot)$. Any queries made by $\mathcal{A}$ that do not correspond to valid blocks are ignored.

2. When $\mathcal{A}$ responds to a lite verifier with a pair $(\mathcal{X}, \pi)$, $\mathcal{A}^*$ searches in BT for a chain $\mathcal{C}$ that is consistent with $(\mathcal{X}, \pi)$, i.e., $\mathcal{X}$ is the suffix of $\mathcal{C}$ and π is a sub-chain of $\mathcal{C}$. If such a chain is found, then $\mathcal{A}^*$ response to a full verifier with $\mathcal{C}$. If no chain is found then $\mathcal{A}^*$ returns no response to the full verifier.

We perform our analysis in the model of [6]. Recall that in their model, there are n parties maintaining the blockchain, each allowed q queries to the hash function (thought of as a random oracle) and t of the parties are controlled by the adversary. The probability of finding a proof of work with a single hash query is $T/2^\kappa$ (recall that the target is T and is stable). We use the same notation as [6] and we denote $\alpha = (n-t)pq$, $\beta = pqt$ and $\gamma = \alpha - \alpha^2$. Intuitively, the parameter α represents the hashing power of the honest parties; it is also an upper bound on the expected number of solutions that the honest parties will obtain in one round; on the other hand β is the expected number of solutions that the adversary may produce in one round. Finally, γ is a lower bound on the expectation of a uniquely successful round, i.e., a round where a single honest party finds a proof of work solution.

We are now ready to formulate the theorem that establishes the security of lite verification. The theorem is conditioned on $\gamma > (1+\delta)\beta$ which roughly[5] corresponds to the setting where the honest parties command the majority of the hashing power.

Theorem 3. *(Security of lite verification) Let $\gamma > (1+\delta)\beta$ for some $\delta > 0$. A full verifier interacting with $\mathcal{A}^*$ reaches the same conclusion as the lite verifier operating with $\mathcal{A}$ with probability $1 - \exp(-\Omega(\delta^2 m))$.*

Proof. (Sketch). We compare any execution with $\mathcal{A}$ where a lite verifier requests a proof to an execution where a full verifier requests a proof from $\mathcal{A}^*$. We define an event BAD to be the event that the two verifiers report a different conclusion. An event BAD would necessarily correspond to the case 2 above in the definition of $\mathcal{A}^*$ when the latter fails to reconstruct a chain $\mathcal{C}$ from BT that corresponds to the proof $(\mathcal{X}, \pi)$ that the adversary $\mathcal{A}$ produces. Let NOWIT be this latter event and observe $\mathsf{BAD} \subseteq \mathsf{NOWIT}$. We will argue that whenever NOWIT happens then with overwhelming probability in m it holds that a proof originating from an honest party will win the comparison performed by the $\mathsf{MaxChain}$. Let this event be HWIN. In more detail we will prove that $\Pr(\neg\mathsf{HWIN} \wedge \mathsf{NOWIT})$ drops exponentially in m. Observe that this is sufficient since $\mathsf{BAD} \subseteq \neg\mathsf{HWIN}$ and hence it will follow that $\Pr(\mathsf{BAD})$ drops exponentially in m.

The event $\neg\mathsf{HWIN}$ suggests that the adversary $\mathcal{A}$ has managed to produce a proof for which no honest party could outperform in the view of the $\mathsf{MaxChain}$ procedure. Furthermore, if NOWIT happens also, it follows that it is impossible for $\mathcal{A}^*$ to reconstruct a chain that corresponds to the proof that wins the $\mathsf{MaxChain}$ algorithm. This suggests that the winning proof $(\mathcal{X}, \pi)$ contains blocks that were impossible to attach to the blockchain tree BT by $\mathcal{A}^*$, due to the fact

[5] Roughly because $\gamma = \alpha - \alpha^2$ and thus this condition approximates the "honest majority" condition only if α^2 is close to 0. See [6] for more details.

of not being valid extensions of a (level-0) chain. It follows that in the response $(\mathcal{X}, \pi)$, the proof π should diverge from all chains that belong to an honest party (otherwise all the blocks in $\mathcal{X}$ would have been attached to BT and a witness for $(\mathcal{X}, \pi)$ would be reconstructed by $\mathcal{A}^*$). Let b be the most recent common honestly generated block of π with the longest chain $\mathcal{C}$ from BT that belongs to an honest party. Given that MaxChain elected $(\mathcal{X}, \pi)$ over the proof provided by the owner of $\mathcal{C}$ it holds that π contains a sequence of at least m blocks starting from b (or later) that are of target $T/2^i$ where $i > 0$ is the depth of π. Let r be the round that block b was created. We will next show that the probability that $\mathcal{A}$ obtains m blocks with hashes less than $T/2^i$ faster than the honest parties' chains advance by $2^i m$ blocks is negligible in m. It follows that it will be with negligible in m probability that $\mathcal{A}$ can produce a proof that will be selected by MaxChain.

Let X_r be the random variable that is equal to 1 if r is a successul round (following the terminology of [6]). In [6] it is shown that in any s rounds following round r it holds that the length of the honest parties' chains will be at least $\ell + \sum_{l=r}^{s} X_l$ where ℓ is the length of an honest parties' chain at round r.

The number of rounds that will be required for the adversary to compute m blocks with hash less than $T/2^i$ follows a negative binomial distribution. The expectation for the number of rounds is $2^i \beta^{-1} m$ where $p = T/2^\kappa$ and $\beta = pqt$. By applying a tail bound for the negative binomial distribution we obtain that the probability the number of rounds is less than $(1-\delta/4)2^i\beta^{-1}m$ is $\exp(-\Omega(\delta^2 m))$.

On the other hand, in $(1-\delta/4)2^i\beta^{-1}m$ rounds, by applying a Chernoff bound, the probability that the honest parties will produce less than $(1-\delta/4)^2\gamma 2^i\beta^{-1}m$ blocks is bounded by $\exp(-\Omega(\delta^2 m))$.

Observe now that $\gamma > (1+\delta)\beta$ implies $\gamma(1-\delta/4)^2\beta^{-1} > 1$ and thus the probability that the proof of work of the chains owned by the honest parties will exceed that of the adversary is $1 - \exp(-\Omega(\delta^2 m))$. □

On the Feasibility and Infeasibility of Non-interactive and/or Constant Size Proofs. Observe that our security parameter for the proof is m and the size of the proof in the optimistic case is $O(m)$. In our construction, the lite verifier may require further interaction with the provers if it discovers forks in the inner chains that it receives. This leaves open the question whether shorter proofs can be achieved (e.g., constant size) or whether it is possible to obtain non-interactive proofs, i.e., proofs that require always a single message from the full nodes to the lite verifier. With respect to constant size proofs it is unlikely that the techniques like the ones we consider here would provide such an improvement: for instance, if a single block of exceptionally low hash value is transmitted as a proof of many proofs of work of proportional length, concentration bounds will not be able to provide a sufficiently low probability of attack. In other words, such short proofs might be exploitable by an attacker in very much the same way that the difficulty raising attack of [3] operates and hence they will not be secure. Similarly, given any non-interactive proof that goes arbitrarily low in terms of the inner chain it selects, one can always imagine

an attacker that attempts to fork in the very last block of the inner chain and thus gain an unfair advantage compared to the honest parties even in the honest majority setting. However this may be countered by requiring sufficient number of blocks following such low hash blocks; we leave for future work the feasibility of investigatng the design of short and secure non-interactive SPV proofs.

The Dynamic Setting. To account for a dynamically changing population of miners, in Bitcoin and related blockchain protocols, the target is recalculated at regular intervals. It is possible to build our interconnected blockchains in the dynamic setting as well; some care needs to be applied during verification of proofs however since target recalculation will need to be performed over the inner chains. We leave the analysis in the dynamic setting for future work.

Acknowledgement. The authors wish to thank Giorgos Panagiotakos for helpful discussions as well as the anonymous referees of the 3rd Workshop on Bitcoin and Blockchain Research for their valuable comments.

References

1. Back, A.: Hashcash (1997). http://www.cypherspace.org/hashcash
2. Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timn, J., Wuille, P.: Enabling Blockchain Innovations with Pegged Sidechains (2014). https://blockstream.com/sidechains.pdf
3. Bahack, L.: Theoretical bitcoin attacks with less than half of the computational power (draft). Cryptology ePrint Archive, Report 2013/868 (2013). http://eprint.iacr.org/
4. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993)
5. Friedenbach, M.: Compact SPV proofs via block header commitments, Bitcoin-development mailing list post (2014). https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2014-March/004727.html
6. Garay, J., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015)
7. Juels, A., Brainard, J.G.: Client puzzles: a cryptographic countermeasure against connection depletion attacks. In: NDSS. The Internet Society (1999)
8. Lewenberg, Y., Sompolinsky, Y., Zohar, A.: Inclusive block chain protocols. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 528–547. Springer, Heidelberg (2015)
9. Miller, A.: The high-value-hash highway, Bitcoin forum post (2012). https://bitcointalk.org/index.php?topic=98986.0
10. Nakamoto, S.: Bitcoin: a peer to peer electronic cash system (2008). http://bitcoin.org/bitcoin.pdf
11. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA (1996)
12. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in bitcoin. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 507–527. Springer, Heidelberg (2015)