Secure Scheme Against Compromised Hash in Proof-of-Work Blockchain

Fengjun Chen, Zhiqiang Liu(✉), Yu Long(✉), Zhen Liu(✉), and Ning Ding(✉)

The Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
{chenfengjun,ilu_zq,longyu,liuzhen,dingning}@sjtu.edu.cn

Abstract. Blockchain is built on the basis of peer-to-peer network, cryptography and consensus mechanism over a distributed environment. The underlying cryptography in blockchain, such as hash algorithm and digital signature scheme, is used to guarantee the security of blockchain. However, past experience showed that cryptographic primitives do not last forever along with increasing computational power and advanced cryptanalysis. Therefore, it is crucial to investigate the issue that the underlying cryptography in blockchain is compromised.

This paper aims at the challenge that the underlying hash algorithm is compromised in blockchain. In 2017, M. Sato et al. firstly addressed the issue by proposing a framework of transition approach from the compromised hash algorithm to a secure one. Nevertheless, this approach is actually a hardfork if it is applied to proof-of-work blockchain, which is much likely to cause disagreement of the blockchain community and should be avoided accordingly. To fill this gap, we propose a softfork transition scheme to deal with the challenge that compromised hash brings into proof-of-work blockchain. Our scheme provides a secure transition in the case of compromised hash, keeping the validity of past data in the blockchain as well. We also show that a proof-of-work blockchain with our scheme is much more secure than the original one (i.e. without our scheme).

Keywords: Blockchain · Compromised hash
Softfork transition scheme · Proof of work

1 Introduction

Blockchain technology has attracted great interest of researchers and developers since Bitcoin [1] was proposed by Nakamoto in 2008. As the first remarkably successful and secure implementation of blockchain, Bitcoin took groundbreaking use of proof-of-work mechanism, to solve double-spending problem in cryptocurrency and maintain consistency of data in a decentralized environment. In proof-of-work blockchain, block is constructed with collected transactions and hash of previous block header by nodes called *miners*. Miners contribute computational power to competition in solving a hard cryptographic puzzle. In return for the

M. H. Au et al. (Eds.): NSS 2018, LNCS 11058, pp. 1–15, 2018.
https://doi.org/10.1007/978-3-030-02744-5_1

computational work, the miner who first solves out the puzzle will receive mining reward (i.e. newly generated coins) and transactions fee, and the block generated by him could have the chance to be accepted by the network and appended to the chain. To maximize the profit, rational miners will always follow the longest chain to do computation. Split from main chain may appear when two miners get solutions almost at the same time, but it will be eliminated soon when length of one branch outweighs the others. If an adversary maliciously intends to overturn transactions in a block followed by k block, the only way is to re-do computation to create another chain from this block overtaking the currently longest chain. However it is hard to succeed when k and the whole computational power in network are large enough. In Bitcoin, a block with 6 following blocks is usually considered safe since the success probability of generation of a branch from it overwhelming main chain, for an adversary with 10% computational power of the entire network, is less than 0.1% [1].

The underlying hash algorithm in proof-of-work blockchain is one of the key factors for security of the blockchain. More specifically, in proof-of-work blockchain, hash is used in computing reference of block (i.e. hash of previous block in each block) and proof-of-work puzzle, constructing Merkle Tree and hashing the transactions to be signed. Thus the hash algorithm adopted in blockchain should have sufficient security margin against known attacks. An eligible cryptographic hash algorithm ought to be computationally secure against pre-image, second pre-image and collision attacks. But history of cryptography showed that all hash algorithms can not stay computationally secure forever. Evolution in mathematical cryptanalysis and quantum computing are two main reasons to bring possible compromise to hash algorithm. The blockchain protocol will face severe security risk or even fail to work if the hash algorithms used are compromised.

To deal with the challenge, the most intuitive measure is to replace compromised hash algorithm with a more secure one, which is also a response to 0-day failure of SHA256 in Bitcoin contingency plans [2]. It is a kind of hardfork solution, i.e. an upgrade strategy adopting new form of blocks or transactions that are incompatible with original blockchain protocol [3]. However, hardfork is much likely to result in disagreement in the blockchain community. The success of hardfork depends on switch of the entire computational power from original blockchain protocol to new blockchain protocol. In the decentralized environment, such an agreement outside consensus protocol is hard to achieve in blockchain community. Danger of split of original blockchain will come even if only a small fraction of computational power staying at non-upgraded chain. Until now, Bitcoin has never implemented any hardfork successfully and safely without split [3]. Besides the intuitive hardfork, M. Sato et al. have proposed a transition approach to protect transactions in the case of compromised hash [4]. Their scheme utilizes proof of existence (PoE) model and archives transactions using a more secure hash algorithm. Nevertheless, this approach is also actually a hardfork if it is applied to proof-of-work blockchain.

This paper aims to tackle the challenge that the underlying hash algorithm is compromised in proof-of-work blockchain, as well as to avoid hardfork which is difficult to deploy without split of the original blockchain.

Contribution. With the proof of existence model given in [4], we introduce two-layer proof-of-work framework and then establish a novel transition scheme from the compromised hash to a secure one. As far as we know, this is the first softfork scheme against compromised hash in proof-of-work blockchain, which is more feasible, practical and easier-to-deploy than hardfork solutions since it is backward-compatible such that nodes that still run original blockchain protocol can admit blocks of new version, and softfork mechanism can completely avoid split of blockchain system when the majority computational power of the system fulfills upgrading blockchain protocol. Our scheme provides secure transition for a proof-of-work blockchain in the case of compromised hash, keeping the validity of past data in the blockchain as well. We also analyze the security of our newly-proposed scheme and show that a proof-of-work blockchain with our scheme is much more secure than the original one (i.e. without our scheme).

Paper Organization. The remainder of the paper is organized as follows. Section 2 introduces proof-of-work consensus, compromised hash and its impact briefly. In Sect. 3, we describe the existing solution to compromised hash in blockchain. Section 4 presents a novel secure approach against compromised hash. Finally, Sect. 5 concludes this paper.

2 Preliminaries

2.1 Proof of Work

In proof-of-work blockchain, miners compete in brute-force search to successfully solve a hard cryptographic puzzle and win reward for block generation. Practically the puzzle is to construct a block whose hash is less than a certain value, which also known as *difficulty of mining*. Only block that follows the latest block and has a hash value less than difficulty of mining will be accepted by network. In the view of long term, the received reward for mining is proportional to rate of computational power nodes contribute in network.

Within block structure, a *target* field indicates the difficulty of mining. For a blockchain where value domain of hash algorithm used by consensus is D, the probability of successfully finding an eligible block is as follows:

$$Pr[H \leq T] = \frac{T}{D}$$

Where H is the hash value of generated block and T is the value of target.

Target is not a constant value but will be adjusted periodically, to make block generation time stable when the network's overall computational power is changed. In Bitcoin, for example, *target* will be re-calculated every 2016 blocks to keep average block generation time at 10 min:

$$T' = \frac{t_{sum}}{14 * 24 * 60 * 60s} * T$$

where T is old target, T' is the new target after adjustment and t_{sum} is the accumulated time to produce latest 2015 blocks [5,6], which is calculated based on timestamp written by miners within block.

Beyond that, a *nonce* field is afforded in block to provide a enough space to find proof-of-work solution. Before computing to mine a block, miners should at first collect transactions and then construct a complete block structure. After that they fill *nonce* with a random value until hash value of the block meets target requirement.

2.2 Compromise of Hash

The basic security of an ideal cryptographic hash algorithm $h(x)$ can be defined by the following properties.

1. *Pre-image resistance.* Given a hash value y it is difficult to find a value x such that $h(x) = y$.
2. *Second pre-image resistance.* Given a value x_1 it is difficult to find a different value x_2 such that $h(x_1) = h(x_2)$.
3. *Collision resistance.* It is difficult to find two distinct values x_1 and x_2 such that $h(x_1) = h(x_2)$. Pair (x_1, x_2) is called a collision.

Collision resistance implies second pre-image resistance, while reduction from pre-image resistance to collision resistance is proved impossible [7]. In practice, collision always exists because of the fixed and limited output size of hash. Since the meaning of "difficult" here is infeasibility of breaking these properties in polynomial time, the three resistances indicate that there is no method for an adversary to modify a value without change of its hash value in a computational way.

Practically, security of a practical cryptographic hash algorithm can be estimated with its output size as a security parameter. A hash algorithm can be considered computationally secure when the possibility of successful attack in polynomial time is negligible. For a hash algorithm with k-bits output size, the computational complexity of brute force attacks to find a collision is $O(2^{k/2})$. Computational safety could be satisfied when k is large enough.

However in the history of cryptography, most hash algorithms suffer potential attacks, and breakages of them within certain amount of time are possible. Security of a hash algorithm does not fall abruptly, but will happen in a comparatively smooth process. In general, collision resistance is the relatively weakest part to be broken, followed by pre-image and second pre-image resistance. For example, widely-used hash algorithms MD5 and SHA-1 have turned out to be vulnerable to collision attacks – In 2005 Wang et al. firstly proposed efficient attacks to find collision for full version of MD5 within 2^{32} operations [8], and SHA-1 within 2^{69} operations [9] rather than 2^{80} for brute-force attack. The complexity of collision attack for SHA-1 is reduced to 2^{63} in later research. Until the first collision for full SHA-1 has been found under a practical attack by Google in 2017 [10], SHA-1 survives for over 12 years from proposal of the first attack

in theory. Besides cryptanalysis on particular algorithm, quantum computing like Grover's fast quantum mechanical algorithm for pre-image attack [11], will accelerate the process of hash compromise as well.

2.3 Impact of Compromised Hash

Giechaskiel et al. have discussed potential impact of compromised hash algorithms used by Bitcoin [12]. In Bitcoin protocol, compromised hash algorithms results in steal of coins, double-spend and complete failure of the blockchain. Here we apply and extend their research into general proof-of-work blockchain protocol. We summary the impact in the aspect of mining, Merkle Tree and signature as follows.

Mining. Since the pre-image for a given hash value can be found through pre-image attack, miners can easily mine a block whose hash value less than target of proof of work. Thus split from main chain will be easier to happen. Moreover, an malicious adversary can use second pre-image attack or collision attack to mine two blocks with the same hash value. If the adversary has a sufficient control of network, he can transmit these two blocks into different network respectively. Since nodes always accept the first one they received and reject the latter, partition of the entire network consequently occurs and reverse of transactions become possible. In such case, blockchain protocol will fail to work.

Merkle Tree. Merkle Tree is constructed from transactions, and root of it will be put into block header for simplified verification. Generally, when computing a Merkle Tree with transactions, miners can include their own transactions with arbitrary data filled in some certain fields, e.g. input of *coinbase* transaction for Bitcoin [5,13] and *data* field in transactions for Ethereum [14]. With the help of these field, there may be enough bytes space for an adversary to launch second pre-image attack, to construct a distinct Merkle Tree with the same root value as an existing one. Therefore even a confirmed block is possible to be altered. The adversary can transmit the altered block to newly joined nodes and lead to failure of them to reach agreement with the network. Finally failure of the blockchain happens as well. Similar attack strategy can be applied to other specific fields within block structure, like the stored hash value of previous block.

Signature. Signature is computed based on hash of transaction in blockchain. If the used hash algorithm is compromised, two transactions as the hashed messages are possible to be found for a single hash value. The adversary can create a transaction with the same hash value as a on-chain transaction, such that these two transactions can both be acknowledged by network. For cryptocurrency like Bitcoin, as a result coins will possibly be stolen. Ownership of other data secured by signatures can also be broken.

3 Existing Solution to Compromised Hash in Blockchain

In this section we give an introduction and analysis of M. Sato et al.'s method [4] that applies proof of existence model with a decentralized manner into blockchain to resolve the problem of compromised hash.

3.1 Model of Proof of Existence

Sato et al. are the first to give the definition of proof of existence to describe validity of transactions in blockchain [4].

Definition 1 (Proof of Existence). *Supposing d is the data needed to be verified, we have the following definitions for d:*

- *poe_d: the proof of existence for d, which is calculated from d.*
- *$vpoe_d$: the data required for verification of poe_d.*
- *v: the verification algorithm outputing **true** only when input d and $vpoe_d$ are both valid, otherwise outputing **false**.*
- *A time-wise order can be obtained when poe_d is produced from d.*

Under centralized system poe_d is the signature produced by public key certificate from a trustful authority, while in decentralized scenario poe_d is determined by consensus of the whole network.

Proof of Existence for Blockchain. Under the model of proof of existence, we give a description of validity of transactions in blockchain. At first we define the following terms for blockchain:

- b_i: i-th block.
- tx_{ij}: j-th transaction in b_i ($1 \leq j \leq N_i$ where N_i is the number of transactions in b_i).
- hb_i: The reference of previous block b_i stored in b_{i+1}, i.e. hash value of b_i.
- $txid_{ij}$: The transaction ID of tx_{ij}, i.e. hash value of tx_{ij}.
- H_1: hash algorithm in use.

b_i is constructed as follows:

$$b_i = [hb_{i-1}, mkroot_i, [tx_{i1}, \ldots, tx_{iN_i}]]$$

where $mkroot_i$ is root of Merkle Tree calculated from $txid_{i1}, \ldots, txid_{iN_i}$.

Here transaction $tx_{ij} (1 \leq j \leq N_i)$ is the data d needed to be verified, hb_i is the proof of existence poe_d for d and b_{i-1} serves as $vpoe_d$ for poe_d. The verification algorithm v to check if tx_{ij} is existent is as follows:

1. Check d: calculate Merkle Root from $txid_{i1}, \ldots, txid_{iN_i}$ and check if it is equal to $mkroot_i$.
2. Check $vpoe_d$: calculate $H_1(b_{i-1})$ from $b_{i-1} (= vpoe_d)$ and check if it is equal to hb_{i-1} in b_i.
3. Calculate $H_1(b_i)$ and check if it is equal to hb_i in b_{i+1}.

3.2 Transition Scheme on Blockchain

Long-term signature scheme is a secure transition method originally designed for PKI (Public Key Infrastructure) model [15,16]. It extends validity of digital signature and keeps a time-wise order of data relying on valid timestamps signed by a centralized trustful authority from PKI. Long-term signature scheme aligns model of proof of existence.

Sato et al. apply similar concept to secure transactions in the case of compromised hash in blockchain [4]. Their method can keep a block-wise order of transactions without participation of a trusted third party. Here we give a brief description of the method.

In proof-of-work blockchain once H_1 is compromised, poe_d will become an invalid PoE (proof of existence) for d because the mathematic relationship between them is broken. To deal with this problem, the transition scheme creates new PoE with a more secure hash algorithm H_2, to make verification of transactions available again. There are two ways to implement this method - basic transition procedure and support chain transition procedure.

Assuming that transition scheme starts from b_{M+1}, then we divide historical blocks $b_i (1 \leq i \leq M)$ into r groups of s blocks. Let b'_{M+k} be the new block constructed with H_2. Generation of new PoE poe'_d in basic transition procedure is as follow.

1. Calculate archive hash

$$archiveHash_k = H_2(b_{(k-1)s+1}, b_{(k-1)s+2}, \ldots, b_{(k-1)s+s})$$

2. Calculate new transaction ID $txid'_{(M+k)j}$ from $tx_{(M+k)j}$ with H_2

$$txid'_{(M+k)j} = H_2(tx_{(M+k)j})$$

3. Build Merkle Tree from $txid'_{(M+k)j} (1 \leq j \leq N_{M+k})$ with H_2 and set $mkroot'_{M+k}$ as the Merkle Root
4. Calculate hb'_{M+k-1} as follows:

$$hb'_{M+k-1} = H_2(b'_{M+k-1})$$

when $k = 1$, $hb'_{M+k-1} = H_2(b_M)$

5. Construct b'_{M+k}

$$b'_{M+k} = [archiveHash_k, hb'_{M+k-1}, mkroot'_{M+k}, [tx_{(M+k)1}, \ldots, tx_{(M+k)N_{M+k}}]]$$

6. Calculate new PoE

$$poe'_d = H_2(b'_{M+k})$$

where d is the collection of all transactions from $b_{(k-1)s+1}, \ldots, b_{(k-1)s+s}$ and b'_{M+k}.

After generation of b'_{M+k} for $1 \leq k \leq r$, each transaction in b_i for $1 \leq i \leq M$ will have corresponding new PoE protected by H_2, and the further blocks store newly collected transactions only, but not archive hash.

As mentioned in Subsect. 4.4.2 of [4], the above basic transition procedure is actually a hardfork since it inserts archive hash for past blocks into future blocks (that is, introducing new block structure into original blockchain protocol). Hardfork requires all nodes to follow new blockchain protocol to contribute computation but it always brings disagreement to blockchain community. The possible split can not be eliminated completely even if the computational power that still supports the original blockchain is very small. As a result, it will be divided into two blockchains. In example of Ethereum, after the DAO attack [17,18], a hardfork was processed from original chain to reverse state at the approval of majority of community members, but at the support of a minority of computational power the original chain still survived known as Ethereum Classic.

For the support chain transition procedure given in [4], new PoE poe'_d is stored in block from a second chain called support chain, which is maintained by the same or a part of miners of original chain, while block structure of original chain stays the same as before. As mentioned in Subsect. 4.4.3 of [4], proof-of-work competition is applied to one of these two chains, and both chains store the same transactions after completion of all archive hashes. Transactions verification is only conducted in original chain for most of time, but when dispute arises support chain will serve for final verification. This is a way of external protection. However, if proof of work is only applied to support chain, then proof-of-work mechanism in original chain should be removed. This is equal to introduction of hardfork into original chain, which results in the same issue as basic transition procedure. On the other hand, if proof of work is only applied to original chain, then block generation in support chain is not a computationally hard problem any more. Consequently, blocks in support chain will have no protection of accumulated computation. Even at some point where support chain has been maintained to be a very long chain, it is still possible for an adversary to take attacks on the compromised hash to replace blocks in original chain and then generate new corresponding support chain.

4 A Secure Approach Against Compromised Hash in Blockchain

In this section, based on the above proof of existence model, we introduce two-layer proof-of-work framework and then propose a softfork transition approach rather than a hardfork way, to provide a more feasible, practical and easier-to-deploy solution to the challenge of compromised hash algorithm in proof-of-work blockchain. Our scheme conforms to the model proposed by Garay et al. for secure blockchain backbone protocol [19].

4.1 A Novel Transition Scheme

To keep a proof-of-work blockchain secure, a transition from the compromised hash algorithm H_1 to a more secure one H_2 should be processed together with generation of new PoE. We define the following terms about proof of work with H_1 for block b_i:

1. $target_i$: current target for mining b_i.
2. $nonce_i$: the field that can be filled with random value to meet target requirement for b_i.
3. tsp_i: timestamp stored in b_i

When considering proof of work, b_i is constructed as follows:

$$b_i = [hb_{i-1}, mkroot_i, [target_i, nonce_i, tsp_i], [tx_{i1}, \ldots, tx_{iN_i}]]$$

In the following part, we let $target'_i, nonce'_i$ and tsp'_i represent corresponding parameters for proof of work with H_2. Adjustment rule of $target'_i$ can be set to the same algorithm as original blockchain protocol.

We assume that our scheme starts from b_{M+1}. Past M blocks will be divided into r groups of s blocks each, and we let $b_{M+k}(k \geq 1)$ be the block of new version. The new PoE is generated as follows:

1. Solve out H_1 puzzle: collect transactions and mine a complete block b_{M+k} that satisfies target of proof of work with H_1, i.e. using H_1 construct

$$\begin{aligned} b_{M+k} = [&hb_{M+k-1}, mkroot_{M+k}, \\ &[target_{M+k}, nonce_{M+k}, tsp_{M+k}], \\ &[tx_{(M+k)1}, \ldots, tx_{(M+k)N_{M+k}}]] \end{aligned}$$

   to subject to

$$H_1(b_{M+k}) \leq target_{M+k}$$

2. Calculate archive hash

$$archiveHash_k = H_2(b_{(k-1)s+1}, b_{(k-1)s+2}, \ldots, b_{(k-1)s+s})$$

3. Calculate transaction ID $txid'_{(M+k)j}$ from $tx_{(M+k)j}$ using H_2

$$txid'_{(M+k)j} = H_2(tx_{(M+k)j})$$

4. Calculate Merkle Root $mkroot'_{M+k}$ from $txid'_{(M+k)j}(1 \leq j \leq N_{M+k})$ using H_2
5. Calculate hb'_{M+k-1} as follows:

$$hb'_{M+k-1} = H_2(b'_{M+k-1})$$

   where $hb'_{M+k-1} = H_2(b_M)$ for $k = 1$.

6. Construct outer block ob_{M+k}.

$$ob_{M+k} = [hb'_{M+k-1}, mkroot'_{M+k}, [target'_{M+k}, nonce'_{M+k}, tsp'_{M+k}]]$$

7. Construct the new block by

$$b'_{M+k} = [b_{M+k}, ob_{M+k}, archiveHash_k]$$

8. Solve out H_2 puzzle: mine b'_{M+k} to make it meet H_2 target requirement, i.e. fill $nonce'_{M+k}$ with random data such that to

$$H_2(b'_{M+k}) \leq target'_{M+k}$$

9. New PoE is generated as

$$poe'_d = hb'_{M+k} = H_2(b'_{M+k})$$

where d are transactions from $b_{(k-1)s+1}, \ldots, b_{(k-1)s+s}$ and b'_{M+k}.

Construction of outer block ob_{M+k} and new PoE hb'_{M+k} are respectively shown in Figs. 1 and 2.

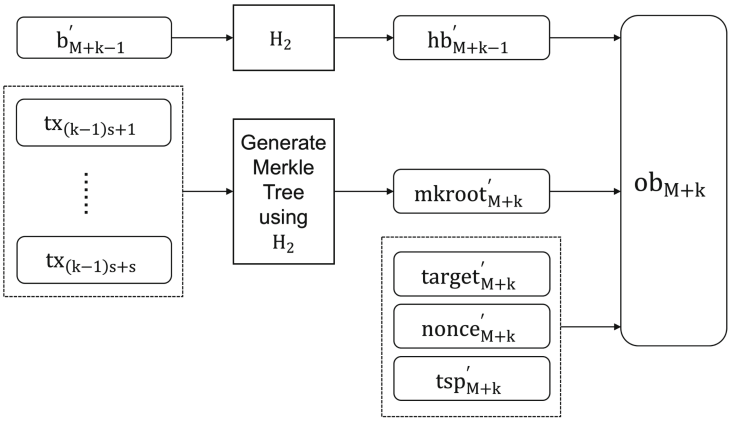

Fig. 1. Construction of outer block ob_{M+k}

Protection under H_2 of all historical transactions from $b_1, \ldots, b_M$ will be completed when b'_{M+r+1} is mined. For block $b'_{M+k}(k > r)$, new PoE is processed at the same way except to set $archiveHash_k$ empty. Chain structure of our solution is shown in Fig. 3. Since two kinds of proof of work for the hash algorithms H_1 and H_2 respectively both need to be solved out, we call the construction of the chain structure a *two-layer proof-of-work framework*.

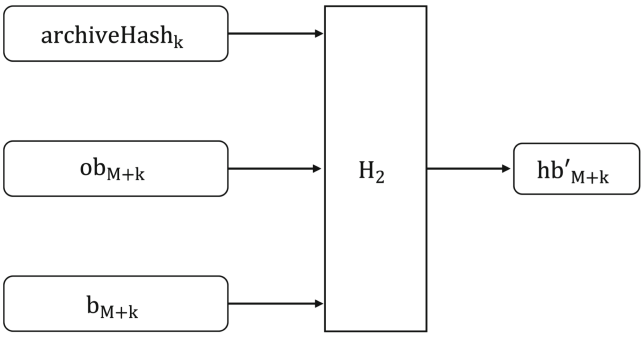

Fig. 2. Construction of new PoE hb'_{M+k}

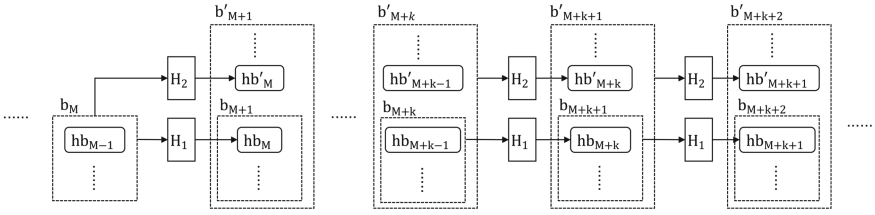

Fig. 3. Chain structure in our scheme

4.2 Verification Procedure in Blockchain

All participants in network can verify the validity of transactions when our scheme is implemented successfully. For nodes who run our upgraded blockchain protocol, as poe'_d is the new proof of existence for transactions, the algorithm v to verify $hb'_{M+k}(= poe'_d)$ is shown below.

1. Check transactions $(=d)$
   (a) if $1 \leq k \leq s$, then calculate $archiveHash_k$ from historical blocks $b_{(k-1)s+1}, \ldots, b_{(k-1)s+s}$

$$archiveHash_k = H_2(b_{(k-1)s+1}, b_{(k-1)s+2}, \ldots, b_{(k-1)s+s})$$

   otherwise set $archiveHash_k$ empty. And check if this $archiveHash_k$ is equal to the stored $archiveHash_k$ in b'_{M+k}
   (b) Calculate $txid'_{(M+k)j}$ from $tx_{(M+k)j}(1 \leq j \leq N_{M+k})$

$$txid'_{(M+k)j} = H_2(tx_{(M+k)j})$$

   (c) Calculate root of Merkle Tree from $txid'_{(M+k)j}, \ldots, txid'_{(M+k)N_{M+k}}$ and check if it is equal to $mkroot'_{M+k}$ in b'_{M+k}
2. Check proof of work and $b'_{M+k-1}(=vpoe_d)$:
   (a) Calculate $H_1(b_{M+k})$ and check if it meets the proof-of-work target for H_1 specified by $target_{M+k}$

(b) Calculate $H_2(b'_{M+k})$ and check if it meets the proof-of-work target for H_2 specified by $target'_{M+k}$
(c) Check if $H_2(b'_{M+k-1})$ is equal to hb'_{M+k-1} in b'_{M+k}
3. Check if value of $H_2(b'_{M+k})$ is the same as hb'_{M+k} in b'_{M+k+1}.

These upgraded nodes consider that a complete block is formed with b_{M+k}, ob_{M+k} and $archiveHash_k$. But in the view of old nodes that still run original blockchain protocol, they see b_{M+k} as a valid block, and ob_{M+k} and $archiveHash_k$ are transparent data to them. Therefore our scheme is a softfork strategy backward-compatible with original blockchain protocol. When verifying transactions, old nodes run the same verification algorithm as before, and do not check proof of work and Merkle Tree calculated with H_2.

4.3 Security Analysis

The key point to process our scheme safely is the implementation of upgraded blockchain protocol with majority computational power. In order to prevent the possible split, our transition scheme should be activated at a certain safe threshold of supported hast rate - that is, the dominant computational power should be migrated from original proof of work with H_1 to the two-layer proof of work with H_1 and H_2. Miners that still stay at original blockchain protocol recognize b_{M+k} within block of new version as a valid block, but upgraded nodes do not acknowledge the block generated by non-upgraded miners (i.e. b_{M+k}) as a complete block and will not follow it to mine further blocks. Therefore, according to the longest chain rule, branch generated only by non-upgraded miners will be given up by majority computational power switching to upgraded blockchain protocol. As for non-miners or light nodes, they do not contribute computational power and thus can still run original blockchain protocol.

Next, we consider the scenario where the adversary maliciously tries to mine an alternative chain with block of new version, to overtake the honest chain. Under hypothesis of rational man, miners always chase and extend the longest chain in order to maximize the benefits. Once the malicious branch outweighs the honest chain, rational miners will follow the malicious branch as the longest chain and abandon honest chain. As a result, blocks mined by honest miners will be overthrown and the adversary can benefit from it, e.g. taking back coins from transactions in honest chain. The probability of success for such an attack is related to the proportion of computational power that the adversary controls in whole network. To calculate out the probability, we first define the following terms.

- p_1 and q_1 are probabilities that honest miners and the adversary solve out H_1 puzzle respectively.
- p_2 and q_2 are probabilities that honest miners and the adversary solve out H_2 puzzle respectively.
- q_z is probability that the adversary overtakes honest chain from z blocks behind.

Then we can know that the relative probability for honest miners to find next block is $p = \frac{p_1 p_2}{q_1 q_2 + p_1 p_2}$ and for the adversary is $q = \frac{q_1 q_2}{q_1 q_2 + p_1 p_2}$. According to calculations in [1], we can compute probability P that the adversary overtakes the honest chain by the following formula:

$$P = 1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} (1 - (\frac{q}{p})^{(z-k)})$$

Table 1. Solution to P less than 0.1% for different proof-of-work consensus, where q_1 and q_2 are probabilities that an adversary solves out H_1 and H_2 puzzle respectively, and z is the minimum number of blocks behind where an adversary can not generate an alternative chain overtaking the honest chain with probability more than 0.1%.

q_1	q_2	z	q_1	q_2	z
0.10	0.10	2	0.30	0.10	4
	0.20	3		0.20	5
	0.30	4		0.30	8
	0.40	4		0.40	12
	0.50	5		0.50	24
0.15	0.10	3	0.35	0.10	4
	0.20	3		0.20	6
	0.30	4		0.30	10
	0.40	6		0.40	17
	0.50	8		0.50	41
0.20	0.10	3	0.40	0.10	4
	0.20	4		0.20	7
	0.30	5		0.30	12
	0.40	7		0.40	26
	0.50	11		0.50	89
0.25	0.10	3	0.45	0.10	5
	0.20	5		0.20	9
	0.30	6		0.30	17
	0.40	9		0.40	43
	0.50	15		0.50	340

(a) Our Two-layer Proof of Work

q_1	z
0.10	5
0.15	8
0.20	11
0.25	15
0.30	24
0.35	41
0.40	89
0.45	340

(b) Original Proof of Work with H_1

The result of solving out P less than 0.1% for our scheme is shown in Table 1a. Compared with that for original proof of work with H_1 presented in Table 1b, we can see that two-layer proof-of-work framework in our scheme provides higher security since fewer blocks can reach the same security level of preventing chain from being reversed on the occasion that H_1 is risky to be compromised but still computationally secure.

Moreover, when complete break of second pre-image resistance or collision resistance for H_1 happens, transactions and blocks can not be modified under

protection of H_2. In the case that pre-image resistance for H_1 is totally broken, i.e. x can be found in polynomial time for a given $h(x)$, solving out H_1 puzzle is not any more a hard job, but proof of work with H_2 and the majority computational power switching to the upgraded blockchain protocol can serve as the main guarantee to prevent malicious split and reverse of transactions.

5 Conclusion

This paper aimed to propose a secure solution against compromised hash in proof-of-work blockchain. Firstly, we analyzed the known approach given by M. Sato et al. which presented two transition procedures, i.e. the basic transition procedure and the support chain transition procedure. For the basic transition procedure, it is actually a hardfork which is difficult to be implemented since it always brings disagreement into blockchain community, while in the different case of support chain transition procedure, we observed that hardfork would also be introduced or support chain is risky to be tampered.

Then, we proposed the first softfork scheme against compromised hash in proof-of-work blockchain - a novel transition scheme with two-layer proof-of-work framework from the compromised hash to a secure one. Unlink hardfork solution, our scheme is backward-compatible such that potential splits from original chain can be avoided when the majority of computational power of the blockchain network migrates to our scheme. Furthermore, in our analysis we presented that a proof-of-work blockchain with our scheme can provide much more security than the original protocol in the aspect of resistance of malicious split attack.

Acknowledgement. We would like to thank the anonymous reviewers for their helpful feedback. The authors are supported by the National Natural Science Foundation of China (Grant No. 61672347, 61572318, 61672339).

References

1. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. In: Consulted (2008)
2. Bitcoin Wiki: Contingency plans - SHA-256 is broken (2015). https://en.bitcoin.it/wiki/Contingency_plans#SHA-256_is_broken
3. Bitcoin Wiki: Hardfork (2017). https://en.bitcoin.it/wiki/Hardfork
4. Sato, M., Matsuo, S.: Long-term public blockchain: resilience against compromise of underlying cryptography. In: IEEE European Symposium on Security and Privacy Workshops, pp. 1–8 (2017)
5. Okupski, K.: Bitcoin Developer Reference Working Paper, 30 June 2016. http://enetium.com/resources/Bitcoin.pdf
6. Bitcoin Developer Guide. https://bitcoin.org/en/developer-guide#proof-of-work
7. Stevens, M., et al.: Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 55–69. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_4
8. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_2

9. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_2
10. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 570–596. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_19
11. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of ACM Symposium on the Theory of Computing, pp. 212–219 (1996)
12. Giechaskiel, I., Cremers, C., Rasmussen, K.B.: On bitcoin security in the presence of broken cryptographic primitives. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9879, pp. 201–222. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45741-3_11
13. Bitcoin Wiki: Coinbase (2018). https://en.bitcoin.it/wiki/Coinbase
14. Wood, G.: Ethereum: a secure decentralized generalised transaction ledger. https://ethereum.github.io/yellowpaper/paper.pdf
15. European Telecommunications Standards Institute (ETSI): Electronic Signatures and Infrastructures (ESI); CAdES digital signatures; Part 1: Building blocks and CAdES baseline signatures. EN 319 122-1 V1.1.1 (2016)
16. International Organization for Standardization (ISO): Processes, data elements and documents in commerce, industry and administration - Long term signature profiles - Part 1: Long term signature profiles for CMS Advanced Electronic Signatures (CAdES). ISO 14533-1:2014 (2014)
17. The DAO, The Hack, The Soft Fork and The Hard Fork. https://www.cryptocompare.com/coins/guides/the-dao-the-hack-the-soft-fork-and-the-hard-fork/
18. Buterin, V.: Critical update re: DAO vulnerability, 17 June 2016. https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/
19. 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). https://doi.org/10.1007/978-3-662-46803-6_10