POSTER: Smart Contract-based Miner Registration and Block Validation

Shijie Zhang
Protocol Engineering Lab., Sangmyung University
Cheonan, Republic of Korea
zhangshijie@pel.smuc.ac.kr

Jong-Hyouk Lee
Protocol Engineering Lab., Sangmyung University
Cheonan, Republic of Korea
jonghyouk@pel.smuc.ac.kr

ABSTRACT

Rewriting-history attacks are devastating for blockchain-based cryptocurrency systems. To deter such attacks, this paper introduces a new model utilizing a smart contract to effectively prevent such attacks from succeeding. Each node who creates a new block is required to register with the smart contract to get a voucher required for the subsequent block validation. We explain the flow of the proposed model and the algorithms of the smart contract. We have implemented this smart contract in the Solidity language and simulated on the Ethereum test network. We also present security and simulation results of the proposed model.

CCS CONCEPTS

• **Networks** → **Network reliability**; • **Computer systems organization** → **Peer-to-peer architectures**.

KEYWORDS

Blockchains; Smart contracts; Rewrite history; Solidity

ACM Reference Format:
Shijie Zhang and Jong-Hyouk Lee. 2019. POSTER: Smart Contract-based Miner Registration and Block Validation. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS '19), July 9–12, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3321705.3331010

1 INTRODUCTION

As a hot topic of the current research, the blockchain technology has attracted extensive attention. The anonymity, tamper-proof, and other cryptographic properties of the blockchain effectively secure a cryptocurrency system. However, critical security issues still exist in blockchain-based cryptocurrency systems. Attacks such as selfish mining in Proof-of-Work (PoW) [3], short/long-range attacks in Proof-of-Stake (PoS) [1], and double-spend attacks can rewrite the entire history of the blockchain, which will cause considerable damage to emerging blockchain-based cryptocurrencies. The designs and implementations of various mitigations are thus imperative. However, many reported mitigations are only theoretical, and some of them either reduce the performance of the blockchain or weaken the decentralization trait of the blockchain. In this paper, we propose a smart contract-based model that does not change the inherent properties of the blockchain, and we also present some algorithms that implement the smart contract.

Smart contracts are Turing-complete computer programs used to develop contracts within the blockchain, e.g., Ethereum [2]. Since no one can change the code or state of smart contracts with ease, the smart contracts guarantee the traceability and irreversibility of transactions in blockchain systems. Smart contracts currently have many application scenarios, e.g., finance, digital identity, and supply chain, etc., but there is no research on using the smart contracts to prevent attacks that rewrite the blockchain history. In this paper, we leverage security properties of a smart contract to design a smart contract-based miner registration and block validation model. Note that the miner in this paper represents the block creator. Our aim is to adopt this model to effectively detect and deter any malicious behaviors that create a private chain to rewrite the blockchain history. The smart contract in our model has been implemented by the Solidity language v0.5.0 and is available on GitHub[1].

The rest of the paper is organized as follows: Section 2 presents the security issue and the threat model. Section 3 introduces the flow of our proposed model. Section 4 conducts the security analysis. Section 5 shows our simulation results. In Section 6, we conclude this paper.

2 SECURITY ISSUE STATEMENT

Security threat: A number of attacks focusing on blockchain consensus protocols aims at rewriting the blockchain history. For instance, double-spend attackers withhold the blocks they mined and secretly create their private chain to rewrite the history, which erases the valid transactions in the main chain, making the double-spend transactions valid. In PoW-based systems, the attackers can collude with other malicious players to form a mining pool to facilitate the success of rewriting the history [3]. In PoS-based systems, no matter the attackers execute a short-range or long-range attack, due to the costless simulation, creating a private chain is feasible and does not require many computational power [4].

Threat model: We assume the rewriting-history attack occurs in the eventual-consensus protocol that adopts the longest-chain rule, e.g., PoW, some PoS such as Ouroboros [5]. We also assume that a set of adversaries $\mathcal{A}$ stealthily creates a private chain and tries to extend it to rewrite the blockchain history. The speed at which they create blocks in the private chain is initially slower than the main chain. However, $\mathcal{A}$ can exploit some means (e.g., launch a concurrent attack) to help them catch up with the main chain in a short period of time.

[1] https://github.com/clutchydsb/Miner-Registration

AsiaCCS '19, July 9–12, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-6752-3/19/07.
https://doi.org/10.1145/3321705.3331010

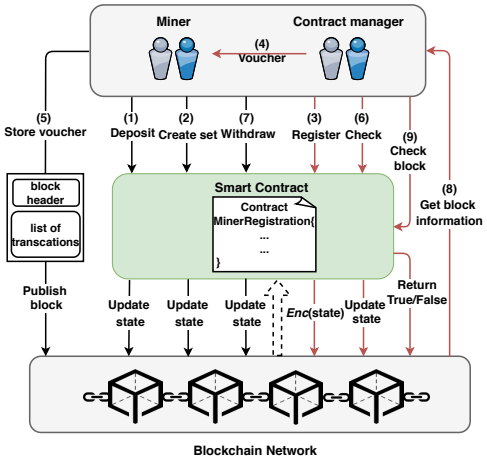

Figure 1: Main flow of the proposed model.

3 MODEL DESIGN AND FLOW

In this section, we introduce the main design idea of the smart contract-based model that can prevent the rewrite-history attack. Then, we shed light on the implementation procedures of this model in detail.

3.1 Design

Our design goal is to fundamentally invalidate the behaviors of rewriting the blockchain history, i.e., making the private chain created by $\mathcal{A}$ invalid. We need to let all the honest nodes realize that the chain created by $\mathcal{A}$ is a forged chain. We thus design a model that can differentiate if a block or a chain is valid. We leverage a smart contract that can conduct miner registration and block validation to fulfill our model. Before a new block is broadcast throughout the network, one miner who has created this block needs to register with the smart contract, and this contract can use some algorithms to output a unique voucher $\mathcal{V}$ for each miner who finishes the registration. Each miner stores $\mathcal{V}$ into the block he will publish. Hence, $\mathcal{V}$ in each block is a proof of the validity. Any block that does not contain $\mathcal{V}$ or has a forged $\mathcal{V}$ is considered as an invalid block.

3.2 Flow and algorithms

Before introducing the main flow of the proposed model, we present the following assumptions: Let $\mathcal{M}$ denote the miner who intends to register, C denote the miner registration and block validation smart contract, and C_m denote the node who deploys and manages this contract. We assume C_m is elected via voting by other honest nodes. Fig. 1 depicts the main flow of the proposed model.

(1) A stipulated amount of deposit needs to be charged for $\mathcal{M}$, which can be withdrawn after the registration if $\mathcal{M}$ does not have any malicious behaviors. Then C sends the updated state to the blockchain for confirmations. C creates two registration states "unregistered" and "registered" for $\mathcal{M}$.
(2) $\mathcal{M}$ calls the function of creating a transaction set in C. In this function, $\mathcal{M}$ submits the block height and a number of transaction hashes of his mined block to C. The purpose of submitting a series of transaction hashes is to generate $\mathcal{V}$ that is as random as possible.

Algorithm 1: Miner registration

```
1  blockheight ← a counter recording the block height
2  now ← timestamp of the latest block
3  height ← block height submitted by M
4  trans ← a set of transaction hashes submitted by M
5  logger ← a hash table for storing V
6  if height == blockheight + 1 then
7      index ← int(Sha256(now || trans[0])) % trans.length
8      voucher ← Sha256(now || trans[index] || index + 1)
9      logger[height][voucher] ← True
10     blockheight ← blockheight + 1
11     return voucher
12 end if
```

(3) The miner registration is performed by C_m. The algorithm of the miner registration is presented in Alg. 1. We first stipulate that the block height submitted by $\mathcal{M}$ in Step (2) must be the next block height of the current blockchain. This requirement limits the fact that $\mathcal{M}$ can only register at the next block height each time. Then C uses a *Sha*256 algorithm to randomly select one transaction hash from the created transaction set. After the first random algorithm, the transaction hash at *index* in the set is fetched, and C uses a *Sha*256 algorithm again to output the random value of it. We regard this random value as $\mathcal{V}$. The purpose of using the *Sha*256 algorithm twice is to reduce the predictability of $\mathcal{V}$ and deter some malicious miners from calculating $\mathcal{V}$ in advance. A two-dimensional mapping hash table entitled *logger* is used to store $\mathcal{V}$ at each block height. Lastly, C returns $\mathcal{V}$ to C_m. Moreover, since the transaction data and status update of the contract will be broadcasted throughout the network, the output $\mathcal{V}$ should be encrypted to prevent a malicious miner storing this valid $\mathcal{V}$ into his block and publishing it in advance of the block that should contain this $\mathcal{V}$. We thus assume only C_m has a key to decrypt the encrypted $\mathcal{V}$.
(4) C_m sends the decrypted $\mathcal{V}$ to $\mathcal{M}$ in a secure manner (e.g., SSH channel) to ensure this $\mathcal{V}$ is only visible to $\mathcal{M}$ at that time.
(5) After receiving $\mathcal{V}$, $\mathcal{M}$ stores it into the mined block and broadcasts this block throughout the network.
(6) C_m does another check in C that confirms if the current block height in the blockchain is increased by 1. If the current block height is not increased by 1, which indicates that $\mathcal{M}$ has not published his block, C will delete the record of $\mathcal{V}$ at that block height in *logger*. On the flip side, if $\mathcal{M}$ passes this check, the registration status of $\mathcal{M}$ turns to be "registered".
(7) Any $\mathcal{M}$ whose registration status is "registered" can withdraw the deposit invested before.
(8) Once a new block occurs in the network, C_m will first get some block information of this block, e.g., the stored $\mathcal{V}$, the previous block hash, the current block hash and the block height.
(9) C_m submits the information obtained in Step (8) to C for block validation. One valid block needs to meet the following conditions: the stored $\mathcal{V}$ can be found in *logger*, the stored $\mathcal{V}$ has not been verified before, and the previous block hash is valid.

Afterward, C announces the validation result "True" or "False" over the network, and any node can make a consensus based on this result. We present the algorithm of the block validation in Alg. 2.

Algorithm 2: Block validation

```
#height ← block height of the block being checked
#preHash ← previous hash of the block being checked
#currentHash ← current hash of the block being checked
#voucher ← V stored in the block being checked
verVoucher ← a hash table storing the verified V
verBlock ← a hash table storing the verified block hashes
if logger[#height][#voucher] && !verVoucher[#voucher]
   && verBlock[#preHash] then
    verVoucher[#voucher] ← True
    verBlock[#currentHash] ← True
    return True
else
    if #height == blockheight then
        blockheight ← blockheight − 1
    end if
    return False
end if
```

4 SECURITY ANALYSIS

In this section, we analyze whether $\mathcal{A}$'s possible threat behaviors can be successfully prevented by our smart contract-based model.

As shown in Fig. 2, the blocks represented in grey are mined by $\mathcal{A}$, and the blocks represented in blue are mined by the honest nodes. We suppose the current block height in the blockchain is H, thus the latest block in the main chain is at H, while the latest block in the private chain is at $H-1$. Afterward, $\mathcal{A}$ has four different options:

Case 1: $\mathcal{A}$ truthfully submits the necessary information to create a transaction set. However, the block height submitted by $\mathcal{A}$ is $H-1$, which does not satisfy the requirement in Step (3) (i.e., $\mathcal{A}$ must submit the height $H+1$ to C). The subsequent registration process will not be executed, therefore $\mathcal{A}$ cannot get $\mathcal{V}$ issued by C.

Case 2: $\mathcal{A}$ submits a bogus block height $H+1$ instead of $H-1$, which does not breach the requirement in Step (3). Unfortunately, after $\mathcal{A}$ successfully gets $\mathcal{V}$, C will discover $\mathcal{A}$'s malicious behavior that stealthily creates blocks by doing a check as Step (6). Then, C will delete the record of $\mathcal{V}$ originally belonging to $\mathcal{A}$ in *logger*.

Case 3: $\mathcal{A}$ can choose to evade registering with C. Hence, $\mathcal{A}$ cannot get any $\mathcal{V}$, which is the same as Case 1.

Case 4: It is the subsequent behaviors of $\mathcal{A}$ in Case 1 and 3. When $\mathcal{A}$ does not get any $\mathcal{V}$ issued by C, $\mathcal{A}$ will store a set of $\mathcal{V}$ from the valid blocks into the blocks of the corresponding block heights in the private chain. However, these valid $\mathcal{V}$ have been verified.

Combining the above four cases, the blocks before $H+1$ in the private chain cannot pass the block validation in C. If $\mathcal{A}$ successfully creates one block at $H+1$ in advance of the main chain, although $\mathcal{A}$ can obtain a valid $\mathcal{V}$, according to Step (9), C will check whether the previous block is valid. Therefore, the block at $H+1$ is proved invalid, resulting in the private chain released by $\mathcal{A}$ not being supported by the honest nodes. In general, the behavior of rewriting the blockchain history fails under our smart contract based model.

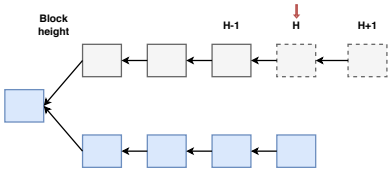

Figure 2: Overview of the attack scenario.

Table 1: Processing time of the smart contract.

Number of hashes	Average processing time
2	393ms
10	381ms
15	397ms
20	393ms

5 SIMULATION

We tested our smart contract on the Ethereum test network by using Truffle v5.0.11 and Ganache v1.1.0. We recorded the average processing time of the smart contract from the miner registration to the block check.

The simulation result is presented in Table 1. We can deduce that the number of submitted transaction hashes does not affect the processing time of our smart contract. Moreover, since the average processing time is less than 400ms, we suppose that our model does not bring a big network delay to influence the consensus process of the blockchain systems.

6 CONCLUSIONS

In this paper, we designed a smart contract-based protection model against rewriting-history attacks. It requires miners to register with a smart contract before publishing blocks. Each registered miner gets a valid voucher. Then, the proposed smart contract verifies the validity of blocks that appear in the network by checking the stored vouchers. Through analyzing the behaviors of the attackers under different cases, we demonstrated that the proposed model can make rewriting-history attacks invalid. In addition, we simulated the proposed smart contract on the Ethereum test network. We also provided the Solidity code that implements our smart contract on GitHub.

ACKNOWLEDGEMENTS

This research was supported by the MSIT, Korea, under the ITRC support program (IITP-2019-2018-0-01799) supervised by the IITP.

REFERENCES

[1] BitFury Group 2015. *Proof of Stake versus Proof of Work White paper.* BitFury Group. https://bitfury.com/content/downloads/pos-vs-pow-1.0.2.pdf.
[2] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* (2014).
[3] Ittay Eyal and Emin Gün Sirer. 2018. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM* 61, 7 (2018), 95–102.
[4] Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2018. Stake-bleeding attacks on proof-of-stake blockchains. In *Proc. of the Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, 85–92.
[5] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Proc. of Annual International Cryptology Conference (CRYPTO)*. Springer, 357–388.