The Trojan Message Attack on the Pay-to-Public-Key-Hash Protocol of Bitcoin

Maoning Wang, Meijiao Duan and Jianming Zhu

School of Information, Central University of Finance and Economics, 100081 China

Abstract. Bitcoin is the first and seemingly the most successful cryptocurrency based in a peer-to-peer network that uses blockchain technology. Given Bitcoin's growing real-life deployment and popularity, its security has aroused more and more attention in both financial and information industries. As a body containing a variety of cryptosystems, Bitcoin may also suffer from cryptanalysis attacks. This paper focuses on one of such attacks: the Trojan message attack, and presents in detail how to conduct the attack according to the structure and workflow of the Pay-to-Public-Key-Hash protocol of Bitcoin. The attack aims at forging an upcoming transaction record and results from the fact that all users' candidate input transactions are open to the attacker. The construction of the attack employs a combination of the Bitcoin transaction structure with standard Merkle–Damgard extension vulnerabilities. The conclusion of the attack shows that both the mathematical structure of the hash function itself and the public information in the blockchain are important to the security of Bitcoin. These factors should be considered in the future for the design of other cryptocurrency and blockchain systems.

Keywords: Bitcoin, Blockchain, Hash Fuction, Trojan Message Attack, Pay-to-Public-Key-Hash (P2PKH) Protocol

1 INTRODUCTION

Decentralized cryptocurrencies have generated considerable interest in recent years, as they enable users to "securely" transfer currency without the intermediation of a trusted authority. Bitcoin [1], the first distributed cryptocurrency system presented by Nakamoto in 2008, has laid the foundation for subsequent decentralized cryptocurrencies. Moreover, the technical essence of Bitcoin is a public data structure called a blockchain, which gathers transactions of currency in blocks. It is widely considered to have revolutionary application prospects, as blockchain's advantages lie in its decentralization, high efficiency, transparency, and low cost [2].

Concretely, Bitcoin is considered to be "secure", as transactions of currency are added-allowed-only to the blockchain. The consensus algorithm of Bitcoin guarantees that, for an attacker to be able to alter an existing block, the attacker must control the majority of the computational resources of the network [3]. Hence, attacks aiming at incrementing one's balance, e.g., by deleting transactions that certify payments to other users, are infeasible in practice. This security property is often rephrased by saying that the blockchain can be seen as an immutable data structure.

However, as an integration of a series of protocols, blockchain technology is confronted with challenges such as algorithmic security, protocol security, usage security, implementation security, and system security. Specially, because Bitcoin has been the most successful cryptocurrency so far, numerous research reports on threats to its security and security analysis have appeared. According to currently published research results, issues such as anonymity and privacy [4,5], adversarial miners [6–8], double-spending attacks in special scenarios [9], network splitting [10–12], flaws in instructions of the scripting language [13], and so on [14,15] have been discussed.

In addition to the abovementioned fields, another important aspect for researchers is based on the principle of cryptography, that is, treating the Bitcoin blockchain system as a system containing a variety of cryptoschemes and then applying cryptanalysis techniques to address and evaluate the system's security mechanism, i.e., to stand on an attacker's side for analyzing those cryptographic algorithms' structures, to find the relationships between the security strength of cryptographic schemes and the mathematical problems, to seek possible cryptosystem defects in a practical environment, to study the problems caused when cryptosystems are used overlappingly, and so on. Similar research methods have appeared in recent articles that focus on the security of many other systems containing cryptographic primitives, and even attack instances are constructed, such as a rogue CA certificate, colliding X.509 certificates for different identities, and forged PDF files, as given by Stevens et al. [16–18], and transcript collision attacks that break authentication in TLS, IKE, and SSH, as given by Bhargavan et al. [19], which extend the results given by Mavrogiannopoulos et al. [20] and Vaudenay [21]. Moreover, from the point of view of cryptanalysis, compared with the other systems mentioned above, the blockchain system in Bitcoin tends to be more vulnerable owing to the extensive application of various cryptographic techniques because any highly algorithm-intensive engineering system often brings more problems. Hence, it is necessary to apply such techniques of cryptanalysis once the security of Bitcoin is considered.

There have been numerous papers on cryptanalysis of Bitcoin and its successors. For example, Zheng et al. [22] discussed the elliptic curve digital signature algorithm (ECDSA) signature vulnerability of the current Bitcoin scheme when introducing a new backup scheme that can provide protection when an attacker manages to obtain the backup; Abusalah et al. [23] gave better time–memory tradeoffs based on Hellman's algorithm with applications to proofs of space (PoS), where PoS were suggested as a more ecological and economical alternative to proofs of work and are currently used in blockchain designs other than Bitcoin; Dinur et al. [24] constructed time–memory tradeoff attacks on the Merkle tree proof proof-of-work (PoW) scheme proposed by Biryukov et al. in 2016 [25]. Focusing on Bitcoin, for which hash functions are one of the most important foundations in ensuring its functionality and security, Giechaskiel et al. [26] specifically discussed the impact on the security of the entire Bitcoin blockchain system when the most classical hash function security criteria—preimage resistance, second preimage resistance, and collision resistance—were broken. Their conclusion is theoretically meaningful; however, it is not comprehensive enough: The three security criteria describe only the security strength of the hash functions themselves. For a hash function applied in an actual scene, in addition to its

own mathematical structure, its security also depends on the particular aspects of the environment in which it works, because, in general, the application scenarios tend to bring some restrictions, making it difficult for the hash functions to establish the necessary conditions of the security criteria.

In Bitcoin, constraints such as the specified data structure of the messages, working together with the ECDSA signature, the tree-like structure of the Merkle tree hash function, and other cryptographic components, make the hash functions' security no longer an intuitively true argument but an inconclusive problem that requires more rigorous and innovative theoretical analysis.

Hence, in this paper, we focus on such security problems faced by the hash functions in Bitcoin. Concretely, the Trojan message attack on the Pay-to-Public-Key-Hash (P2PKH) protocol of Bitcoin is constructed, relying on the interaction of the Bitcoin transaction structure with standard Merkle–Damgard extension vulnerabilities. Intuitively, the Trojan message attack reduces the complexity of stealing bitcoins from a generic second-preimage attack to a collision attack. To our knowledge, this is also the first concrete cryptanalysis attack construction in the circumstance of Bitcoin.

The structure of this paper is as follows: In Section 2, we introduce basic knowledge about Bitcoin's blockchain system, especially the data structure and the formation principle of the P2PKH transaction records involved in the attack proposed in this paper. In Section 3, we introduce the model of a Trojan message attack and detail how to construct such an attack based on Bitcoin's property that all the users' available input transactions are public. In Section 4, we give some conclusions about the above attack.

2 BACKGROUND

2.1 Basic Blockchain Principle

First, we introduce the basic workflow of the blockchain system in Bitcoin. The blockchain can be seen as a public log in which all transactions that have occurred are recorded in the form of blocks of these transactions. Specifically, each transaction uses scripting language to describe the owner of the Bitcoin involved in this transaction and is guaranteed by miners that only valid transactions can be included in the blockchain. To ensure that transactions that have occurred cannot be changed or removed, the miners will solve difficult problems to prove their workload.

Therefore, from the perspective of the transaction participants, the blockchain's workflow can be summarized as follows: The initiator of a transaction will write the transaction information in accordance with the specified data structure into a transaction record and then send the transaction record to the miner nodes; one of the miner nodes validates the transaction record by solving the difficult problem to complete the PoW to include the transaction record in the blockchain, forms a new block, and broadcasts the block to the entire network; the recipient of the transaction sees the blocks in the network and verifies the information in the block.

Simply and intuitively put, in Bitcoin the "mining" mechanism is employed to ensure that the concept of decentralization and the double-spend problem is addresses

through a distributed network and hash chain mechanism. In fact, instead of electronic coins, only the transaction records exist in the Bitcoin system. That is, the monetary value is dependent on the existence of the transaction records, and the change in the user amount of coins is substantially the change in the transaction records. At the same time, this also means that, in a digital currency system such as Bitcoin, a transaction record that contains money-transferring information between different users is an important part of the whole system's security.

2.2 Data Structure of P2PKH Transaction Records

Therefore, to analyze the security of the cryptographic components involved and to identify potential security problems, we focus on the data structure of the transaction records, particularly the parts relevant to the cryptographic components. In the following, we will introduce the structure of the transaction record of the P2PKH protocol, which is the most commonly used protocol in Bitcoin. Such a transaction record can be seen as a series of inputs and outputs. Inputs are some transactions that can be used by a user, that is, transactions indicating there are Bitcoins transferred into a user's account that have not been spent by the user. An output is the address to which the currency will be transferred. The following is the process of constructing a new transaction record using the P2PKH protocol, described in the Bitcoin source code [26, 27].

Input: source transaction record outputs o_n, destination addresses $addr$, and currency amount $value$ for each address to transfer into
Output: a transaction record

// indicate where the currency used in this transaction is from
For each source transaction record output o_n:
$pub, priv \leftarrow$ the public and private key pairs obtained according to information in o_n;
$sig \leftarrow Sign_{priv}(\text{selected parts})$;
$ScriptSig \leftarrow sig, pub$;
Add $(o_n, ScriptSig)$ to the input list of this transaction record;
End the loop
// indicate one or more addresses to receive the Bitcoins in this transaction
For each destination address $addr$:
$ScriptPubKey \leftarrow DUP\ HASH160\ addr\ EQV\ CHKSIG$;
Add $(value, ScriptPubKey)$ to the output list of this transaction record;
End the loop
Transaction record $T \leftarrow$ (version number, input list, output list, lock time)

By observing the definition of the class CTransaction in the Bitcoin source code [27], we can see that a transaction record should contain the following components:

Integer variable nVersion has 32 bits and stores the version number of the current system.

Array vin contains the source of the currency used in this transaction. The number of elements in the array is that of the previous transactions that the initiator selected as inputs in this transaction. Each element in the array is a CtxIn class object, occupying a 1344-bit space, in particular, that contains (1) a CoutPoint class object prevout, (2) a Cscript class object scriptSig, and (3) a 32-bit integer nSequence, the latter of which is used as a fixed identifier to show whether to enable the lock time.

The CoutPoint class object prevout occupies a 288-bit space and identifies the source transactions and outputs; it contains a 256-bit hash value used to identify the source transaction and a 32-bit integer used to indicate which output in that transaction, corresponding to the symbol o_n in the pseudo-code above. The Cscript class object scriptSig occupies 1024 bits of space and contains the signature and public key of the initiator. Because the digital signature algorithm used in the Bitcoin system is ECDSA (secp256k1 curve) and the public key adopts the uncompressed version, the signature and public key pubkey in scriptSig takes 512 bits of space each, respectively.

Array vout specifies the account address and the amount of Bitcoins to be transferred in this transaction. The number of the array's elements is that of the destination addresses. Each element in the array is a CtxOut class object that occupies 224 bits of space; specifically, it contains a 32-bit integer nValue and a script scriptPubKey. The form of the script scriptPubKey is as shown in the above pseudo-code: $DUP\ HASH160\ addr\ EQV\ CHKSIG$, which is used to parse the subsequent operation. Here, DUP, $HASH160$, EQV, and $CHKSIG$ are OP opcodes, where each opcode is a 8-bit char-type constant, and $addr$ is a public key address' 160-bit hash value; that is, the script scriptPubKey accounts for 192 bits of space totally.

Integer variable nLockTime occupies 32 bits of space and is usually set to 0. Only when the number representing the time that the transaction is broadcast to the network is greater than this value is the transaction record considered to be valid.

Figure 1 shows an example of the data format of the P2PKH protocol transaction record described above, where there is only one CtxIn class object in the array vin, which is quite a typical case in practice.

The process of generating a transaction record is the process in which an initiator completes going through the above pseudo-code as needed. It is worth noting that scriptSig in the source input array vin is not filled in at first. In fact, finally scriptSig should be the signature of a transaction initiator, and the signed message contains both the inputs and the outputs of the current transaction. Specifically, scriptSig is generated by the following process: First, the transaction information, i.e., the above nVersion, vin, vout, and nLockTime parts, will be filled out in the required data structure to generate a temporary transaction record, in which scriptSig in vin will be filled as a temporary value—scriptPubKey in this input source script to show that the source transaction was transferred to the current initiator's account address. Second, the transaction initiator calculates the hash value of the temporary transaction record and uses his private key to sign this temporary record, and then the signature and the transaction initiator's public key will be filled into the position where in the previous step scriptSig is put as a temporary value.

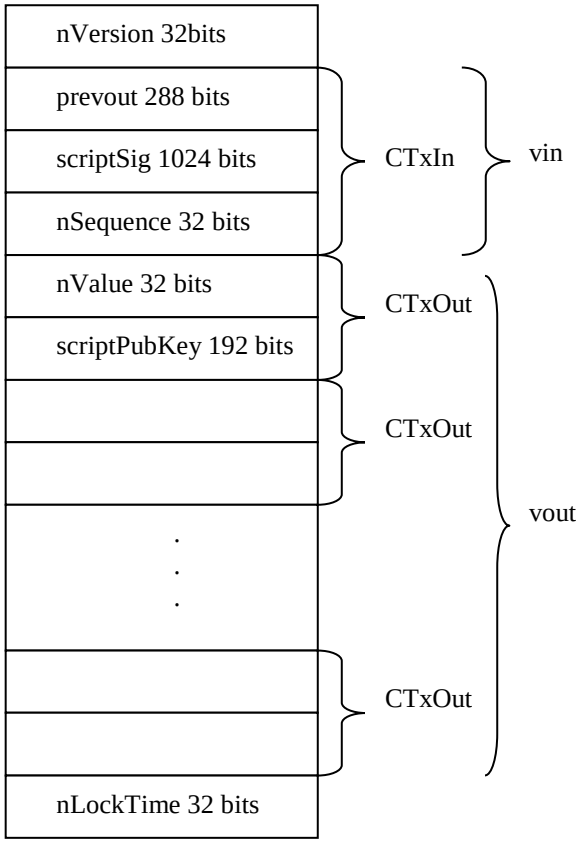

Fig. 1. Data format of Bitcoin's P2PKH protocol transaction record.

2.3 Data Structure of Blocks

After receiving the transaction records written by initiators, the miners will aggregate the transaction records collected from the entire network to form a public, global, and only-increasing allowed book, i.e., the blockchain. In this way, the currency is guaranteed not to be reused. Specifically, each block will contain a combination of all existing transaction records through the Merkle tree, and a new block forms part of the entire blockchain if it passes through the mining process, which is, one of the miners needs to find a random number *nonce* that makes the hash of a block's header less than a given target value, as expressed by the formula

$$hash(header \parallel nonce) < target,$$

wherein the block header *header* comprises the following parts:

Integer nVersion: version number, comprising 32 bits;

hashPrevBlock: hash value of the previous block, which indicates the identity of the block, comprising 256 bits;

hashMerkleRoot: hash value computed from all transaction records by the Merkle tree hash operation, comprising 256 bits;

nTime: timestamp, comprising 32 bits; and

nBits: upper bound target value, comprising 32 bits.

The mechanism of the process above is PoW; that is, the probability of generating a new block is proportional to the miner's computational power, and because the miners can get the transaction fee from this, they are hence motivated to perform these operations to confirm transaction information by generating valid blocks.

3 THE TROJAN MESSAGE ATTACK

3.1 Model of Trojan Message Attack

In 2009, Andreeva et al. proposed a new attack method for the hash function, the "Trojan" message attack [28]. The basic idea behind such attack is as follows: First, attacker A constructs in advance a Trojan message string and provides it to the victim V. V arbitrarily chooses a message prefix P from a limited set, cascades $P \| S$ to form a message, calculates the hash value $hash(P \| S)$, and sends it to A or to the public. Since the Trojan message is constructed by attacker A, if it can satisfy some specific properties, then A can successfully give a second preimage of the message.

Based on the above idea, we will show how to construct a Trojan message attack on the transaction record's data structure in Bitcoin's P2PKH protocol. It should be noted that this method will be used to generate new transaction records and blocks, rather than existing blocks. If the attack is successful, the attacker can make a transaction record in which the victim signs to transfer currency to some malicious address controlled by the attacker without the victim's knowledge, and, at the same time, the transaction record can also get through the miners' verification and hence the attacker can spend coins from it as a source input of subsequent transactions.

Our attack applies to the following scenario: Adversary A and user V reach an agreement that V will transfer Bitcoins to some designated legal accounts in a transaction, and user V uses the P2PKH-type protocol and chooses one of his own available source transactions as the input of the transaction; user V has N source transactions that can be used as an input. According to the working principle of the blockchain, the source transactions are publicly visible, but before the attack adversary A does not know which one user V will choose; adversary A cannot know the private key of user V, and, for a given message, the probability that adversary A can forge a signature is negligible.

Such a situation can occur easily as: (1) adversary A can be one of the participants in the Bitcoin blockchain system who have different accounts while others cannot tell which accounts in the whole system belong to A because of Bitcoin's pseudo-anonymity; (2) adversary A can communicate with user V so that user V agrees to transfer currency to some legitimate, visible addresses, fill in a transaction record, and then sign it; and (3) because the key management mechanism generally requires that

user V's private key be isolatedly saved, adversary A needs not to know the user's private key. These conditions are not hard to achieve for attacker A practically.

3.2 Steps of the Attack

The attack focuses on the step when calculating the hash value of the temporary transaction record before calculating the script scriptSig, and it will produce the effect that adversary A can construct another temporary transaction record with the same hash value, which has the same source input transaction and output amounts as the target temporary record but different output addresses. With this attack, adversary A can replace the output list of the transaction record and broadcast it to the network. Because the miner nodes do not confirm the information with the initiator during the verification of transaction records (and in fact the miner nodes only use the initiator's public key to verify whether the information received in the transaction records can be calculated according to the corresponding signature) and the information is first compressed by the hash function into a fixed length and then signed, the miner nodes would consider the modified transaction records to be valid as the two temporary transaction records have the same hash value. At the same time, the application scenario of the attack bypasses the conditions for the existential unforgeability of ECDSA, so that adversary A can get the signature of another message without knowing the private key of user V.

Similar to the Trojan message attack of Andreeva et al. [28], in our attack, we need to use an identical prefix collision search algorithm, which takes one intermediate value of the iteration hash function as an input parameter to produce a pair of messages that start with the intermediate value and compute the same hash output value.

If we denote by H^f the hash function SHA256's iterative compression function [29,30] and define $|x|$ as the length of bit string x, where the symbol $\|$ indicates concatenation of the bit strings, then the attack process of adversary A proceeds as follows:

(1) Fill user V's available transaction sources in the data structure of the temporary transaction record, which includes nVersion (32 bits), prevout (288 bits), and scriptSig (192 bits; actually, it is the script scriptPubKey to which a transaction's output *prevout* corresponds) and denote the filled information by $P^i, i = 1,2,\cdots,N$.

(2) Arbitrarily select n_0 addresses as the output addresses and amounts in the transaction, fill each of the n_0 address and amount information into the CtxOut class object, and denote this by S_0, which takes a space of $224 \cdot n_0$ bits such that $|P_i| + 224 \cdot n_0$ is an integer multiple of 512, as 512 is the block size of input to each SHA256's iterative compression function according to the Merkle–Damgard construction.

(3) Use H^f to calculate the iterated compression values $h_0^i, i = 1,2,\cdots,N$ for $P^i \| S_0, i = 1,2,\cdots,N$.

(4) Use the following method to calculate N pairs of messages $(S_i, T_i), i = 1,2,\cdots,N$ that can generate collisions:

For each $i=1$ to N

Construct candidate lists $List_S, List_T$;

Choose $S_i \in List_S, T_i \in List_T$ such that $H^{n_2,f}(h_{i-1}^i, S_i) = H^{n_2,f}(h_{i-1}^i, T_i)$;

For each $j=1$ to N

$h_i^j \leftarrow H^{n_2,f}(h_{i-1}^j, S_i)$;

The loop ends

The loop ends

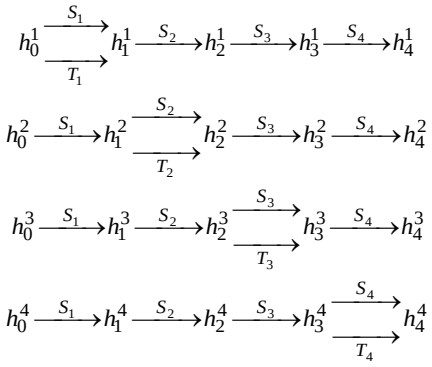

Fig. 2. Computation of each h_i^j.

Specifically, for each $i=1$ to N, first, when the candidate list $List_S$ is being constructed, each candidate S_i contains n_1 CtxOut class objects; that is, it contains n_1 legal output addresses and amounts $v_j, j=1,2,\cdots,n_1$, where n_1 satisfies that $224 \cdot n_1$ is an integer multiple of 512 (where we denote $\frac{224 \cdot n_1}{512} = n_2$, and n_2 is an integer, for example, if $n_1 = 16$ is selected, then $n_2 = 7$) and $\sum_{j=1}^{n_1} v_j < \frac{V_i}{N}$ (where V_i is the amount of Bitcoins available in the source input corresponding to P^i); second, when the candidate list $List_T$ is being constructed, each candidate T_i also contains n_1 CtxOut class objects, where the 160-bit address in one of them is controlled by attacker A and the other $n_1 - 1$ addresses are random legal ones, and the output amounts are also $v_j, j=1,2,\cdots,n_1$. After that, by using a sort-and-match technique, for each element T in $List_T$, calculate $H^{n_2,f}(h_{i-1}^i, T)$ and test whether there is an element S in $List_S$ such that $H^{n_2,f}(h_{i-1}^i, S)$ equals the value; if there are any, take out the pair of elements and denote them by (S_i, T_i), where the symbol $H^{n_2,f}(h_{i-1}^i, x)$ represents the hash value of message x computed by using H^f as the iterative compression function, h_{i-1}^i as the initial value, and n_2 as the number of iteration times; then, as shown

in Figure 2, taking S_i, which have just been found as the message, compute each $h_i^j \leftarrow H^{n_2,f}(h_{i-1}^j, S_i)$, in which h_i^{i+1} will be used for the next collision message search.

(5) For $i = 1,2,\cdots,N$, calculate the remaining amount that needs to be transferred to the last output address (that is, the address of user V) according to $P^i, S_0, S_1, \cdots, S_N$, nLockTime, the padded bits of hash function's requirements, and denote these as message block S_{N+1}^i. Calculate the final inner hash value $h_{N+1}^i = H^f(h_N^i, S_{N+1}^i)$ of the temporary transaction records by taking each P^i as the prefix message and the outer hash values $SHA256(h_{N+1}^i)$.

(6) Adversary A sends to user V a string of output addresses and amounts $S_0, S_1, \cdots, S_N$. Because these addresses are meaningful and legal, adversary A can make user V agree to transfer the Bitcoin currency to these target accounts.

(7) User V chooses an available transaction source (numbered i_0) and completes the filling in of the temporary transaction record, i.e., getting $P^{i_0} \| S_0 \| S_1 \| \cdots \| S_N \| S_{N+1}^{i_0}$, calculating its hash value with the function SHA256 (SHA256 (.)), and uses this hash value for subsequent operations including formal scriptSig signature generation, filling in the complete transaction record, and sending the final record to the network.

(8) After this transaction record sent by user V is collected into the blockchain by miners, adversary A can see the transaction record and obtains the information that user V has selected the transaction source of number i_0. At this time, the adversary can forge a new temporary transaction record $P^{i_0} \| S_0 \| S_1 \| \cdots \| T_{i_0} \| \cdots \| S_N \| S_{N+1}^{i_0}$. According to the above process, its signature is the same as the formal scriptSig signature sent by user V, and hence adversary A has obtained a forged transaction record. At this time, it is equivalent to say that user V has transferred coins into the address in T_{i_0}.

Figure 3 shows the whole flow of the attack:

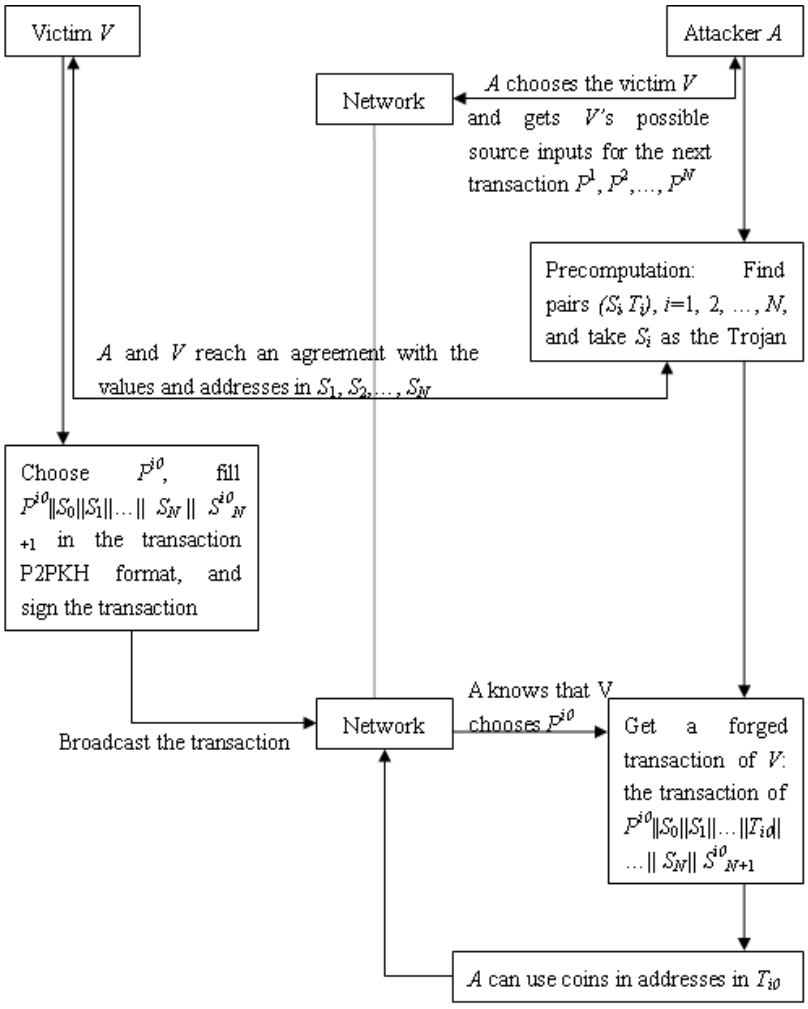

Fig. 3. Whole flow of the Trojan message attack.

3.3 Complexity of the Attack

In the following, we analyze the complexity of the attack method described above. It can be seen that, in the above attack, except for step (4), the other steps are direct steps such as filling and modifying that do not need additional calculations. Consequently, the entire attack requires nontrivial complexity for step (4) only; hence, we need to analyze the complexity of this step. According to the birthday attack principle (balanced case), for each $i=1$ to N, if we want to find a collision that satisfies the condition, the number of elements included in either of the candidate lists should be greater than $2^{n/2}$, for the compression function H^f of n-bit output length (where, in SHA256, $n=256$). That is, adversary A can select $|List_S|=|List_T|=2^{n/2}$, so the com-

plexity of the collision search phase is also $2^{n/2}$. In addition, in this step, also the computation $h_i^j \leftarrow H^{n_2,f}(h_{i-1}^j, S_i)$ is conducted N times; that is, it is an n_2 times of H^f compression function evaluation operation. In summary, the time complexity of the attack algorithm is $N \cdot 2^{n/2} + N^2 \cdot n_2 = O(N \cdot 2^{n/2})$.

It can be seen that the above attack constructs a new temporary transaction record that has the same hash value as the original temporary transaction record. Therefore, the above attack can also be regarded as a second preimage attack, for which the complexity $N \cdot 2^{128}$ is much lower than the security strength setting 2^{256} of the blockchain as the hash function used is SHA256.

Although the complexity seems to be high and impractical at present, the attack proposed still makes sense. The first reason is that cryptographic primitives usually break gradually, instead of abruptly breaking completely, as also mentioned by Giechaskiel et al. [26]. Theoretically, in cryptanalysis an attack is called successful if its complexity is less than that of a brute-force attack. Hence, as a second preimage attack with complexity of order 2^{128}, which is exponentially much less than 2^{256} (the complexity of a brute-force attack), the attack proposed in this paper is meaningful. The second reason is that no plan or adequate mechanism for such an attack has been built into Bitcoin or even considered for updates of Bitcoin in the future. Consequently, with the rapidly increasing computational power, it will be dangerous to leave a gap between the attack and current-version Bitcoin alone without deploying a proper treatment.

In addition, if there is a more efficient identical prefix collision search algorithm for the SHA256 function to be used in step (4), such as Stevens [16–18] mentioned, the complexity of such a collision search algorithm is 2^{16} for the hash function MD5, which also has a Merkle–Damgard iteration structure, and for the hash function SHA-1 in the same SHA series the complexity of this type of collision search algorithm is $2^{77.1}$, which is much lower than the security strength setting, so the attack method in this paper will be more effective. Another piece of evidence to indicate that such attacks might be potentially practical is the result shown by Mendel et al. [31] that collisions exist for 31/64-steps reduced number of rounds SHA256 with $2^{65.5}$ operations and the improvements shown by Kortelainen et al. [32] that Trojan message attacks can be conducted in more efficient ways.

4 CONCLUSIONS

In summary, we have constructed the steps for and studied the theoretical complexity of a Trojan message attack, the new kind of cryptanalysis attack to Bitcoin. Such an attack is based on the fact that all users' candidate input transactions are open to the attacker and employs a combination of the Bitcoin transaction structure with standard Merkle–Damgard extension vulnerabilities. Conditions for an attacker to launch the attack have been identified, revealing that they are not difficult to reach for malicious participants in Bitcoin's blockchain system. Because of the potential practicability and

harmful effects that the attack brings, appropriate and effective contingency plans for updating Bitcoin's P2PKH protocol are needed. Also, it is crucial to anticipate the impact of partial breakage or weakening of a cryptography primitive caused by data format and other public information in the blockchain when revising Bitcoin and designing other cryptocurrency systems in the future.

References

[1] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. 2008

[2] Bonneau J, Miller A, Clark J, et al. Sok: Research perspectives and challenges for Bitcoin and cryptocurrencies, 2015 IEEE Symposium on Security and Privacy, IEEE, 2015: 104–121.

[3] Garay JA, Kiayias A, Leonardos N. The Bitcoin backbone protocol: Analysis and applications, EUROCRYPT (2), 2015: 281–310.

[4] Miers I, Garman C, Green M, et al. Zerocoin: Anonymous distributed e-cash from Bitcoin, 2013 IEEE Symposium on Security and Privacy, IEEE, 2013: 397–411.

[5] Sasson EB, Chiesa A, Garman C, et al. Zerocash: Decentralized anonymous payments from Bitcoin, 2014 IEEE Symposium on Security and Privacy, IEEE, 2014: 459–474.

[6] Eyal I, Sirer EG. Majority is not enough: Bitcoin mining is vulnerable, International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2014: 436–454.

[7] Velner Y, Teutsch J, Luu L. Smart contracts make Bitcoin mining pools vulnerable, IACR Cryptology ePrint Archive, 2017, 2017: 230.

[8] Kwon Y, Kim D, Son Y, et al. Doppelganger in Bitcoin mining pools: An analysis of the duplication share attack, International Workshop on Information Security Applications, Springer, Cham, 2016: 124–135.

[9] Dmitrienko A, Noack D, Yung M. Secure aallet-assisted offline Bitcoin payments with double-spender revocation, Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security, ACM, 2017: 520–531.

[10] Gervais A, Ritzdorf H, Karame GO, et al. Tampering with the delivery of blocks and transactions in Bitcoin, Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, 2015: 692–705.

[11] Heilman E, Kendler A, Zohar A, et al. Eclipse attacks on Bitcoin's peer-to-peer network, USENIX Security Symposium, 2015: 129–144.

[12] Liao K, Katz J. Incentivizing blockchain forks via whale transactions, International Conference on Financial Cryptography and Data Security, Springer, Cham, 2017: 264–279.

[13] Bartoletti M, Pompianu L. An analysis of Bitcoin OP_RETURN metadata, Financial Cryptography and Data Security—FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017: 218–230, Revised Selected Papers, arXiv preprint arXiv:1702.01024, 2017.

[14] McCorry P, Shahandashti SF, Hao F. Refund attacks on Bitcoin's payment protocol, International Conference on Financial Cryptography and Data Security, Springer, Berlin, Heidelberg, 2016: 581–599.

[15] Bonneau J. Why buy when you can rent?, International Conference on Financial Cryptography and Data Security, Springer, Berlin, Heidelberg, 2016: 19–26.

[16] Stevens M, Lenstra AK, De Weger B. Chosen-prefix collisions for MD5 and applications, International Journal of Applied Cryptography, 2012, 2(4): 322–359.

[17] Stevens M, Sotirov A, Appelbaum J, et al. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate, Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology. Springer Verlag, 2009: 55–69.

[18] Stevens M. New collision attacks on SHA-1 based on optimal joint local-collision analysis, Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, Berlin, Heidelberg, 2013: 245–261.
[19] Bhargavan K, Leurent G. Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH, Network and Distributed System Security Symposium, NDSS 2016, 2016.
[20] Mavrogiannopoulos N, Vercauteren F, Velichkov V, et al. A cross-protocol attack on the TLS protocol, Proceedings of the 2012 ACM Conference on Computer and Communications Security, ACM, 2012: 62–72.
[21] Vaudenay S. Security flaws induced by CBC padding-applications to SSL, IPSEC, WTLS, EUROCRYPT, 2002, 2332: 534–546.
[22] Zheng Z, Zhao C, Fan H, et al. A key backup scheme based on Bitcoin. , IACR Cryptology ePrint Archive, 2017, 2017: 704.
[23] Abusalah H, Alwen J, Cohen B, et al. Beyond Hellman's time-memory trade-offs with applications to proofs of space, International Conference on the Theory and Application of Cryptology and Information Security, Springer, Cham, 2017: 357–379.
[24] Dinur I, Nadler N. Time-memory tradeoff attacks on the MTP proof-of-work scheme, Annual International Cryptology Conference, Springer, Cham, 2017: 375–403.
[25] A. Biryukov, D. Khovratovich. Egalitarian computing. In T. Holz and S. Savage, editors, 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10–12, 2016, pages 315–326, USENIX Association, 2016.
[26] Giechaskiel I, Cremers C, Rasmussen KB. On Bitcoin security in the presence of broken cryptographic primitives, European Symposium on Research in Computer Security, Springer International, 2016: 201–222.
[27] bitcoincore.org. Bitcoin core, https://github.com/bitcoin/bitcoin.
[28] Andreeva E, Bouillaguet C, Dunkelman O, et al. Herding, second preimage and Trojan message attacks beyond Merkle-Damgård, Selected Areas in Cryptography, 2009, 5867: 393–414.
[29] Menezes AJ, Van Oorschot PC, Vanstone SA. Handbook of Applied Cryptography, CRC Press, 1996.
[30] Standard SH. Federal Information Processing Standard Publication 180-2, US Department of Commerce, National Institute of Standards and Technology (NIST), 2002, csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice. pdf, 2002.
[31] Mendel F, Nad T, Schläffer M. Improving local collisions: New attacks on reduced SHA-256, Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, Berlin, Heidelberg, 2013: 262–278.
[32] Kortelainen T, Kortelainen J. On diamond structures and Trojan message attacks, International Conference on the Theory and Application of Cryptology and Information Security, Springer, Berlin, Heidelberg, 2013: 524–539.