Bitcoin Fast Payment Protocol Based on Chameleon Authentication Tree

Yuxiang Zhang[1], Ming Xian[1,*], Hongjiang Zhang[2], Huimei Wang[1], Congwang Kong[1]

[1]*College of Electronic Science and Technology, National University of Defense Technology, Changsha, Hunan China, 410000*
[2]*College of Electronics and Information Engineering, Ankang University, Ankang, Shaanxi China, 725000*

Abstract

With the development trend of distributed and decentralized, blockchain-based bitcoin payment methods are gradually gaining attention and research from academia and industry. In order to prevent the "double spend" attack in the Bitcoin payment process, it is officially recommended to wait for 6 blocks (about 1 hour) to confirm the transaction is established, which creates a serious time delay constraint for small payments and leads to inefficient payments. Therefore, this paper proposes a fast payment protocol based on the chameleon authentication tree, the main idea of which is to combat the "double spend" attack through a deposit penalty. The protocol associates the key in the scheme with the deposit key, enabling the implementation of fast payment protocols on distributed systems through deposit penalties.

Keywords

Bitcoin; Fast Payment; Chameleon Authentication Tree; Blockchain

1. Introduction

Bitcoin is the first decentralized cryptocurrency that maintains a publicly accessible and extremely difficult to tamper with a ledger of transactions in a decentralized manner, which is the blockchain. In Bitcoin, any participant can store a full backup of the blockchain, and all participants reach a consensus on the past blockchain history through Proof of Work, a consensus mechanism that prevents a small number of attackers from tampering with or deleting the transaction history. With the increasing development and popularity of blockchain technology, the application scenarios of bitcoin payments are becoming more and more widespread. We notice that there is also a huge demand for fast payment scenarios, such as: buying an item at a convenience store or a bottle of drink at a vending machine. The exchange time between bitcoin and goods in fast payment scenarios is extremely short (usually less than 30 seconds), and according to the characteristics of the bitcoin blockchain, a According to the nature of the Bitcoin blockchain, it takes at least 10 minutes for a transaction to be recorded on the blockchain through miner mining, and even then, the recipient is not theoretically able to fully identify the transaction at this point.

The Bitcoin developer team has implicitly acknowledged the problem of verifying fast payments, telling users that they don't need to wait to verify payments as long as the transaction is posted in the network[1]. Transactions that pass this validation are called Zero Confirmation Transactions, which do not exist in the blockchain and refer to the state of a transaction before it is broadcast to the entire network and is about to be packaged into a block. It refers to the state of a transaction before it is broadcast across the network and is about to be packaged into a block. The seller of a transaction delivers goods or services without waiting for the transaction to be confirmed by a node in the blockchain network, but the seller must trust that the buyer will not attempt to spend the same bitcoin elsewhere until the transaction is recorded by another node in the blockchain. While "zero-confirmation transactions" do increase the speed of transactions and adapt Bitcoin to fast payment scenarios, there

ICCEIC2022@3rd International Conference on Computer Engineering and Intelligent Control
EMAIL: *Corresponding author: xming@nudt.edu.cn (Ming Xian)

is a risk of "double-spend attacks" by malicious payers, as both the Bitcoin and Ether systems have been subject to. "Double-spend attacks.

To prevent "double spend attacks" Bitcoin officially recommends waiting for 6 blocks to be generated after the transaction block is completed to confirm the transaction, and due to the anonymity of the user, it is difficult to find the payer again after the quick payment for goods or services leaves the scene. Therefore, it is necessary and urgent to study the fast payment protocol for bitcoin. Based on the fact that the real micro-payment flow is becoming more and more intensive, and the requirements of both parties for transaction efficiency are getting higher and higher, studying a new fast payment protocol to make bitcoin transactions realize instant payment can make bitcoin transactions more and more convenient, and can allow some goods that need instant payment to be included in the scope of bitcoin payment, solving the blind spot caused by the payment delay This will solve the application blindness caused by payment delays.

2. Theoretical background

2.1. Bitcoin

Bitcoin is a distributed P2P payment system first proposed by Satoshi Nakamoto in 2008[2]. Each user first generates a local public-private key pair to create an account and uses the public key to generate the corresponding bitcoin address through the SHA256 hash function and RIPEMD160 hash function successively, and then through the Base58Check encoding, both sides of the transaction use the bitcoin address (generated by the public key) as the identity in the transaction, and whoever has the private key indicates who has the Whoever has the private key has the right to dispose of the bitcoins at that address (i.e., can transfer bitcoins from that bitcoin address to another bitcoin address). The Bitcoin system is essentially a transaction-driven state machine, where the state of the entire Bitcoin system is constantly changing through the blockchain's constant determination of transactions. Typically, for account security and anonymity, each bitcoin user has several different bitcoin addresses, which are stored and managed by their wallet.

A transaction between users is divided into an input and an output. In the input section, the source of the bitcoin, the unlock script ScriptSig (usually a digital signature and public/public key hash proving ownership of the bitcoin), and the sequence number need to be included, and there is usually multiple outputs. 's public/public key hash), and by running the unlock script and lock script separately, any node in the Bitcoin network can verify the legitimacy of the transaction. In short, a transaction is informing the entire network that this part of the bitcoin holder is authorized to transfer it to a new holder, who can spend the bitcoin held by creating a new transaction, which in turn authorizes the bitcoin again to another holder, and so on, passing it on in a chain of owners.

For all nodes in the Bitcoin system to reach consensus, the Bitcoin system relies on a hash-based PoW (Proof-of-work) scheme. More specifically, to generate a block, a Bitcoin node must find a random number nonce value that makes the hash of the block header smaller than a given target value. The block header contains the version number, the previous block hash, a timestamp, the Merkle tree root hash (used to verify transactions within the block), the random number nonce, and the target hash.

The main principle of tamper-proofing the Bitcoin system: if a node/user wants to double the same bitcoin, they must replace the block that contains the transaction the bitcoin is in, otherwise the transaction they want to double spend will be detected immediately. This means that for a malicious node to reuse bitcoins undetected, they would need to recalculate not only all the work required for the block in which the bitcoin is located but also the PoW of all subsequent blocks in the chain. This mechanism ensures that the Bitcoin network can counter such misconduct as long as the percentage of honest nodes in the network exceeds the percentage of malicious collusion[2].

2.2. Double-spend attack

A "double-spend attack," as the name implies, is when the same bitcoin is spent twice or more in exchange for an item or service of excess value in the Bitcoin system. Unlike physically circulating cash, digital currency data is reproducible, which allows a digital asset to be reused. Simply put,

"double spend" is double payment or even multiple payments. The important thing to note about the concept of "double spend" is the action of "spend", not whether the double payment is successful or not, as long as the same bitcoin is "spent" twice or more. As long as the same bitcoin is "spent" two or more times, this behavior is called a "double spend attack", and the success or failure of the attack does not affect the definition of this behavior (which is also the basis for the penalties in the next section). There are two main types of "Double Flower Attacks": 51% Attacks and Finney Attacks.

The 51% attack is a "double spend attack" that is launched after the transaction has been confirmed by the blockchain. 51% attack refers to the attacker who has 51% of the arithmetic power of the blockchain system and then uses the powerful arithmetic power to cause the blockchain to fork and overwrite the blocks of the original transaction to gain benefits. 51% attack takes advantage of the "fork mechanism" of the blockchain: when different miners mine two new blocks at the same time, the main chain (the longest legitimate chain) will generate a fork. The blockchain's "fork mechanism": when different miners mine two new blocks at the same time, the main chain (the longest legal chain) will fork, and then the blockchain system will ask subsequent miners to always choose the blockchain with the largest cumulative proof of workload as the main chain to mine. Due to this convention, whenever a fork occurs in the main chain, miners will link their blocks to the chain with the largest cumulative workload currently, forming a new main chain, and transactions on the short chain at the fork will be discarded from recognition. The specific process for an attacker to launch a 51% attack is as follows.

1. First, the attacker should have more than 50% of the computing power of the entire network.

2. Using the "fork mechanism", first find the confirmed block x of the exchange that needs to be covered, and start regenerating the block after the x-1st block (containing a transaction to re-spend the spent bitcoins, possibly to an account under its control, or to another seller's account).

3. since the attacker holds more than 50% of the computing power of the whole network, the blockchain generated by the attacker's re-mining will become the main chain.

4. When the blockchain regenerated by the attacker becomes the main chain, at this time not only the attacker will mine on this chain, but also other honest nodes across the network will mine along the chain.

At this point, the attacker has successfully overwritten the original transaction to achieve double payment through powerful arithmetic.

A Finney attack targets a scenario where a transaction has not yet been confirmed, usually a "double spend attack" on a zero-confirmation transaction. Let's understand the process of a Finney attack through a case study.

The malicious payer intends to launch a Finney attack, the attacker uses bitcoin to purchase an item at a convenience store that supports zero-confirmation payments, assuming the item is priced at 0.1 BTC, the attacker will first issue a transaction TX_1, TX_1 shows the attacker transferring 0.1 BTC into the store's bitcoin address, the transaction is broadcasted in the bitcoin network, after a few seconds the store's node detects the legal transaction TX_1, and since the store is supporting zero-confirmation transactions, the store will hand over the goods to the attacker at this point. At the same time, the attacker will issue another transaction TX_2, and TX_2 will transfer the same bitcoin just paid to an address under his control. Since TX_1 and TX_2 are contradictory, the node that first receives TX_1 will decide that TX_2 is an illegal transaction, and The node that receives TX_2 first will decide that TX_1 is an illegal transaction and stop the continued broadcast of TX_2. Similarly, the node that receives TX_2 first will decide that TX_1 is an illegal transaction and stop the continued broadcast of TX_1. Eventually, only one transaction will be packaged on the blockchain, and if TX_2 is packaged on the blockchain, TX_1 will naturally not take effect and the attacker will have successfully implemented a Finney attack.

To increase the probability of successful attacks, attackers usually collude with multiple physically widely distributed malicious nodes to publish TX_2 transactions at the same time, so that the probability of successful attacks is extremely high[3], and also increase the chance of TX_2 being packaged on the blockchain in priority by increasing the transaction fee, and because a node receives TX_1 first and then does not broadcast the illegal transaction TX_2, so that the store node cannot detect it in time The "double flower attack" is not detected by the store node in time to stop the loss.

3. Fast payment protocol design

3.1. Protocol idea and content

The main idea of preventing double-spending attacks in fast payment is achieved through deposit penalty, i.e., the payer needs to create a time-locked deposit account, and if the payer has double-payment behavior, the deposit can be deducted to achieve the purpose of punishing the malicious payer. From the attacker's point of view, the purpose of the malicious payer is to gain benefit from the "double spend attack", and in order to achieve this goal, the attacker will make a trade-off between launching the attack and not launching the attack. The node will not risk losing the deposit to carry out a "double flower attack". The above main ideas are refined to form a specific agreement.

Assume that payer A seeks goods or services from merchant B via fast payment. Before initiating a transfer transaction, payer A needs to create a timed deposit account (with private key skA and corresponding public key pkA), set the lock time T, deposit amount D, and verify the beneficiary P (there are two beneficiary modes, if you choose to specify the beneficiary mode, you need to predefine P, which will be described in detail later), during the deposit lock time, payer A cannot transfer the deposit even if it has the private key Only the node that has both private key skA and private key skP has the power to dispose of the deposit account during the lock time T. That is, as long as the verified beneficiary P gets skA, he can withdraw the deposit during the lock time.

Merchant B receives the transaction and the transfer voucher and verifies their legitimacy respectively. If the verification is passed, Merchant B can provide goods or services to Payer A without waiting for the transaction to be packaged on the blockchain (still a zero-confirmation payment scenario), and at the same time send the transfer voucher of this transaction to the beneficiary P.

The main idea of the penalty mechanism of the protocol is that if payer A intends to launch a "double spend attack", then P will definitely receive the same bitcoin transfer credentials, using these two conflicting transfer credentials to obtain the deposit account corresponding to the private key skA, if payer A does not act maliciously, then the private key skA will be kept secret If payer A does not act maliciously, then the private key skA will remain confidential, ensuring that payer A can get back the deposit after the deposit lock time expires.

Assuming that A transfers money to B in exchange for expedited services, the main agreement is programmed as follows.

1. A needs to create a deposit of D with a lock time of T to verify the deposit of beneficiary P before the transaction.
2. A initiates a transfer transaction to B and at the same time needs to send a supporting voucher to B.
3. B needs to verify that this supporting voucher is indeed the voucher for the transfer to B.
4. B provides the service to A after the verification is passed, and at the same time sends the proof credentials to P.
5. If A carries out a double-spending attack with malicious intent, P must receive proof of credentials from another user B' (who receives the same funds as B).
6. P can use these two contradictory credentials to withdraw the deposit corresponding to the private key skA and transfer the deposit created before the A transaction to punish A.

3.2. Implementation theory support

3.2.1.Time-Locked Functionality in Bitcoin

A transaction-level time-locking feature existed at the beginning of the Bitcoin system, implemented through the nLocktime field in the transaction, which defines the earliest valid time for the transaction. Transactions with a specified future block or time nLocktime must be owned by the builder and are sent to the Bitcoin network only after they are valid. If a transaction is transmitted to the network before the specified nLocktime, the node receiving the transaction considers it invalid and stops broadcasting it. However, this time-locking feature has some limitations: it only restricts the transaction from being recognized by the entire network for a specified period of time, but it does not

restrict the user from constructing another transaction that costs the same amount of bitcoins during the lock time. This time-locking feature clearly does not satisfy the need to build deposit accounts in the protocol, and to overcome this limitation time locking must be placed at the script level, shifting from restricting transactions to restricting UTXOs.

In December 2015, the Bitcoin system introduced a new time locking feature through a soft fork upgrade that added a new script operator to the scripting language called CHECKLOCKTIMEVERIFY (CLTV), CTLV is a time lock on the output in each transaction, i.e. the output is restricted by adding CLTV to the output's locking script, thus ensuring that the output is not used until the The CTLV is a time lock on the output of each transaction. This script level time locking feature is perfectly suited for the deposit account requirements in this protocol. The use of CLTV is described in detail below[4].

CLTV opcodes take block height or Unix epoch time format as parameters. As with other validation-type opcodes, it returns true/false and continues execution of subsequent scripts if validation succeeds, or stops execution if validation fails. Suppose Alice pays Bob for a 1-month time-locked transaction. A transaction without a time lock would typically contain the following P2PKH script.

DUP HASH160 <Bob's public key hash> EQUALVERIFY CHECKSIG

With the lock time added, the transaction will be a P2SH transaction with the following redemption script included.

< OneMonthTime> CHECKLOCKTIMEVERIFY DROP DUP HASH160

<Bob's public key hash> EQUALVERIFY CHECKSIG

The previous time parameter will still occupy the top of the stack after CLTV verification, so it needs to be deleted by DROP command in order to execute the subsequent scripts correctly, so CHECKLOCKTIMEVERIFY is usually used with DROP in the scripts. CLTV does not replace nLocktime, but is used in conjunction with nLocktime to achieve the purpose of time locking. When Bob tries to spend this UTXO, he will stop execution if nLocktime is less than the lock time when building the transaction, even if Bob has the signature and public key corresponding to the unlock script, because CHECKLOCKTIMEVERIFY verification fails.

3.2.2.Establishment of deposit account

There are two types of beneficiaries in this Agreement (users who are entitled to transfer the deposit in the event of a "double spend" by the payer): one is a designated beneficiary P, which needs to be set at the time the deposit is created, and the other is an unspecified beneficiary, which is not required and is instead contested by the miner. The latter approach does not require a designated beneficiary P at the time the deposit is created, and instead the miners compete for ownership of the deposit. These two methods correspond to the two ways of creating deposit accounts, the creation of a deposit without a designated beneficiary and the creation of a deposit with a designated beneficiary.

1）Creation of a deposit without a designated beneficiary

In this case, in order to create a deposit account with a lock time of T and a deposit amount of D, payer A needs to create a transaction with the following script in the output section.

$$(T + T_{conf}^{impl})\ \texttt{CHECKLOCKTIMEVERIFY DROP}$$
$$pk_A\ \texttt{CHECKSIG}$$

The script T_{conf}^{impl} is the time margin, which will be discussed in detail in the security analysis section of this paper. The script uses CHECKLOCKTIMEVERIFY to ensure that the deposit is in a locked state for time T. If the CLTV verification passes, at this point, the top element of the stack is the digital signature pressed onto the stack by the unlock script, and the script continues execution by pressing A's public key pkA onto the stack, and the CHECKSIG operator verifies that the signature matches the public key, returning a boolean result to after the verification is complete (TURE/FALSE). The Bitcoin system verifies that the transaction is legitimate by executing the input script for this transaction and the output script for the transaction from which the funds originated, one after the other, and if they run successfully, the transaction is legitimate. In summary, the script ensures that the deposit account can only be disposed of by the user holding the key skA (not necessarily the payer A) after time T.

If payer A makes a malicious payment, then the private key skA of the deposit account becomes known to everyone, and each user can use the private key to create a transfer of the deposit to a bitcoin address in their possession, and the nLocktime field of the created transaction must be greater than or equal to the lock time. In this scenario, the probability and incentive for miners to transfer the deposit is highest, and because of the arithmetic power available, each miner does his best to try to transfer the deposit to his own transaction including posting it in the block, and the first miner in the competition to gain bookkeeping rights will receive this lucrative deposit.

2）Deposit creation for designated beneficiaries

In this case, in order to create a deposit account with a lock time of T, a deposit amount of D, and a designated beneficiary of P, payer A needs to create a transaction with the following script in the output section.

```
IF
    pk_P CHECKSIGVERIFY
ELSE
    (T + T_conf^expl + T_net^expl) CHECKLOCKTIMEVERIFY DROP
ENDIF
pk_A CHECKSIG
```

The script $T_{conf}^{expl} + T_{net}^{expl}$ is the time margin, which will be discussed in detail in the security analysis section of this paper. The flow control in the Bitcoin script is used in order to implement the protocol where (i) beneficiary P can transfer the deposit via skA during the deposit lock time if payer A has a "double spend" behavior, and (ii) A can get the deposit back after it expires if payer A has no malicious behavior.

A common use of flow control in Bitcoin scripts is to construct an unlocking script that provides multiple execution paths, each with a different method of redeeming UTX. The statements IF...ELSE...ENDIF are used in this script to implement the provisions in the protocol. The disposer can choose the path to be executed by itself. the IF statement will fetch elements from the stack and if the fetched element is TRUE or 1, it will execute the statement after IF, and the opposite will execute the statement after CHECKSIGVERIFY is similar to CHECKSIG, but if the signature is invalid, it will immediately cause the whole script to fail.

In a nutshell, the script ensures that if payer A has "double spend" during the lock time, then beneficiary P, who has obtained the key skA, can transfer the deposit via path ① in the script; if payer A acts honestly, then after the lock time A can retrieve the deposit via path ② in the script.

4. Conclusion

With the growing development of the Bitcoin system, the need for fast payment scenarios is becoming more and more urgent. To expand the scope of Bitcoin payments, research on new fast payment protocols is necessary and urgent. In this paper, to solve the drawbacks of zero-confirmation transactions in the current fast payment, a fast payment protocol based on a chameleon authentication tree is proposed. In order to implement the main contents of the protocol, this paper analyzes several problems of deposit account creation and adopts an authentication scheme based on the Merkle tree structure of the chameleon hash function. The new fast payment protocol cleverly combines the deposit with the authentication scheme by associating the key in the scheme with the deposit key, making it possible to implement a fast payment protocol on a distributed system through a deposit penalty. This protocol extends the scope of Bitcoin payments and refines their use in fast payment scenarios.

In the next step, the underlying data structure of the payment protocol will be further optimized by improving the static chameleon authentication tree into a dynamic chameleon authentication tree. The dynamism of the structure is mainly reflected in the fact that there is no need to determine its size at initialization, and instead, the tree size of the Merkle tree is adaptively expanded during the working process to meet the needs of the rising number of transactions to be processed in the future.

5. References

[1] FAQ-Bitcoin [EB/OL].https://en.bitcoin.it/wiki/FAQ.

[2] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system[J]. Decentralized Business Review, 2008: 21260.
[3] Karame G O, Androulaki E, Capkun S. Double-spending fast payments in bitcoin[C]// Proceedings of the 2012 ACM conference on Computer and communications security. 2012: 906-917.
[4] Mastering Bitcoin , Andreas M. Antonopoulos (O'Reilly,2017) , 978-1-491-95438-6.
[5] Schröder D, Schröder H. Verifiable data streaming[C]//Proceedings of the 2012 ACM conference on Computer and communications security. 2012: 953-964.
[6] Ruffing T, Kate A, Schröder D. Liar, liar, coins on fire! Penalizing equivocation by loss of bitcoins[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. 2015: 219-230.
[7] Li Lijuan. Research on the design and application of chameleon hash function[D]. Henan University of Technology, 2014.
[8] Rosenfeld M. Analysis of hashrate-based double spending[J]. arXiv preprint arXiv:1402. 2009, 2014.
[9] Decker C, Wattenhofer R. Information propagation in the bitcoin network[C]//IEEE P2P 2013 Proceedings. IEEE, 2013: 1-10.
[10] Krawczyk, H., Rabin, T.: Chameleon signatures. in: Proceedings of NDSS 2000.(2000) 143-154.