

# Attack and protection schemes on fabric isomorphic crosschain systems

Zhuo Lv<sup>1</sup>, Di Wu<sup>2</sup>, Wen Yang<sup>1</sup> and Li Duan<sup>2</sup>

## Abstract

Crosschain solves the problem of value transfer and asset interaction between different blockchains with different consensus mechanisms, or different infrastructures. It not only realizes the mutual communication of multiple independent blockchains, but also ensures the data consistency. However, existing crosschain technologies like notary mechanism, hash locking, distributed private key control, and sidechain/relaychain have potential vulnerabilities, resulting in different kinds of attacks. For example, some vulnerabilities that only exist in a blockchain like DDOS, overflow, double-spending, and so on, may have an impact on another secure blockchain due to the openness of the crosschain system. Even if all blockchains are safe enough, the security of the crosschain system cannot be guaranteed, and there are still many loopholes in the process of crosschain, which affect all blockchains. The security of crosschain has emerged as an important issue. In this work, based on BitXHub, an open-source project of crosschain, we are motivated to study the security of Fabric isomorphic crosschain system. First, we analyze the vulnerabilities in different layers of Fabric isomorphic crosschain systems, and discuss the principles of different kinds of attacks. Based on the principles, we discover and define five new crosschain attacks, including crosschain integer overflow attack, crosschain transaction forgery attack, crosschain transaction replay attack, crosschain transaction sequence attack, and crosschain routing attack. Second, we implement all the five attacks, and indicate the applicable scenarios and the boundary conditions in which each attack may occur. Third, we propose five schemes to prevent the corresponding attacks and evaluate the effectiveness of the protection schemes. We comprehensively discuss and analyze the schemes of attacks and their corresponding prevention toward Fabric isomorphic crosschain systems.

## Keywords

Blockchain, Fabric, isomorphic crosschain, attack, protection schemes, relaychain

Date received: 31 May 2021; accepted: 16 October 2021

Handling Editor: Yanjiao Chen

## Introduction

Blockchain technology originated from the white paper<sup>1</sup> published on 31 October 2008 by Satoshi Nakamoto. Due to the characteristics of decentralization, information transparency, impossibility of tampering, and security, blockchain has been widely applied in finance, logistics, e-commerce, medical treatment, justice, and other fields. According to different application scenarios, the blockchain can be classified into three categories: public blockchain, private blockchain, and consortium blockchain. Each node on the public blockchain can join or exit the blockchain

networks and participate in the data reading and writing of the blockchain. Besides, all nodes can read and write data in a flat topology, and there is no centralized service node in the network. The representative projects

<sup>1</sup>State Grid Henan Electric Power Research Institute, Zhengzhou, China

<sup>2</sup>Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing, China

## Corresponding author:

Li Duan, Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Shangyuancun 3, Haidian District, Beijing 100044, China.  
Email: duanli@bjtu.edu.cn

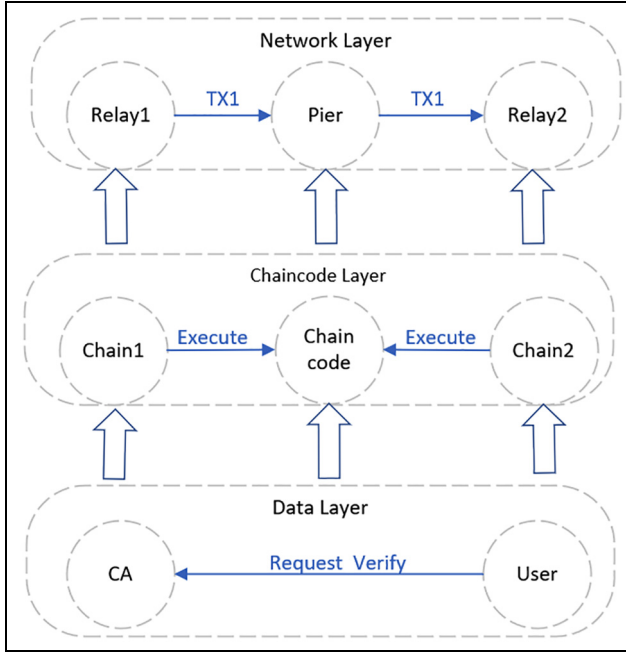


of public blockchain include Bitcoin,<sup>1</sup> Ethereum<sup>2</sup> with POW consensus, and EOS<sup>3</sup> with DPoS consensus. The write permission of each node in the private blockchain is controlled internally, while the read permission can be selectively opened to the outside world as required. Private blockchain is applicable to the internal data management and audit of specific institutions, such as Quorum<sup>4</sup> of JP Morgan Chase. Each node on the consortium blockchain usually has its own corresponding entity organization, and can only join or exit the blockchain network after being authorized. Various institutions and organizations form a consortium of stakeholders to jointly maintain the healthy operation of the consortium blockchain, such as Corda<sup>5</sup> and Hyperledger Fabric.<sup>6</sup> The essential difference among these three types of blockchains lies in the degree of decentralization. In addition to the widespread coexistence of public blockchains, private blockchains and consortium blockchains support different organizations to have their own blockchains, and even allow multiple blockchains to run simultaneously within the same organization. The number of global blockchain projects is increasing constantly, but the isolation of different blockchain networks hinders the efficient transfer of digital assets and accuracy of crosschain communication. More specifically, the continuous emergence of blockchain projects leads to an increase in asset types. If they cannot be exchanged with each other, a large number of different types of assets will appear in different business scenarios of different blockchains, which cannot be converted directly between these assets, and transaction information cannot be shared. Therefore, a technology is needed to solve the problem of information interaction and value transfer between different blockchain ecosystems, namely, crosschain. From the bookkeeping point of view, crosschain solves the problem of how to accurately book accounts in two distributed ledgers when data transfer happens in accounts controlled by the same or different users.

There are four ways to realize crosschain: notary mechanism, hash locking, distributed private key control, and sidechain/relaychain. The notary mechanism ensures crosschain fairness by selecting trusted intermediaries, but there may be malicious behavior among the elected notaries. Even with multiple signatures, the risk of centralization cannot be completely eliminated. Hash locking separates the control and ownership of crosschain assets by setting a private key. However, its protocol compatibility is low, and the refund time caused by the time locking mechanism is too long. Distributed private key control separates the control and ownership of crosschain assets by setting up a private key. Nevertheless, the crosschain system using distributed private key control cannot change the characteristics of the single blockchain that participates in crosschain. It needs to be adapted according to the characteristics of

the original blockchain, which makes its development difficult and time-consuming. Sidechain/relaychain uses another complete blockchain with a similar structure to aid crosschain transactions, in which sidechain is prone to suffer crosschain replay attacks because its structure is similar to the main chain. The security and scalability of the relaychain are high in several ways of crosschain, but its security mainly depends on the relay. Therefore, attention should be paid to the spread of crosschain attacks. At present, the mature crosschain open-source projects mainly include Polkadot,<sup>7</sup> Cosmos,<sup>8</sup> WeCross,<sup>9</sup> and BitXHub.<sup>10</sup>

Although many crosschain projects have been proposed, the current crosschain system is still facing high security risks. In March 2018, Lightning Network was attacked by DDOS, which caused about 200 nodes to go offline, and the total number of nodes decreased from about 1050 to 870. In October 2019, Lightning Network user ZipoTm was identified as cheating by other nodes because he could not get the latest transaction backup due to network dropouts, resulting in the loss of 4 BTC, which was up to US\$32,000. In June 2020, as the commercial sidechain of Blockstream, Liquid Network exposed security vulnerabilities. Due to inconsistent hash times, important accounts in the network were affected by technical loopholes, resulting in hundreds of BTC being stolen. In March 2021, the crosschain stable currency True Seigniorage Dollar (TSD) on ETH and BSC issued a statement indicating that network attackers used TSD DAO to forge 11.8 billion TSD tokens in their accounts and sold them in Pancakeswap. If the security of crosschain transactions cannot be guaranteed, it will bring irreparable economic losses to both sides of crosschain transactions. In the scenario of the combination between IoT and blockchain, the communication between different devices can also be regarded as a crosschain behavior. Therefore, the interaction between different devices will also create security vulnerabilities. For example, once the relay chain is attacked or a single device fails, the attack may spread to all the devices participating in the communication, resulting in the paralysis of the whole IoT network. Currently, most crosschain projects are based on public blockchain. Consortium blockchain, as one of the popular blockchain technologies in recent years, is different from those public blockchain projects that any user can access as long as they have corresponding terminals without identity authentication. Consortium blockchain has a set of strict identity authentication for organizations, which has been widely used in the daily business of various companies. As a representative project of consortium blockchain, Fabric has been applied to the business of major companies as soon as it goes online, with its advantages of excellent performance, strong scalability, high trust, modular architecture, and strict member access. Therefore, it is



**Figure 1.** Hierarchy of isomorphic fabric crosschain.

necessary to focus on the security of crosschain between different Fabric blockchains.

The Fabric isomorphic crosschain system implemented in this article can be divided into three layers: network layer, data layer, and chaincode layer. Its hierarchical structure is shown in Figure 1.

Relay1 and Relay2 represent two different relay chains that are connected through gateway Pier1 to forward transaction TX1. Similar to routers in the Internet, the function of network layer is to forward some crosschain transactions from relaychain to destination blockchain or gateway. However, since the relaychain is deployed in a different network environment from the destination blockchain, it is easily hijacked by attackers. Fabric1 and Fabric2 are two different Fabric blockchains. A crosschain chaincode is deployed on Fabric1 and Fabric2 at the same time, so that they can be called by both parties at any time and be verified easily. The chaincode layer is responsible for executing crosschain smart contracts (chaincode) on the two Fabric blockchains and updating data to the world state. Since the organizations participating in the crosschain have the right to publish contracts, the chaincode layer is most vulnerable to attacks from internal malicious members. CA represents a third-party trusted organization, and User represents a user or an organization. After user requests CA certification, it is allowed to participate in crosschain business. The data layer is responsible for authenticating and managing the identity of organizations participating in crosschain transactions. The difficulty lies in how to identify and determine the legitimacy of an organization on another

Fabric blockchain with the same structure. Once the verification is ignored, malicious organizations will cause irreparable damage to the whole crosschain system. Our work mainly focuses on the chaincode and network layers. We discover and define five crosschain attacks by analyzing the security of these two layers, including crosschain integer overflow attack, crosschain transaction forgery attack, crosschain transaction replay attack, crosschain transaction sequence attack, and crosschain routing attack. The crosschain integer overflow attack is caused by transferring a bigger amount of coins than that it can process. The crosschain transaction forgery attack is caused by an attacker maliciously calling the transaction record function in the user chaincode. The crosschain transaction replay attack is caused by an attacker replaying the encrypted transaction records on the network. The crosschain transaction sequence attack is caused by the crosschain transaction that should be executed later, but is executed in advance due to network congestion. The crosschain routing attack is caused by some attackers occupying the corresponding port or even directly controlling the server where the relaychain is located. It is worth noting that crosschain integer overflow attack, crosschain transaction forgery attack, and crosschain transaction replay attacks occur on the chaincode layer, while crosschain transaction sequence attack and crosschain routing attack occur on the network layer.

At present, the popular crosschain projects have proposed their own solutions to the safety problems at all levels in different crosschain scenarios. Polkadot adopts the RelayChain/Parachain model, which not only designs a secure sharing mechanism to reduce the difficulty of information exchange, and makes all parallel chains at the same security level, so that every blockchain can trust each other, but also introduces the concept of fisher, also called Bounty Hunter on Polkadot network, in order to monitoring malicious behavior on parallel chains. Its disadvantage is that the verifier on the relaychain is responsible for verifying the state change on the parallel chain. For example, the verifier may always reject the block proposed by the verifier in a blockchain for some reason, and the block in this relaychain can never be added to the global state. In order to avoid this situation as much as possible, Polkadot shuffles the verifiers to randomly verify the parallel chain, thus reducing the probability that the same verifier always verifies the same parallel chain. But there are still some security risks. Cosmos uses the Hub and Zone model, each blockchain runs independently, and has its own verifier, independent consensus mechanism, and independent security mechanism. Once there exists a behavior in Cosmos, because of this special structure, only one blockchain will be affected, and not the whole Cosmos network. Because each blockchain of Cosmos has its own verifier, it is very

likely that there will be collusion between verifiers in the partition. That is to say, if two partitions need to communicate, partition A needs to trust Cosmos Hub (communication hub) and verifier in partition B. Theoretically, before deciding to send information to partition B, users in partition A need to investigate the verifier of partition B. WeCross puts forward four core technologies, namely, UBI universal blockchain interface, HIP heterogeneous blockchain interconnection protocol, TTM trusted transaction mechanism, and MIG multilateral cross domain governance, which realize efficient, safe, reliable, and convenient governance of crosschain interaction. However, it mainly provides crosschain services for its self-built isomorphic blockchain called BCOS. Although it also provides some mainstream blockchain programming interfaces, it cannot compete with Cosmos and Polkadot in terms of interactivity with all other blockchains. BitXHub defines three roles of RelayChain, Application Chain, and Crosschain Gateway. It can be built into a flexible architecture of building block based on scenario orientation and has three core functional features, namely, Universal crosschain Transport Protocol, Heterogeneous Transaction Verification Engine, and Multi-Level Routing, which ensure the security, flexibility, and reliability of crosschain transactions. Its structure is relatively simple, which cannot be compared with the first three from the perspective of TPS, but its implementation is much simpler.

In this article, based on the Fabric, combined with the BitXHub project, we focus on the security problem in the crosschain scenario of homogeneous Fabric. We make the following contributions:

- Based on the principles of different kinds of attacks, we discover and define five new crosschain attacks, including crosschain integer overflow attack, crosschain transaction forgery attack, crosschain transaction replay attack, crosschain transaction sequence attack, and crosschain routing attack.
- We implement all the five attacks, and indicate the applicable scenarios and the boundary conditions in which each attack may occur.
- We propose five schemes to prevent the corresponding attacks and evaluate the effectiveness of the protection schemes. We comprehensively discuss and analyze the schemes of attacks and their corresponding prevention toward Fabric isomorphic crosschain systems.

## Background

This section focuses on introducing the basic concept of Fabric and comparing the advantages and disadvantages of several mainstream crosschain methods.

### Hyperledger Fabric

With the advent of the Blockchain 1.0 and 2.0 era, various financial projects based on blockchain technology emerge in an endless stream. Many innovative companies are attracted by it. However, most of them need stronger performance and more secure trading environment in their business scenarios, which cannot be achieved by public blockchain systems that all network members can access without permission. In addition, in many financial fields, the identity authentication of the transaction participants and the privacy protection of the transaction content are also core demands. So Hyperledger Fabric was born, which is an enterprise-level distributed ledger technology platform with strict identity authentication and access control. Fabric provides pluggable consensus protocols and supports a variety of general programming languages such as Java, Go, Node.js, and so on. The following five peers are defined in the Fabric:

- Anchor peer: each organization of each channel in Fabric has an anchor peer, which is used to receive the gossip messages sent by other organization peers and obtain the whole network information.
- Leader peer: all leader peers are elected by peers. They are responsible for distributing transactions from orderer peers to other peers. There can be multiple leader peers in an organization.
- Orderer peer: the orderer peers sort the verified legal transactions globally, and then package them into blocks.
- Endorser peer: The endorser peers are responsible for endorsing the transaction, namely, executing the transaction sent by the client, verifying the output result, and returning the result at the same time.
- Committer peer: The committer peer accepts the generated transaction block and verifies the correctness of the transaction before it is written into the ledger.

The verification and execution of transactions in Fabric are separated, which greatly improves the efficiency. At the same time, it allows multiple channels to coexist. Each channel is equivalent to an independent ledger isolated from each other, which greatly expands its application scenarios.

### Mainstream crosschain technology

*Notary mechanism.* Under the condition that there is no trust between the two blockchains and the information about both parties is not equal, the simplest way to realize crosschain is to find a trusted intermediary or

notary.<sup>11</sup> The notary mechanism is to help one blockchain involved in the crosschain such as chain A to verify whether another chain B has event X by using the corresponding algorithm to designate one or more trusted nodes as notaries jointly by users. Representative crosschain projects using notary mechanism are Corda<sup>5</sup> launched by R3 and ILP<sup>12</sup> proposed by ripple laboratory. As one of the simplest ways to realize the crosschain, once the notary as the intermediary has malicious behavior, the whole crosschain system will be exposed to serious security risks.

**Hash locking.** Hash locking was first implemented by the Bitcoin Lightning Network.<sup>13</sup> Its purpose is to ensure the security of both parties in the crosschain system by locking assets and setting corresponding unlocking conditions and unlocking time. Hash locking guarantees atomicity between any crosschain transactions. But it only realizes the exchange of crosschain assets; in fact, assets are not transferred from one blockchain to another.

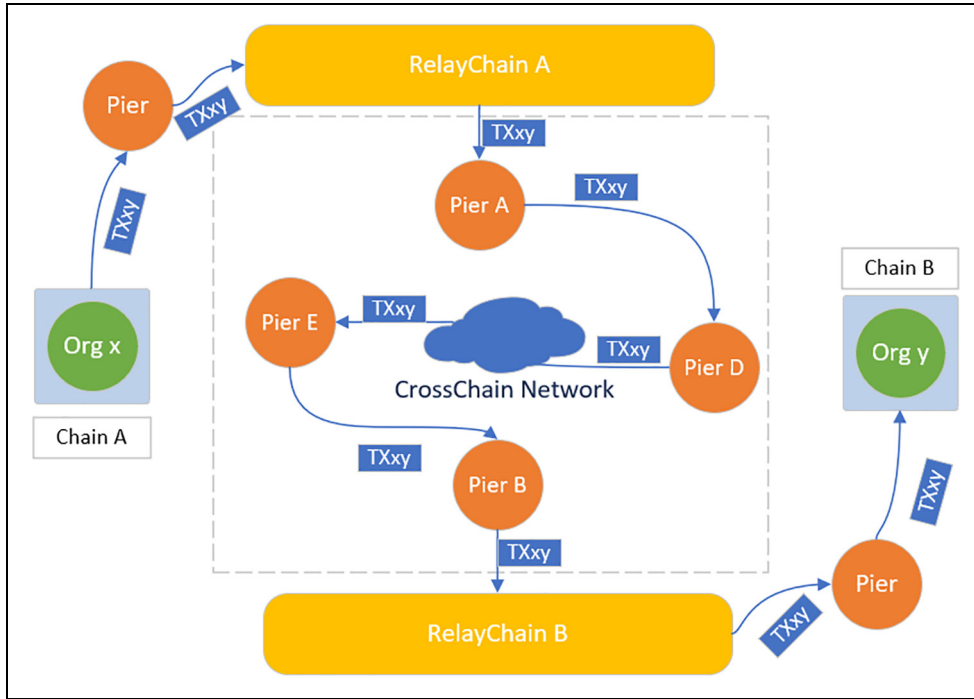
**Distributed private key control.** In the distributed private key control, the private keys of various assets in each blockchain are controlled through a plurality of distributed nodes. And they are mapped into a crosschain system at the same time, so that all assets can be interconnected in the whole crosschain network. In order to realize the complete separation of crosschain asset's ownership and corresponding right of use, it is important to manage and monitor the control of distributed private key. The most famous project of distributed private key control is Fusion.<sup>14</sup> The distributed private key control is similar to the notary mechanism. The difference is that the control of the blockchain's assets are owned by the users themselves, which can effectively avoid the risk of centralization. However, since the characteristics of each blockchain cannot be changed, the crosschain system that uses distributed private key control needs to be adjusted according to the characteristics of each blockchain, which makes development difficult and needs to wait for the confirmation of the blockchain for a long time.

**Side/relay chain.** Two-way anchoring technology<sup>15</sup> is adopted for the sidechain. When the performance of the mainchain encounters a bottleneck, relevant assets are transferred to the sidechain for execution to share the pressure of the mainchain. The main projects of sidechain include BTC Relay<sup>16</sup> and Lightning Network.<sup>13</sup> Compared with the sidechain technology, the relaychain technology abstracts the crosschain operation from all the blockchains, avoids excessive technical restrictions in the process of crosschain, and ensures a certain degree of security.

## Related work

In this section, we investigate the research status of crosschain attack and crosschain security at the domestic and foreign levels. And it is worth noting that there are still many gaps in the research of crosschain.

Peter et al.<sup>17</sup> discussed the possible attacks on the sidechain, proposed the crosschain replay attack, and pointed out that the attackers can replay transactions selectively to win coins and cash from sidechain. At the same time, he suggested that the transactions must have a private key signature corresponding to the account and a nonce value to prevent replay attack. But this replay attack only aims at the sidechain and cannot be extended to more scenarios of crosschain. In some public blockchain projects using POS consensus algorithm,<sup>18</sup> attackers can calculate a large number of blocks in advance by controlling malicious nodes on some public blockchains, and then spread them to the blockchain network at one time to realize long-distance attack. When the computing power is large enough, crosschain transactions based on the public blockchain will be reorganized, resulting in the cancelation of some previously completed crosschain transactions, leading to double-spending. Some crosschain systems using crosschain atom exchange technology are vulnerable to competitive condition attack.<sup>15</sup> In this attack, the attacker and the victim send coins of equivalent value to the designated address at the same time. Then the attacker sends an application to extract coins. In the case of poor network conditions, it is possible that the attacker not only received the coins of the victim, but also received the coins extracted by himself. It can also be called as another form of double-spending. However, long-distance attack and competitive condition attack will only happen in the crosschain systems in which the public blockchain participates. It cannot take effect in the crosschain scenarios with strict identity authentication. Some crosschain technologies get the response of the blocks participating in crosschain transactions by setting an appropriate delay time, so as to ensure the consistency of crosschain transaction data. To a certain extent, long-distance attacks are avoided. However, when the number of crosschain transactions reaches a certain level in a short time, each transaction needs a delay time to confirm, which will lead to a relatively large delay and cause the congestion of the whole crosschain network. It can be seen that at present, the implementation of crosschain attack mainly uses the unique attributes of some public blockchains, trying to spread the attack based on single blockchain to the whole crosschain system. Most of the researches stay at the theoretical level, without extending the analysis of attacks, and without analyzing the security of crosschain from the perspective of inter-chain interaction.



**Figure 2.** Architecture diagram of isomorphic Fabric crosschain system.

Formal analysis, as an important tool of blockchain security analysis, has become quite mature. J Jiao et al.<sup>19</sup> designed a formal specification for Ethereum smart contract to define semantic-level security attributes for advanced verification. Gleb Naumenko et al.<sup>20</sup> designed a new transaction propagation protocol for bitcoin, which improved the security of bitcoin network. Sergei Tikhomirov et al.<sup>21</sup> made a quantitative analysis on the security, anonymity, and scalability of lightning network. Ralf Kusters et al.<sup>22</sup> defined and analyzed the accountability of Fabric formally. The formal verification method can effectively analyze the security of different blockchain in detail. However, we can only regard them as the security analysis of the single blockchain rather than the whole crosschain system. Based on the above research, we can find that there are few researches on crosschain security, most of which focus on public chain attack, and study how to spread the attack based on single blockchain to crosschain. But with the development of blockchain technology and the continuous growth of consortium blockchain, crosschain is also indispensable. Therefore, it is necessary to focus on crosschain attacks and protection schemes in a variety of scenarios, especially in consortium blockchain.

### Fabric isomorphic crosschain

In this section, according to the characteristic of Fabric isomorphic crosschain, five kinds of crosschain attacks

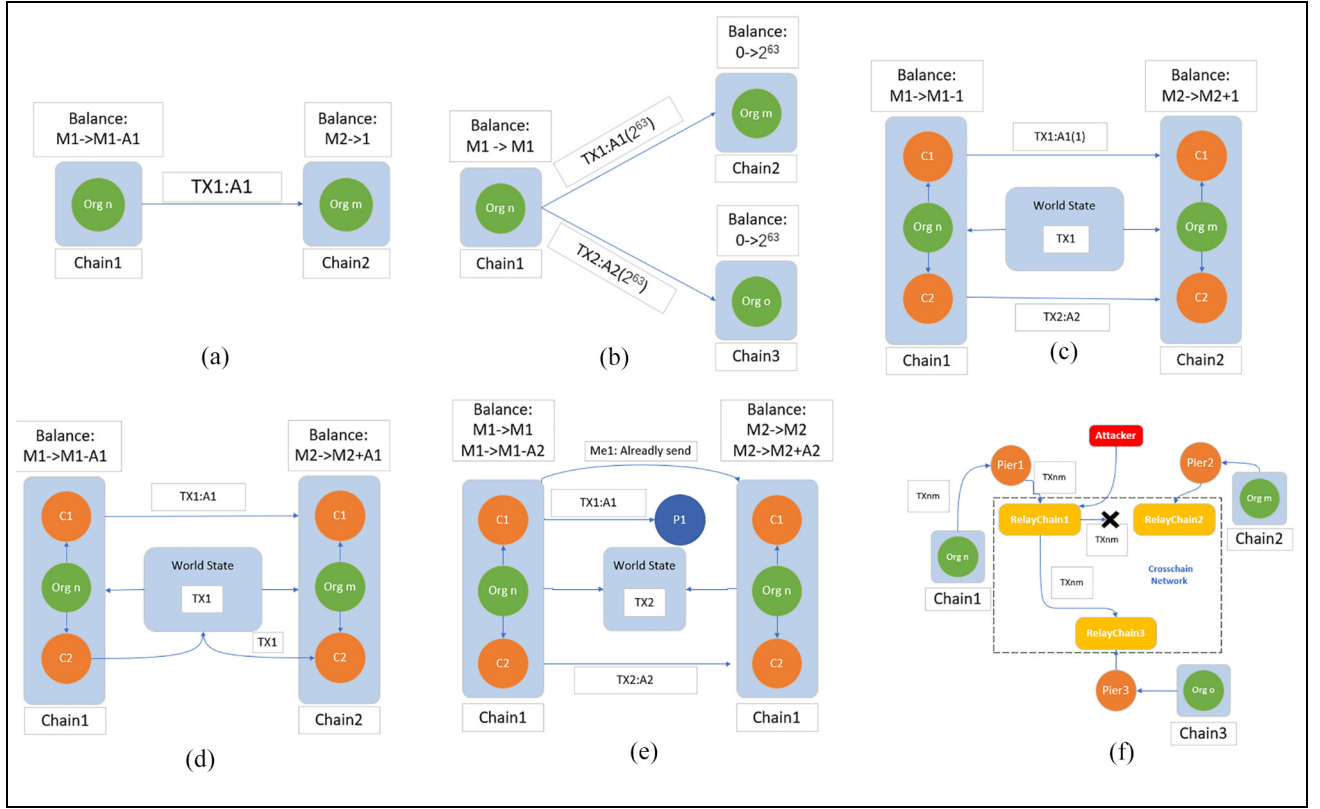
under different scenarios are proposed: crosschain integer overflow attack, crosschain transaction forgery attack, crosschain transaction replay attack, crosschain transaction sequence attack, and crosschain routing attack. The crosschain integer overflow attack can be divided into ordinary transfer integer overflow attack and batch transfer integer overflow attack. The definitions of these kinds of crosschain attacks are shown in Figure 3. Meanwhile, the boundary conditions of these attacks are analyzed and the corresponding application scenarios are proposed.

### Architecture of Fabric isomorphic crosschain

Based on BitXHub, the open-source project of Fabric isomorphic crosschain system, this article implements five crosschain attacks and corresponding protection schemes. The overall architecture of all attacks is shown in Figure 2. Its main components include the relaychain for application chain (the single blockchain which involves in crosschain) management, trusted verification and reliable routing of crosschain transactions, the crosschain gateway for collecting and disseminating transactions between blockchains, and the application chain responsible for specific business logic.

The main process of a crosschain transaction  $TX_{xy}$  sent from *OrgX* in application chain *A* to *OrgY* in application chain *B* is as follows: application chain *A* first initiates a crosschain transaction  $TX_{xy}$  to the corresponding gateway, and then the gateway sends it to





**Figure 3.** Definition of six crosschain attacks: (a) ordinary transfer integer overflow attack, (b) batch transfer integer overflow attack, (c) cross-chain transaction forgery attack, (d) cross-chain transaction replay attack, (e) cross-chain transaction sequence attack, and (f) cross-chain routing attack.

relaychain  $A$  accessed by application chain  $A$ . Relaychain  $A$  executes crosschain transaction to verify the correctness of relevant data. At this time, the transaction result is not included in the *WorldState*. If the verification fails, the transaction  $TX_{xy}$  is rolled back. If the verification is successful, query whether the address of destination chain  $B$  is in the application chain list managed by relaychain  $A$ . If it exists, it will be sent directly to the crosschain gateway where application chain  $B$  is located. If it does not exist, the distributed hash table will be used in the crosschain gateway cluster to query whether the crosschain gateway *PierB* of relaychain  $B$  associated with destination chain  $B$  exists. If it exists, the crosschain transaction is forwarded to the corresponding relaychain  $B$  according to the destination chain address and some routing forwarding protocols. At this time, the relaychain  $B$  verifies whether  $TX_{xy}$  has passed the transaction endorsement of relaychain  $A$ , and then forwards the transaction  $TX_{xy}$  to the destination chain  $B$ . At this time, the transaction  $TX_{xy}$  is executed, and the data are recorded in the *WorldState*.

### Crosschain integer overflow attack

**Attack principle.** Crosschain integer overflow is due to the existence of upper and lower bounds of integer

variables in the computer. If an integer variable exceeds its upper limit or falls below its lower limit, the variable can change from a large number to a small number or from a small number to a large number. When users call crosschain chaincode to transfer money, this phenomenon is easily exploited by attackers, thus achieving the goal of obtaining assets at a lower cost or even free of charge. This article is based on the Fabric isomorphic crosschain system, and its crosschain chaincode is written by *Golang*. Combining with the characteristic of *Golang* that the signed integer *int64* ranges from  $-2^{63}$  to  $2^{63} - 1$  and the unsigned integer *uint64* ranges from 0 to  $2^{64} - 1$ , it can be divided into two kinds of integer overflow attacks: ordinary transfer integer overflow and batch transfer integer overflow.

**Attack definition.** We define the ordinary transfer integer overflow in Figure 3(a). From Table 1, it can be seen that *Chain1* and *Chain2* represent two independent Fabric blockchains, respectively. *Orgn* is the organization of the transaction initiator, *Orgm* is the organization of the transaction receiver, and the balance of the sender on *Chain1* is  $M1$ , the balance of the receiver on *Chain2* is  $M2$ . When *Orgn* sending a crosschain transaction  $TX1$  with the amount of  $A1$  and the sum of  $A1$  and

**Table 1.** Crosschain symbol definition comparison table.

Symbol	Description
$M$	Initial balance of each Fabric blockchain. More specifically, the total initial balance of the organization where the transaction participants are located.
$TX$	A crosschain transaction on Fabric isomorphic crosschain system.
$TX_{nm}$	A crosschain transaction from $Org_n$ to $Org_m$ on Fabric isomorphic crosschain system. The $n$ represents the organization where the sender is located. The $m$ represents the organization where the receiver is located.
$A$	Amount involved in a crosschain transaction which reflects the changes of assets in the two Fabric blockchains with the transaction is in progress.
$Me$	A message in crosschain network that from one Fabric blockchain to another.
<i>Attacker</i>	Malicious attackers with legal identity lurking on two Fabric blockchains.
<i>RelayChain</i>	A relay chain corresponding to connect two or more Fabric blockchains with different piers.
<i>Chain</i>	A conventional Fabric blockchain participating in crosschain system.
$P$	Intermediate nodes for dealing with crosschain transactions, usually are some endorser peers.
<i>Pier</i>	Crosschain gateway corresponding to a Fabric blockchain and relay chain.
$C$	Crosschain chaincode deployed on both sides of blockchain. All legitimate Fabric blockchain can invoke the chaincode.
<i>Balance</i>	Change in the balance of the sender/receiver after crosschain transaction.
<i>WorldState</i>	The current status of all ledgers, include all blocks and transaction data, and the sender and receiver should be consistent.
<i>Org</i>	The organization of the sender (attacker)/receiver (victim) associated with a crosschain attack.

$M2$  is larger than or equal to  $2^{64}$ , crosschain integer overflow will occur. The consequence is that the sender sends a crosschain transaction with the amount of  $A1$ , while the balance of the receiver becomes very small. More precisely, the user-related data (specifically account balance here) stored in each transaction-related chaincode of Fabric has undergone a series of arithmetic operations after the relevant chaincode is called, resulting in the data exceeding the range represented by the variables. That is to say, the balance changes from a maximum to a minimum.

We define the batch transfer integer overflow as shown in Figure 3(b). When  $Org_n$  on *Chain1* sends crosschain transaction  $TX1$  and  $TX2$  with the amount of  $2^{63}$  to  $Org_m$  on *Chain2* and  $Org_o$  on *Chain3*, respectively, the three blockchains will generate the total transaction amount of  $2^{64}$  at the same time, causing integer overflow. At this time, it means that  $Org_n$  on *Chain1* does not pay any price, which increases the balance of  $Org_m$  on *Chain2* and  $Org_o$  on *Chain3* by  $2^{63}$ .

**Boundary condition.** For the integer overflow caused by ordinary crosschain transfer, the boundary condition is that the amount  $A1$  of the crosschain transaction  $TX1$  initiated by the sender  $Org_n$  plus the balance  $M2$  of the receiver  $Org_m$  is bigger than the upper limit of  $uint64$ , that is,  $A1 + M2 > 2^{64} - 1$ . At this time, both the sender and the receiver of crosschain transaction will lose most of their assets. For the integer overflow attack caused by batch crosschain transfer, the boundary condition is that the total amount in the batch crosschain transfer transactions  $TX1$  and  $TX2$  initiated by  $Org_m$  on *Chain1* to  $Org_m$  on *Chain2* and  $Org_o$  on *Chain3* is larger than or equal to  $2^{64}$ , that is,  $A1 + A2 > 2^{64} - 1$ . At this time,

the sender will increase the huge balance of the receiver at a small cost.

**Applicable scenarios.** Some blockchains participating in a crosschain and relaychains connecting each blockchain do not use the same programming language, which can lead to an integer overflow if either of them is improperly designed. If an integer overflow occurs on a single blockchain, it is likely to spread across the whole crosschain system. If an integer overflow occurs in a relay-chain, it will probably affect two single blockchains, thus destroying the whole crosschain system. For the vast majority of crosschain businesses, such as crosschain currency transactions, crosschain finance, or even arbitrary crosschain information exchange, as long as there exists an information change in digital form, integer overflow is easy to occur, because the addition and subtraction of numbers are the two basic arithmetic operations.

### Crosschain transaction forgery attack

**Attack principle.** Crosschain transaction forgery attack is that the attacker makes full use of the differences between the system chaincode and the user chaincode in the Fabric, combines the openness of the user chaincode to the members of the certification organization, calls the corresponding methods in the chaincode, and achieves the effect of forgery transaction in the chaincode layer, thus creating conflicts with the *WorldState*. In extreme cases, it can not only cause network congestion, but also achieve the effect that forged transactions are executed by peers.



Chaincode in Fabric are mainly divided into two categories: system chaincode and user chaincode. Users who want to query the historical data, mainly by calling the system chaincode to query the specified block. Although the historical data of all blocks can be queried, the results are difficult to identify for the general member users who are not related developers. Therefore, some Fabric blockchain developers are used to design a function to read transactions when developing, and display the corresponding crosschain transaction records to the users with query permission in the specified format. In this way, the designated users can read the transaction records in the predetermined format by calling the user chaincode without calling the system chaincode, and the steps are simpler. Taking the crosschain asset transaction chaincode as an example, we designed the transaction record function *writeTransaction()* according to the characteristic of crosschain asset transaction. As shown below, it can record all legal crosschain asset transaction records.

At the same time, the *getTransaction()* function is also designed to provide the query of corresponding crosschain asset transaction records (including sender, receiver, and sending amount) for users with sufficient permission:

```
func writeTransaction(stub shim.
    ChaincodeStubInterface, transaction
    CrossChainTX) (error) {
    txBytes, err := json.Marshal
    ($transaction)
    if err != nil {
        return errors.New ("`Marshalling
        Error`" +
            err. Error())
    }
    id := strconv. Itoa (transaction. Id)
    err = stub. PubState ("`transaction`" + id,
    txBytes
    )
    if err != nil {
        return errors. New ("`PutState Error`" +
        err.
        Error ())
    }
    return nil
}

Func get Transaction (stubshim.
    ChaincodeStubInterface) ([] CrossChainTX,
    error)
{
    var transactions[] CrossChainTX
    var number string
    var err error
    var transaction Cross Chain TX
    i:=0
    for i < transaction No{
        number=strconv. Itoa(i)
        transaction ,_, err=getTransactionById(
```

```
    stub, number)
    if err != nil {
        return nil, errors. New ("`Error get detail`" )
    }
    transactions=append (transactions,
    transaction)
    i=i + 1
    }
    return transaction, nil
}
```

**Attack definition.** According to the above properties, we define the crosschain transaction forgery attack in Figure 3(c). According to Table 1, C1 is the crosschain transfer chaincode with the crosschain transaction record function *writeTransaction()* and the crosschain transaction read function *getTransaction()*. C2 is the malicious crosschain chaincode deployed by the attacker in *Orgn*, which contains the function of writing forged crosschain transaction records. C1 and C2 are jointly deployed on *Chain1* and *Chain2*. *Orgn* on *Chain1* initiates a crosschain transaction *T X 1* with amount *A1* to *Orgm* on *Chain2*. And C1 will be called to write corresponding crosschain transaction records, including crosschain transaction initiator *Orgn*, crosschain transaction receiver *Orgm*, amount *A1*. At this time, transaction data related to *T X 1* will also be written into *WorldState*. However, the malicious attacker in *Orgn* can forge a crosschain transaction *T X 2* and write it into the record by calling crosschain chaincode C2, but actually *T X 2* does not affect the value of *Orgm* and *Orgn* in the *WorldState*, resulting in the inconsistency between the crosschain transaction data in actual ledger and chaincode C1. At this time, the attacker can claim that the crosschain transaction *T X 2* has been successfully initiated, and apply for the transaction receiver located in *Orgm* to return the involved amount *A2*. Because *T X 2* is forged by the attacker, the relevant data is not written into the *WorldState*, but its record exists in C1, which can be obtained by calling the *getTransaction()* function. The two are contradictory, which leads to the disorder of crosschain system and in some extreme cases. For example, the transaction receiver in *Orgm* who does not check the related data of transaction record in *WorldState*, but transfers the amount of *A2* directly to the receiver. In this way, as long as the attacker initiates a crosschain transaction *T X 1* with the amount of *A1* to the victim in *Orgn* at a very low cost and establishes a channel with *Orgn*, the attacker can cause the loss of the amount *A2* to *orgm* by forging transaction *T X 2*, and increase the balance *M1* by *A2*.

**Boundary condition.** The boundary condition of cross-chain transaction forgery attack is the moment that the attacker in *Orgn* just completed the crosschain transaction *T X 1* of transferring *A1* to the victim in *Orgm*,

established the crosschain channel between *Orgn* and *Orgm*, and successfully wrote the related data of *T X 1* into *WorldState*.

**Applicable scenarios.** The aforementioned crosschain transaction forgery attacks are illustrated based on the example of an isomorphic Fabric crosschain transfer transaction, but they do not only apply to a single scenario. As long as it involves the business that needs to record the crosschain transactions of Fabric, and the developer designs the function to read and write the crosschain transaction records in the user chaincode, this kind of attack may occur.

### Crosschain transaction replay attack

**Attack principle.** In the crosschain system using sidechain, the attacker can replay the crosschain transactions of the sidechain on the mainchain, so as to achieve the goal of double spending. In the Fabric isomorphic crosschain system using relaychain, the replay attack of crosschain transaction mainly involves the call of chaincode in the process of crosschain. The attacker can obtain an effective crosschain transaction by calling the user chaincode in the crosschain network, and then transfer the transaction back to the crosschain network. Because these two transactions are actually legal for the whole blockchain, and there is no identification to prove that a certain crosschain transaction is unique, so that these two crosschain transactions could be recognized by both blockchains at this time. In this way, without querying the system chaincode, the crosschain transaction will be executed twice.

**Attack definition.** We define the crosschain transaction replay attack in Figure 3(d). Similar to crosschain transaction forgery attack, *C1* is a crosschain transfer chaincode with crosschain transaction record function *writeTransaction()* and crosschain transaction read function *getTransaction()*. The difference is that *C1* writes and reads encrypted crosschain transaction records. *C2* is a malicious chaincode deployed by the attacker in *Orgn*, which contains a function of replaying encrypted crosschain transaction record. The attacker first initiates a crosschain transaction *T X 1* with the amount of *A1* from *Orgn* to the victim in *Orgm*, the value of *A1* is usually very large. And then waits for the transaction to complete. When the victim's account *M2* increases the amount of *A1*, and *T X 1* is successfully written into *WorldState*, the attacker reads the encrypted *T X 1* information by calling smart contract *C2* and replays it on the crosschain network again. Since *T X 1* is a successful transaction, the information must be correct. At this time, the attacker can claim that they have completed two transactions with the amount of *A1*. If the

victim does not query the system chaincode and directly checks the transaction records through *C1*, then he can indeed get two crosschain transactions with the amount of *A1*. The victim would think that the user on *Orgn* has successfully sent two transactions. As a result, the attacker has achieved the goal of double-spending.

**Boundary condition.** The boundary condition of the crosschain transaction replay attack is the *t* time after the attacker in *Orgn* successfully sends a crosschain transaction *T X 1* that is written in *WorldState* to the victim in *Orgm*, where *t* is bigger than or equal to 0. In other words, the instant after the completion of *T X 1*, this attack may occur. Of course, with the increase of *t*, the probability of successful attack also increasing.

**Applicable scenarios.** This crosschain transaction replay attack is a kind of crosschain forgery attack at the contract level. Therefore, the applicable scenarios are similar. For the business involving Fabric crosschain transaction records, and the developers design the function of reading and writing crosschain transaction records in user chaincode, this attack is likely to occur, such as the distribution of new digital asset, management of crosschain academic certification business, and custody of crosschain certificates. However, unlike the crosschain transaction forgery attack, transaction replay attack can still occur in the crosschain scenarios where private key is used as protection schemes in user chaincode. For example, in the business of crosschain asset transfer, users do not want to directly expose their own transaction records to other users, so they encrypt their own crosschain transaction records. At this time, crosschain transaction forgery attack cannot take effect, but crosschain transaction replay attack may still occur.

### Crosschain transaction sequence attack

**Attack principle.** In Fabric, after a crosschain transaction is executed, it is not directly written into the ledger, but temporarily put into the read-write set. Each peer maintains a read-write set according to the crosschain transaction and sends it back to the sender of transaction together with the endorsement results. When the sender of transaction collects enough endorsement results, it sends the crosschain transaction to the orderer peers to sort and generate blocks. Finally, it can be written into the ledger. The crosschain transaction sequence attack is caused by the attacker using the time difference between the crosschain transaction data entering the read-write set and the final *WorldState* when the crosschain transaction propagates in the network layer.

The function of *putstate()* in Fabric is to write the data into the ledger. However, at this time, the data are

still in the write set and are not really written into the whole ledger. It also needs to go through a series of stages such as crosschain transaction proposal. Some developers will output a prompt that the data have been written in the ledger and the crosschain transaction has been completed after the function is executed. When *putstate()* is called for several consecutive crosschain transactions in a short period of time, due to network blocking or even block cutting strategy, some crosschain transactions will have an arrival delay. Although the prompt of transaction executed successfully is output, the actual crosschain transaction data is not written in the ledger yet.

**Attack definition.** We define the crosschain transaction sequence attack in Figure 3(e). When *Orgn* on *C hain1* and *Orgm* on *C hain2* need to conduct continuous crosschain transaction, both parties stipulate that crosschain transaction *T X 1* can only be awakened after crosschain transaction *T X 2* is completed. First, the sender on *Orgn* calls the crosschain contract *C1* to initiate a crosschain transaction *T X 1* with the amount of *A1* to the receiver on *Orgm*. Due to the built-in method of *C1*, after the crosschain transaction *T X 1* is sent, a message *Me1* will be output to the crosschain network to indicate that the crosschain transaction *T X 1* has been completed. However, due to the network congestion, *T X 1* has just arrived at the intermediate peer, that is, the relevant data of crosschain transaction *T X 1* has not been written into the *WorldState*. At this time, the balance *M1* and *M2* of the sender and receiver have not changed, but the receiver receives the message *Me1* and thinks that the transaction *T X 1* has been completed. Therefore, the sender on *Orgn* is allowed to continue to send the transaction *T X 2* with the amount of *M2* initiated by the contract *C2* deployed on the both two chains, which is associated with *T X 1*. In the end, *T X 2* arrives at *C hain2* first, making the balance *M2* of the receiver increase *A2* first, resulting in the disorder of the whole crosschain system.

**Boundary condition.** According to the definition of attack, assuming that the time taken for the sender in *Orgn* on *C hain1* to write the relevant data of crosschain transaction *T X 1* to *WorldState* is *t1*, the time taken for the sender to receive message *Me1* is *t2*, and the time taken from the initiation of transaction *T X 2* to the writing of relevant data to *WorldState* is *t3*. The boundary condition for attack is that  $t3 \leq t1 - t2$ . The larger the difference between the two sides of inequality, the higher the probability of attack will occur.

**Applicable scenarios.** For the scenario that a large number of continuous crosschain transactions are generated

in the crosschain system, and for the scenario that is sensitive to transaction time and transaction order, such as crosschain auction and crosschain bidding, each blockchain participating in crosschain system starts to bid at the moment of commodity auction, there may be a situation that the starting time of bidding of two blockchains is the same. However, the arrival time of each blockchain's bidding message in the whole crosschain auction system must have a sequence, and it is impossible for two blockchain's bidding messages to reach consensus in the whole crosschain system at the same time, which violates the basic principles of auction. Similarly, for multiple transactions that belong to the same crosschain system in a certain period of time, it is also extremely prone to crosschain transaction sequence attack.

### Crosschain routing attack

**Attack principle.** Route hijacking attack refers to the behavior that hackers exploit vulnerabilities to attack user routers and steal user information, such as controlling router through malicious link or public IP address. After controlling the router, the attacker can implant advertisements and viruses to obtain user data by tampering with the DNS server of the router. Computers and mobile phones hijacked by routing will pop up advertisements constantly, and even lead to the theft of Internet banking, instant messaging tools, and other accounts. In crosschain system, the main function of crosschain routing is to forward crosschain messages and verify some crosschain transactions according to the address. Fabric and other consortium blockchains have a set of strict access control mechanisms that are more stringent in authentication. Different consortium blockchains are generally deployed in different companies, and it is difficult to invade through the IP address. However, for crosschain system adopting relaychain technology, relaychains are generally deployed on the public servers or third-party companies. In this way, attacks can hijack or block the whole crosschain network by attacking the corresponding port that is occupied by the relaychain.

**Attack definition.** We define crosschain routing attack in Figure 3(f). From *C hain1*, the sender in *Orgn* sends a crosschain transaction *TXnm* to a user in *Orgm*. The normal process is that *TXnm* passes through gateway *Pier1* and relaychain *RelayC hain1*, arrives at relaychain *RelayC hain2* through the crosschain network, and then it is forwarded to the corresponding user in *Orgm* through gateway *Pier2*. However, the attacker can attack the port of *RelayC hain1* on the server, resulting in the failure and paralysis of *RelayC hain1*, so that the crosschain transaction *T X nm* stops at the

gateway *Pier1* and cannot be forwarded by the *RelayChain1*. Furthermore, when the server is not well protected, the attacker can control the message forwarding of *RelayChain1* and send the transaction *TXnm* that should be sent to *Orgm* originally to *Orgo*, thereby disrupting the whole crosschain network.

**Boundary condition.** Since this inter-chain attack depends on whether the crosschain transaction *TXnm* reaches *Pier1* and whether the attacker has seized the port or controlled the *RelayChain1*, we must ensure that the crosschain transaction *TXnm* is located between the *Pier1* and the *RelayChain1*. In other words, if the time when the crosschain transaction *TXnm* arrives at *Pier1* is  $t_1$ , the time when it is forwarded to the crosschain network by *RelayChain1* is  $t_2$  and the time when the attacker occupies the port or controls *RelayChain1* at  $t_3$ , the boundary condition for the crosschain routing attack is  $t_1 \leq t_3 \leq t_2$ . The closer the value of  $t_2 - t_3$  is to  $t_3 - t_1$ , the higher the probability of success.

**Applicable scenarios.** There is more than one way to realize the interconnection of multiple blockchains.

However, mechanisms such as sidechain, hash locking, and notary do not have the structure of relaychain, so they do not have the risk of this kind of attack. For the crosschain scenario wherein more than two blockchains access in and each blockchain has several organizations, because the relaychain generally does not serve a single blockchain that is participating in the crosschain system, but is often deployed separately outside of other single blockchain environments, hence crosschain routing attack is much more likely to occur.

### Protection schemes to attacks

Combined with the analysis of the definition, boundary conditions, and applicable scenarios of various crosschain attacks in the previous section, this section puts forward corresponding protection schemes for various crosschain attacks.

**Crosschain integer overflow attack.** For the first kind of integer overflow caused by ordinary crosschain transfer, the protection scheme is to add the judgment of crosschain transfer and the check of account balance. Figure 4 shows the comparison of transaction execution

```
fabric-cli chaincode invoke --ccid=transfer --args '{"Func": "getBalance", "Args": ["ChainB"]}' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: 18446744073709551614
*****

fabric-cli chaincode invoke --ccid transfer \
--args '{"Func": "transfer", "Args": ["${TARGET_APPCHAIN_ID}", "mychannel&transfer", "ChainA", "ChainB", "1"]}' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: success!!!
*****

fabric-cli chaincode invoke --ccid=transfer --args '{"Func": "getBalance", "Args": ["ChainB"]}' \
--config "${CONFIG_YAML}" --payload \
--orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: 0
*****

fabric-cli chaincode invoke --ccid=transfer --args '{"Func": "getBalance", "Args": ["ChainB"]}' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: 18446744073709551614
*****

fabric-cli chaincode invoke --ccid transfer \
--args '{"Func": "transfer", "Args": ["${TARGET_APPCHAIN_ID}", "mychannel&transfer", "ChainA", "ChainB", "1"]}' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: Sorry, OverFlow!!!
*****
```

Figure 4. Protection schemes and renderings of ordinary crosschain transfer integer overflow attack.

**Algorithm 1** Resist Crosschain Integer Overflow Attack.**Input:** A cross-chain transaction  $TX$ ;**Output:** Is  $TX$  legal, and is it executed successfully;1: **repeat**2: Getting the detailed information of transaction  $TX$ , and locate the code segment  $code$  of smart contract  $C$  that involves arithmetic operation;3: Analyzing the operation type of  $code$ , whether it is addition, subtraction, multiplication or division;4: The security function is brought in according to the operation type for verification, and the verification result  $result$  is obtained;5: locating the next code segment  $code$ ;6: **until**  $code = \text{null}$  or  $result = 0$ 7: **return**  $En$ ;

```

fabric-cli chaincode invoke --ccid=transfer --args '{"Func":"getAllBalance","Args":["ChainA","ChainB","ChainC"]}' \
--config "$(CONFIG_YAML)" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | Payload: {"ChainA":"1000","ChainB":"0","ChainC":"0"}
*****

fabric-cli chaincode invoke --ccid=transfer \
--args '{"Func":"batchTransfer","Args":["$(TARGET_RECEIVER_ID)","mychannel&transfer","ChainB","ChainC","9223372036854775808"]}' \
--config "$(CONFIG_YAML)" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | Payload: success!!!
*****

fabric-cli chaincode invoke --ccid=transfer --args '{"Func":"getBalance","Args":["ChainB"]}' \
--config "$(CONFIG_YAML)" --payload \
--orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | Payload: {"ChainA":"1000","ChainB":"9223372036854775808","ChainC":"9223372036854775808"}
*****

fabric-cli chaincode invoke --ccid=transfer --args '{"Func":"getAllBalance","Args":["ChainA","ChainB","ChainC"]}' \
--config "$(CONFIG_YAML)" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | Payload: {"ChainA":"1000","ChainB":"0","ChainC":"0"}
*****

fabric-cli chaincode invoke --ccid=transfer \
--args '{"Func":"batchTransfer","Args":["$(TARGET_RECEIVER_ID)","mychannel&transfer","ChainB","ChainC","9223372036854775808"]}' \
--config "$(CONFIG_YAML)" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | Payload: Sorry, Overflow!!!
*****

```

**Figure 5.** Protection schemes and renderings of batch crosschain transfer integer overflow attack.

results before and after taking the protection scheme. For the second kind of integer overflow caused by crosschain batch transfer, it is necessary to design safety functions that are similar to the SafeMath library in Ethereum smart contract. The main process is shown in Algorithm 1.

These functions are used to check the possible overflow of addition, subtraction, and multiplication in crosschain chaincode involving arithmetic operation. When it comes to the arithmetic part of crosschain chaincode (such as addition, subtraction, and multiplication), as long as the function is used for checking, the integer overflow attack that may occur in most operations can be prevented. It looks similar to the SafeMath library in Ethereum, but in fact they are quite different. Because the bottom layer of Fabric is mainly written by

Golang, compared with solidity, its structure is more complex, and there will be more cases to be judged. At the same time, due to the characteristics of Fabric, this judgment only takes effect for users with specific permissions, and it is difficult for attackers to conduct targeted attacks again according to the judgment results. Figure 5 shows the comparison of transaction execution results before and after taking the protection scheme.

**Crosschain transaction forgery attack.** The way to prevent transaction forgery attacks is to add public and private key fields to each user when designing crosschain chaincode. The main process is shown in Algorithm 2.

When a user initiates a crosschain transaction, for example, after the crosschain transaction from the user

```

fabric-cli chaincode invoke --ccid=transfer --args '{"Func":"makeupTransaction","Args":["ChainA","ChainB","100000"]}' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | Payload: success!!!
*****

fabric-cli chaincode invoke --ccid=transfer --args '{"Func":"getTransactionById","Args":["3"]}' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | Payload: {"Id":"3","Sender":"ChainA","Receiver":"ChainB","Amount":"100000"}
*****

fabric-cli chaincode invoke --ccid=transfer --args '{"Func":"getTransaction","Args":[""]}' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | Payload: {"Id":"3","Sender":"ChainA","Receiver":"ChainB","Amount":"100000"}
*****

fabric-cli chaincode invoke --ccid=transfer --args '{"Func":"makeupTransaction","Args":["ChainA","ChainB","100000"]}' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | Payload: Error!!! Please sign your transaction correctly!!!
*****

```

**Figure 6.** Protection schemes and renderings of crosschain transaction forgery attack.

#### Algorithm 2 Resist Crosschain Transaction Forgery Attack.

**Input:** A cross-chain transaction  $T_X$ ;  
**Output:** Is  $T_X$  forgery;  
1: Transaction initiator  $A$  sign the transaction  $T_X$  with the private key  $P_{KA}$ ;  
2: The intermediate authentication node uses  $A$ 's public key for authentication;  
3:  $Result = Verify(P_{KA}(T_X))$ ;  
4: **if**  $Result = 0$  **then**  
5: Regarding  $T_X$  as an invalid transaction and dropping it;  
6: **Return False**;  
7: **else**  
8: Writing related data of  $T_X$  into the world state.  
9: **Return True**;  
10: **end if**

on *ChainA* to the user on *ChainB* is completed, the private key of the transaction initiator is used to sign the complete crosschain transaction record. When a crosschain transaction is being recorded, the corresponding public key is used to verify the transaction record. Only the transaction that passes the verification can be written into the ledger, the transaction records that fail to be verified are discarded as invalid transaction records, thereby preventing attackers from forging legal transaction records by introducing public and private key pairs. Different from the traditional scenario in which public and private key pairs are used to prevent attacks, the corresponding organizations of different Fabric need to create a connection to the relay chain every time a crosschain transaction is conducted. Therefore, if the fixed pair of public and private key is used, once lost,

the property will also be lost. In this article, the relay chain is designed to generate the temporary public and private key pairs, and the verification nodes distribute the keys to ensure the security of crosschain transactions. Figure 6 shows the comparison of attack results before and after using the public/private key pair.

**Crosschain transaction replay attack.** For crosschain replay attack, there are two kinds of protection schemes: adding timestamp and adding *nonce* field. Adding a timestamp is to record the creation time of a crosschain transaction for each legal crosschain transaction. In this way, if an attacker replays a transaction, two transactions initiated at the same time will appear in the transaction record. It can be considered that the subsequent crosschain transactions are likely to be replayed maliciously by the attacker. However, with the development of crosschain technology, TPS is getting higher and higher; if the timestamp is only accurate to seconds, it is completely possible to launch two transactions in 1 s. At this time, it is impossible to check whether the two crosschain transactions with the same time are forged by malicious attackers or normal crosschain transactions occurred in a short time interval.

Therefore, in order to resist the crosschain replay attack more effectively, it is necessary to set the *nonce* field for each user on the basis of the timestamp, and then record the *nonce* value of each legal crosschain transaction initiator when writing the transaction. The main process is shown in Algorithm 3. When the transaction is being verified, it is also necessary to compare whether the *nonce* values recorded in the transaction



**Algorithm 3** Resist Crosschain Transaction Replay Attack.

**Input:** A cross-chain transaction  $TX$ ,  $creationtime$ ,  $receptiontime$ ,  $t1$ ,  $nowtime$ ,  $t2$ , the *nonce* value  $n1$  recorded in the transaction initiator, the *nonce* value  $n2$  recorded in the user chaincode;

**Output:** Is  $TX$  would be replayed;

```

1: Transaction initiator A sign the transaction  $TX$  with the
   private key  $PK_A$ ;
2: The intermediate authentication node uses A's public
   key for authentication;
3:  $Result = Verify(PK_A(TX))$ ;
4: if  $Result = 0$  then
5:   Regarding  $TX$  as an invalid transaction and dropping it;
6:   Return False;
7: else
8:   if  $|t1 - t2| < 1$  then
9:     if  $n1 = n2$  then
10:      Writing related data of  $TX$  into the world state.
11:      Return True;
12:   end if
13: end if
14: end if
15: Return False;

```

initiator and the user chaincode are consistent. If they are consistent, the crosschain transaction will be verified, then add one to the *nonce* value of the crosschain transaction initiator account or set another random *nonce*. In this way, when the attacker replays the transaction again, the *nonce* value inside the transaction and the *nonce* value of the current account will be inconsistent, resulting in the failure of the attack.

Compared with the schemes of preventing transaction replay attack by Ethereum smart contract, this way of combining *nonce* value with timestamp is simpler. Thanks to the authentication-based features of Fabric and its high TPS, this protection scheme is sufficient to prevent crosschain transaction replay attacks. Figure 7 shows the comparison of attack results before and after adding the timestamp and *nonce* value.

**Crosschain transaction sequence attack.** For the crosschain transaction sequence attack, the protection scheme is to call *getState()* function to query whether the crosschain transaction data is written into the ledger after the sender initiates a crosschain transaction. If it is written successfully, a message about crosschain data having been written successfully will be sent to the sender of the next transaction, and then the next crosschain transaction will be sent normally. Otherwise, it will wait until the confirmation message is received. Although this scheme is easy to achieve, its biggest problem is that it sacrifices the transaction transmission time and trades time for space. In extreme cases, it is easy to cause crosschain network blocking.

Another method is to make a crosschain agreement that a series of continuous crosschain transaction data has an end flag *state*. If the data are the final data of a

certain stage, the *state* is set to 1, otherwise, it is set to 0. After receiving the final crosschain transaction data, restore the flag to 0. Compared with the former, this method does not need to call *getState()* function to query the ledger data, although it still needs time to confirm the flag bit, and its time costs is much less than the first method. We can combine these two methods; the main process is shown in Algorithm 4.

The conditions for such attacks are rather harsh. Therefore, even if the protection means takes a certain amount of time, because of the high efficiency of the Fabric, the whole TPS will not be affected, and it will be more difficult to be attacked by DDOS. Figure 8 shows the comparison of attack results before and after using state variable.

**Crosschain routing attack.** The first scheme is to check the balance of blockchain. For some specific crosschain chaincodes, such as auction, transfer, and so on, a certain amount of balance is required on one or all participants. Otherwise, some malicious attackers will hijack the relaychain, forward the transaction that was supposed to arrive at the *ChainB* to *ChainC* that they newly created, transfer all the balance out, and then destroy *ChainC* to withdraw from the whole crosschain system. Therefore, before starting to send crosschain transactions, it is necessary to agree on a balance range limit of each blockchain. The blockchain whose balance is larger or less than this range will be considered as an attacker and added to the blacklist.

The second scheme is to check the creation time of each blockchain. Most blockchains have been created long before the execution of a crosschain transaction. If one blockchain is created and accessed only at the approaching time when the crosschain transaction is being executed, it is likely to be a malicious attacker. Therefore, through network congestion and execution time, we can specify a corresponding timestamp before the execution of a crosschain transaction. By checking the creation time of each blockchain and comparing with the specified timestamp, we can also filter some suspected malicious transactions. If the creation time of a blockchain is larger, it can be considered that the creation time is too short and is added to the blacklist.

```

from pybloom-live import BloomFilter
import sys
url = str(sys.argv(1))
bf = BloomFilter(capacity = 10000)
fd = open('whitelist.log')
while True:
    url = fd.readline().strip()
    bf.add(url)
    if url == 'exit':
        break
print(url in bf)

```

```

fabric-cli chaincode invoke --ccid=transfer --args '{"Func": "getTransactionbyId", "Args": ["1"]}' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: {"HltzeJ7iaUHE7u/ojN2yR5WMUmyhYBDdBNFgjOS4c7/IC6mK8uB+Mbyc9jw2ryniCCYHhTFPQgGkQavtOzsg
*****

fabric-cli chaincode invoke --ccid=transfer --args '{"Func": "makeupTransaction", "Args": [{"HltzeJ7iaUHE7u/ojN2yR5WMUmyhYBDdBNFgjOS4c7/IC6mK8uB+Mbyc9jw2ryniCCYHhTFPQgGkQavtOzsg"}]' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: success!!!
*****

fabric-cli chaincode invoke --ccid=transfer --args '{"Func": "getTransactionbyId", "Args": ["1"]}' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: {"HltzeJ7iaUHE7u/ojN2yR5WMUmyhYBDdBNFgjOS4c7/IC6mK8uB+Mbyc9jw2ryniCCYHhTFPQgGkQavtOzsg
*****

fabric-cli chaincode invoke --ccid=transfer --args '{"Func": "makeupTransaction", "Args": [{"HltzeJ7iaUHE7u/ojN2yR5WMUmyhYBDdBNFgjOS4c7/IC6mK8uB+Mbyc9jw2ryniCCYHhTFPQgGkQavtOzsg"}]' \
--config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: Sorry! Transaction illegal!!!
*****

```

**Figure 7.** Protection schemes and renderings of crosschain transaction replay attack.

#### Algorithm 4 Resist Crosschain Transaction Sequence Attack.

**Input:** A series of crosschain transaction flows  $TX[n]$ , the state value  $state$  recorded in the user chaincode;  
**Output:** Are the  $TX[n]$  executed correctly;  
1: Get the first transaction  $TX[i]$  of the flow, the initial value of  $i$  is 1;  
2: **while**  $i \leq TX.length$  **do**  
3: **if**  $state=0$  **then**  
4: Querying whether the crosschain transaction data is written into the ledger after the sender initiates a crosschain transaction;  
5: **if not then**  
6:  $i++$ ;  
7: **continue**;  
8: **else**  
9: **break**;  
10: **Return False**;  
11: **end if**  
12: **else**  
13: **break**;  
14: **Return False**;  
15: **end if**  
16: **end while**  
17: **Return True**;

Adding a blockchain whitelist can protect against crosschain routing attacks, but it can also have the most stringent prerequisite. It is necessary to determine which blockchains are trustworthy in advance, and then put them into a whitelist file. When a crosschain transaction is formed in the crosschain network, check whether the sender's ID exists in the whitelist. If it

exists, the transaction will be executed normally, if not, further identity checks are required. With more and more blockchains accessing into the crosschain system, the number of whitelists will increase, so the search efficiency will be seriously reduced when the capacity of lists is very large. Therefore, we introduced the Bloom filter to improve the efficiency of searching whitelist. For the blockchains that pass the filter, they will be added to the whitelist. However, due to the false positive rate of Bloom filter, for those application blockchains that fail to pass the Bloom filter, manual review is needed. These protective schemes are not difficult to achieve. However, if the three schemes are combined, compared with other schemes used to prevent routing attacks, it can ensure the security of the relaychain to the maximum extent on the basis of easy implementation. The main process is shown in Algorithm 5.

#### Safety and feasibility analysis

In this section, we analyzed the security and feasibility of the protection schemes for different crosschain attacks proposed in the previous section. At the same time, different security assumptions based on different protection schemes are proposed.

#### Crosschain integer overflow attack

For the first kind of integer overflow attack caused by ordinary transfer, it occurs because the developer

```

fabric-cli chaincode invoke --ccid transfer \
  --args '{"Func":"transfer","Args":[""${TARGET_APPCHAIN_ID}","mychannel&transfer","ChainA","ChainB","1"]}' \
  --config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: success!!!
*****

fabric-cli chaincode invoke --ccid transfer \
  --args '{"Func":"transfer","Args":[""${TARGET_APPCHAIN_ID}","mychannel&transfer","ChainB","ChainA","2"]}' \
  --config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: success!!!
*****

fabric-cli chaincode invoke --ccid=transfer --args '{"Func":"getTransactionById","Args":["0"]}' \
  --config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: {"id":"0","Sender":"ChainB","Receiver":"ChainA","Amount":"3"}
*****

fabric-cli chaincode invoke --ccid transfer \
  --args '{"Func":"transfer","Args":[""${TARGET_APPCHAIN_ID}","mychannel&transfer","ChainA","ChainB","1"]}' \
  --config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: success!!!
*****

fabric-cli chaincode invoke --ccid transfer \
  --args '{"Func":"transfer","Args":[""${TARGET_APPCHAIN_ID}","mychannel&transfer","ChainB","ChainA","2"]}' \
  --config "${CONFIG_YAML}" --payload --orgid=org2 --user=Admin --cid=mychannel
*****
**** | Response[0]:
**** | | Payload: Front transaction is not been excuted yet, please wait!!!
*****

```

**Figure 8.** Protection schemes and renderings of crosschain transaction sequence attack.

neglects to check the balance when developing the chaincode. In fact, this condition is not easy to meet. But for the integer overflow caused by batch transfer, even if the safety function is adopted, as long as there is a step with arithmetic operation and the safety function is not called, the whole crosschain system still faces the risk of integer overflow.

### Crosschain transaction replay attack and forgery attack

For crosschain transaction forgery attack, when a crosschain transaction from user  $A$  to user  $B$  is completed, a transaction record  $TX1$  is generated, including the initiator ( $A$ ), the relaychain entry node ( $Ec$ ), and the amount ( $m1$ ). At the same time,  $A$  packages  $TX1$ , transaction receiver ( $B$ ), transaction limit time ( $Time$ ), and the minimum number of verification nodes ( $Vnum$ ) on relaychain into a message  $Mess$ , and sends it to verification node  $V$  on relaychain for verification. Figure 9 shows the transaction flow after taking protective scheme.

When a crosschain transaction record is written, it is necessary to use the corresponding public key to verify

the signature of the transaction record. For the verified transaction to be written into the record, that is, equation (1) needs to be met. Records that fail to verify are discarded as invalid transaction records. In this way, by using public and private key pairs, attackers can be prevented from forging legal transaction records as

$$Verify(Sign2) = (A, B, m1, Sign1, Time, Vnum) \quad (1)$$

For crosschain transaction replay attack, adding a timestamp is to write the creation time  $t_{create}$  of each legal transaction record  $TX1$  sent from  $A$  to  $B$ . At this time, the transaction information  $mess1$  sent by  $A$  to  $Ec$  contains the signed transaction record  $TX1$  with  $t_{create}$  field, as shown in Figure 10. At the same time,  $t_{create}$  also needs to meet equation (2). The interval between two transactions is at least 1 s. In this way, if an attacker replays a transaction, two transactions created in the same second will appear in the transaction record. It can be considered that the subsequent crosschain transaction is a transaction replayed by the attacker maliciously as

$$t_{create}(n) - t_{create}(n-1) \geq 1 \quad (2)$$

**Algorithm 5** Resist Crosschain Routing Attack.

---

**Input:** A crosschain transaction  $TX$ , Creation time  $t[n]$  of each Fabric blockchain, the balance  $b[n]$  of each Fabric blockchain, time threshold  $T$ , balance threshold  $B$ , *whitelist*;  
**Output:** Which Fabric blockchains have access to the relaychain;  
1: Get the first Fabric's creation time  $t[i]$ , balance  $b[n]$ , the initial value of  $i$  is 1;  
2: **while**  $i \leq TX.length$  **do**  
3: **if**  $i$  in *whitelist* **then**  
4: Give permission for  $t[i]$  to connect relay chain;  
5:  $i++$ ;  
6: **else**  
7: **if**  $t[i] > T$  and  $b[i] > B$  **then**  
8: Put  $i$  into *whitelist*;  
9: **else**  
10: Artificial in-depth judgment;  
11: **end if**  
12:  $i++$ ;  
13: **end if**  
14: **end while**

---

They both occur under the condition that the user chaincode is used to record the data of crosschain transaction. If both parties involve in crosschain query transaction records by calling the system chaincode, then these two attacks cannot occur. Adding public/private key pair can effectively prevent crosschain transaction forgery attack, but it may cause crosschain replay attack. Although adding timestamp is the best way to achieve anti-replay, due to the improvement of TPS, multiple transactions can be completed in 1 s. Adding a timestamp may lead to misjudgment, so it needs to be combined with random number *nonce* for protection. Setting the  $User_{nonce}$  value by random for the transaction record  $TX1$  based on the creation time  $t_{create}$ , and then recording whether the  $User_{nonce}$  value of the initiator of each legal transaction is *nonce* when the transaction record is written. At the transaction verification stage, comparing whether the initiator of the transaction and the *nonce* value recorded in the transaction are consistent, as shown in equations (3) and (4). If consistent, the crosschain transaction is verified. At this time, the  $User_{nonce}$  value will be set to another random number. Even if the attacker replays the transaction again, the value of *nonce* in the transaction will be inconsistent with the value of  $User_{nonce}$  in the current account, resulting in the failure of the attack as

$$Verify(Sign2) = (A, B, m1, Sign1, Time, V_{num}, t_{create}, nonce) \quad (3)$$

$$nonce = User_{nonce}(User_{nonce} \in A) \quad (4)$$

**Crosschain transaction sequence attack**

For crosschain transaction sequence attack, if the transaction data is the final data of a certain stage, that is,

after the transaction is executed, the relevant data has been included in the *WorldState*. Then the *state* is set to 0, otherwise, it is set to 1, as shown in equation (5). After calling the final data, restore the *flag* to 0. This method does not need to call *getState()* method to query the ledger data. Although it still needs time to confirm the *flag's* value, the time cost is much less

$$Flag = \begin{cases} 1, & TX \in World\ State \\ 0, & TX \notin World\ State \vee TX_{n+1} = \emptyset \end{cases} \quad (5)$$

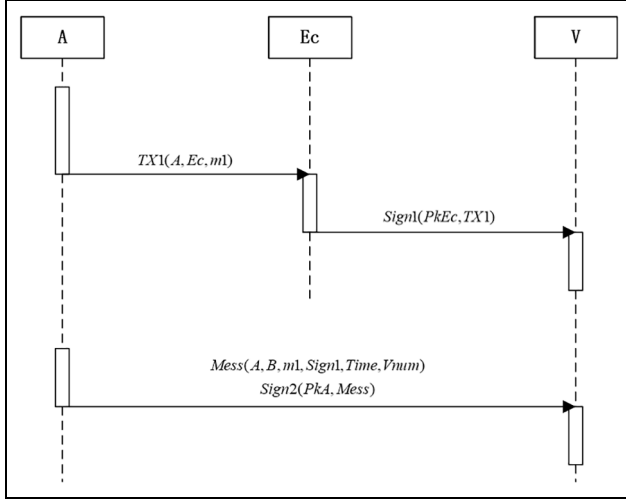
It is based on two security assumptions: continuous crosschain transaction data flow and crosschain network congestion. If only continuous crosschain transactions arrive, but the network is unobstructed, based on the performance of Fabric, it is difficult to block the previous transaction and execute the next transaction first. Similarly, if the network is blocked, but the transaction is not continuous, even if the transaction is blocked, it will not last too long. When the next transaction is sent, the previously blocked transaction has already been executed.

**Crosschain routing attack**

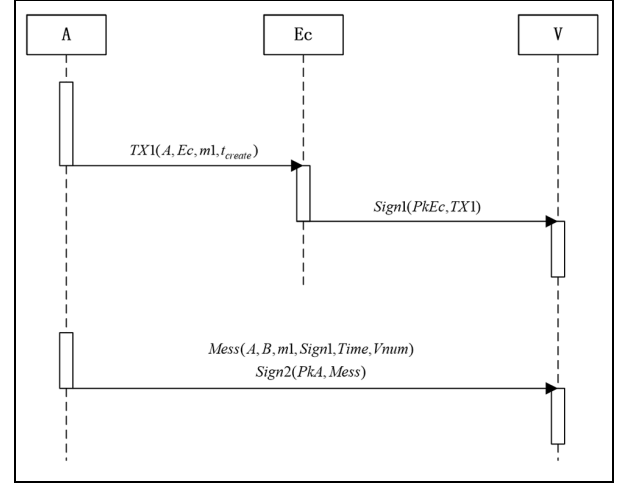
For crosschain routing attacks, the security assumption is the strictest. It requires the attacker to occupy the port of the server that responds for deploying the relaychain and interacting with the gateway. Meanwhile, the target crosschain transaction has just arrived at the gateway and has not been forwarded by the relaychain. If the assumption goes further, the attacker not only needs to occupy the port but also to control the relaychain that is responsible for forwarding. Although the assumption is strict, the loss to the crosschain system is also the most serious. Once hijacking occurs, the corresponding blockchain will never be able to access the crosschain network, the relaychain will lose the forwarding function, and the corresponding crosschain assets will be invalid.

**Experiment****Experimental setup**

We deployed the relaychain on Alibaba cloud server that having 4 processors with 128GB RAM and 1TB hard Disk, and each processor has 8 cores. This server is running with Ubuntu 18.04. We also deployed three different Fabric blockchains on different local servers in different network segments. All of them have the same network bandwidth and 2 processors with 64GB RAM and 1TB hard Disk, and each processor has 4 cores. All chaincodes were written in JAVA.



**Figure 9.** Transaction process after taking protective scheme (crosschain transaction forgery attack).



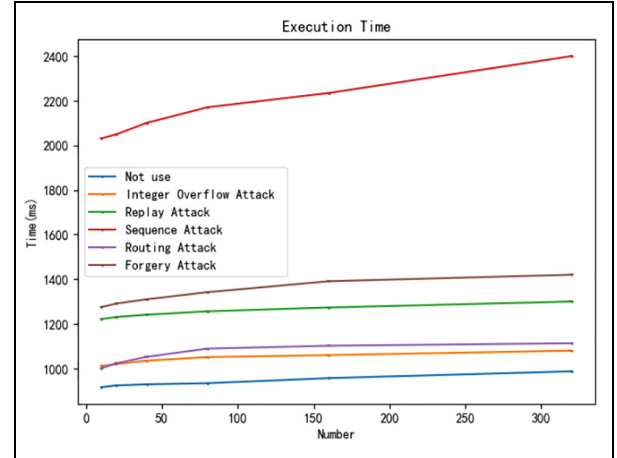
**Figure 10.** Transaction process after taking protective scheme (crosschain transaction replay attack).

### Experimental step

Based on the above configuration, we have implemented five protection schemes corresponding to crosschain integer overflow attack, crosschain transaction forgery attack, crosschain transaction replay attack, crosschain transaction sequence attack and crosschain routing attack. Their implementation details are shown in section “Safety and feasibility analysis.” In order to study the efficiency of these protection methods and their impact on trading time, we conducted 50, 100, 150, 200, 250, and 300 crosschain transactions under the condition of fixed smart contracts. The time cost of the crosschain system with five protection methods is compared with the original crosschain system. The experimental results are shown in Figure 11. Figure 12 shows the comparison after removing the system startup activation time. The size of crosschain transaction data is also the focus of our attention, so we set the transaction data sizes to 0.05, 0.1, 0.2, 0.4, 0.8, and 1.6 KB when the network bandwidth is consistent and the number of transactions is fixed at 50. Figure 5 shows the time cost of five different protection schemes and without any protection schemes.

### Result analysis

It can be found from Figure 11 that, except for crosschain sequence attack, the time overhead caused by taking other protection schemes does not fluctuate much when the data size remains unchanged. It is worth noting that even if the number of transactions is small, the time spent is more than 900 ms. This is because Figure 11 counts the startup time of each crosschain system. Therefore, if the startup time of crosschain system is removed, the time statistics will be



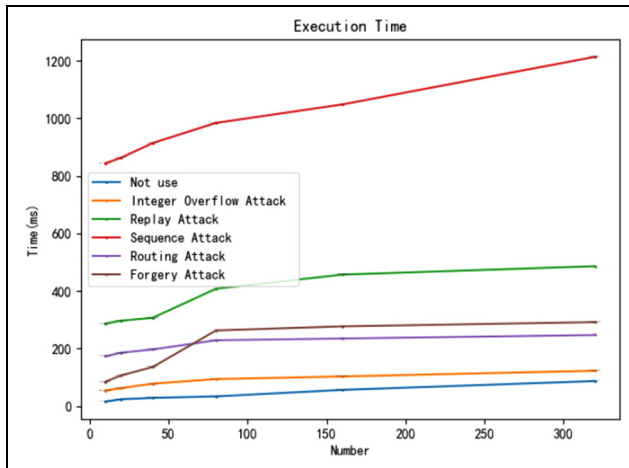
**Figure 11.** Time cost after taking different protective schemes.

more accurate. On the basis of Figure 11, we completed the statistics shown in Figure 12.

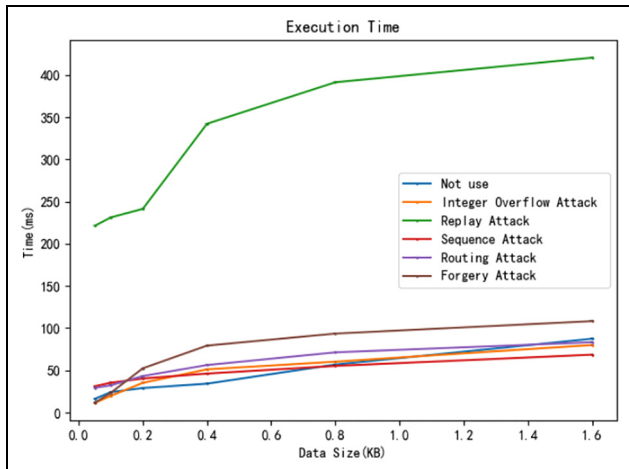
It can be found that compared with Figure 11, the crosschain transaction forgery attack shown in Figure 12 has a big change, which is due to the need for setting the *nonce* value and timestamp before the transaction starts in order to prevent forgery. So the preparation time of the relay node will be longer. The time cost from low to high is crosschain integer overflow attack, crosschain routing attack, crosschain transaction forgery attack, crosschain transaction replay attack, and crosschain sequence attack. The main reason for the high time cost of the crosschain transaction sequence attack is that it is necessary to check the value of the *state* variable every time, which is equivalent to invoking the crosschain chaincode once again.

As can be seen from Figure 13, the time cost of crosschain transaction replay attack is the largest. This





**Figure 12.** Time cost after taking different protective schemes (remove startup time).



**Figure 13.** Time cost with data size after taking different protective schemes (remove startup time).

is because it adopts a signature mechanism to prevent replay. As a result, when the number of transactions remains unchanged, the larger the transaction data is, the more the time spent in the signature and verification process. Based on all the experimental results, we can find that compared with the system startup time, in most cases, the time overhead caused by these five protection schemes to the crosschain system is not high. In most cases, there will be no such large-scale and large amount of transactions between different Fabric blockchains. More often, the connection will be disconnected after a small number of transactions. The next transaction needs to restart the system to establish a channel. Therefore, most of the time is spent on system startup. In other words, the five protection methods proposed in this article will not cause a large time delay to the crosschain system, and are relatively efficient.

## Conclusion

In this article, based on BitXHub, the open-source crosschain project, we analyze and implement five attacks on different layers in the Fabric isomorphic crosschain system, such as crosschain integer overflow attack, crosschain transaction forgery attack, crosschain transaction replay attack on chaincode layer and crosschain transaction sequence attack, and crosschain routing attack on network layer. This article also introduces the application scenarios for each attacks in detail, and puts forward the corresponding protection schemes and security analysis for different attacks. This security analysis can be widely used in a variety of Fabric consortium crosschain systems. It is not only beneficial to find the potential attacks of crosschain systems and provide effective schemes for crosschain security, but can also promote the security application of consortium crosschain system in bonds, stocks, financial derivatives, cross-border remittance, supply chain, and other aspects, bringing more economic benefits.

For the future work, our primary goal is to go deep into the data layer and study the attack protection from identity authentication. In addition, we also plan to implement attacks and protection schemes between Fabric, BTC, ETH, LTC, and other heterogeneous blockchains.

## Declaration of conflicting interests


The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

## Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the National Key RD Program of China under Grant 2020YFB1005604, partly supported by the National Natural Science Foundation of China (No. 61902021), Beijing Natural Science Foundation Municipality (No. 4212008).

## ORCID iDs

Di Wu  <https://orcid.org/0000-0002-1689-4577>

Li Duan  <https://orcid.org/0000-0002-1825-7416>

## References

1. Nakamoto S. Bitcoin: a peer-to-peer electronic cash system. Consulted.
2. Abichandani P, Lobo D, Kabrawala S, et al. Secure communication for multi-quadrotor networks using Ethereum blockchain. *IEEE Internet Things J* 2021; 8(3): 1783–1796.



3. Lee S, Kim D, Kim D, et al. Who spent my EOS? On the (in) security of resource management of EOS.IO. In: *13th USENIX workshop on offensive technologies, WOOT 2019* (eds A Gantman and C Maurice), Santa Clara, CA, August 12–13, 2019. Berkeley: USENIX Association.
4. Holbrook J. *Blockchain security and threat landscape. Architecting enterprise blockchain solutions*. New York: Wiley, 2020.
5. Brown RG, Carlyle J, Grigg I, et al. *Corda: an introduction*. New York: R3 CEV, 2016.
6. Androulaki E, Barger A, Bortnikov V, et al. Hyperledger Fabric: a distributed operating system for permissioned blockchains. In: *Proceedings of the thirteenth EuroSys conference*, Porto, 23–26 April 2018, pp.1–15. New York, NY: ACM.
7. Burdges J, Cevallos A, Czaban P, et al. Overview of Polkadot and its design considerations. arXiv:2005.13456, 2020.
8. Yoo YC, Park SY, Jeong JW, et al. A study on the electronic medical record system and cosmos block chain. *Korean J Ind Secur* 2019; 9(2): 137–159.
9. Jimmyshi and Shareong. Wecross: a collaboration platform of crosschain. <https://github.com/WeBankBlockchain/WeCross> (accessed 19 February 2020).
10. Shao-Jie YE, Xiao-Yi W, Cai-Chao X, et al. BitXHub: side-relay chain based heterogeneous blockchain interoperable platform. *Comput Sci* 2020; 47(06): 300–308.
11. Aitong LU, Zhao K, Yang J, et al. Research on cross-chain technology of blockchain. *Netinfo Secur* 2019; 19: 83–90.
12. Schwartz E. A payment protocol of the web, for the web: or, finally enabling web micropayments with the Interledger Protocol. In: *Proceedings of the 25th international conference on world wide web, WWW 2016* (eds J Bourdeau, J Hendler, R Nkambou, et al.), Montreal, QC, Canada, 11–15 April 2016, pp.279–280. New York: ACM.
13. Wu J and Jiang S. Local pooling of connected supernodes in lightning networks for blockchains. In: *2020 IEEE international conference on blockchain, blockchain 2020*, Rhodes Island, 2–6 November 2020, pp.421–427. New York: IEEE.
14. Na D and Park S. Fusion chain: a decentralized lightweight blockchain for Iot security and privacy. *Electronics* 2021; 10(4): 391.
15. Pillai B, Biswas K and Muthukkumarasamy V. Cross-chain interoperability among blockchain-based systems using transactions. *Knowl Eng Rev* 2020; 35: E23.
16. Robinson P. Consensus for crosschain communications. arXiv:2004.09494, 2020.
17. Robinson P, Hyland-Wood D, Saltini R, et al. Atomic crosschain transactions for Ethereum private sidechains. arXiv:1904.12079, 2019.
18. King S and Nadal S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake, self-published paper, August, vol.19, 2012. <https://decred.org/research/king2012.pdf>
19. Jiao J, Kan S, Lin S-W, et al. Semantic understanding of smart contracts: executable operational semantics of solidity. In: *2020 IEEE symposium on security and privacy, SP 2020*, San Francisco, CA, 18–21 May 2020, pp.1695–1712. New York: IEEE.
20. Naumenko G, Maxwell G, Wuille P, et al. Erelay: Efficient transaction relay for bitcoin. In: *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security, CCS 2019* (eds L Cavallaro, J Kinder, X Wang, et al.), London, UK, 11–15 November 2019, pp.817–831. New York: ACM.
21. Tikhomirov S, Moreno-Sanchez P and Maffei M. *A quantitative analysis of security, anonymity and scalability for the lightning network*. Lyon: IACR Cryptology Eprint Archive, 2020.
22. Küsters R, Rausch D and Simon M. *Accountability in a permissioned blockchain: formal analysis of Hyperledger Fabric*. Lyon: IACR Cryptology Eprint Archive, 2020.