electronics

Article

Toward a Secure and Private Cross-Chain Protocol Based on Encrypted Communication

Yuli Wang [1,2], Zhuo Chen [1,2], Ruihe Ma [3], Bin Ma [1,2,*], Yongjin Xian [1,2,*] and Qi Li [1,2]

1 Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences), Jinan 250316, China; wyl@qlu.edu.cn (Y.W.); 10431220485@stu.qlu.edu.cn (Z.C.); qluliqi@sdas.org (Q.L.)
2 Shandong Provincial Key Laboratory of Computer Networks, Shandong Fundamental Research Center for Computer Science, Jinan 250014, China
3 School of Economics, Jilin University, Changchun 130012, China; 20007601@sdufe.edu.cn
* Correspondence: mabin@sdas.org (B.M.); xianyj@sdas.org (Y.X.)

Abstract: Blockchain technology is becoming more prominent and is being used in many different industries. Data islands have emerged as a result of the difficulty in transferring assets and exchanging information between blockchains because of differences in the underlying technology. Cross-chain technology is becoming increasingly prevalent as a solution to the data security problem. Decentralized blockchain networks frequently use the Hashed Timelock Contract (HTLC) to solve the problem of balancing atomicity and time sensitivity. However, it suffers from drawbacks such as limited security and privacy protection capabilities. To overcome these limitations, a secure and fully functional system named the Exchange Smart Contract (ExchangeSC) has been developed; the ExchangeSC can integrate smart contracts and Paillier homomorphic encryption into the Mid-Account HTLC (MA-HTLC) cross-chain protocol. This integration effectively resolves the problem of low security and privacy protection in the HTLC cross-chain protocol. Specifically, the locked information in the solution is encrypted using homomorphic encryption before uploading to the blockchain, which is operated by participating nodes in the ciphertext domain. The ExchangeSC demonstrates reasonable performance on the official testing network's EVM platform. Further evaluation of the ExchangeSC-based HTLC cross-chain reveals its superior security and lower time cost compared to the BitXHub cross-chain project.

Keywords: blockchain; cross-chain; Paillier homomorphic encryption; security and privacy

check for updates

Citation: Wang, Y.; Chen, Z.; Ma, R.; Ma, B.; Xian, Y.; Li, Q. Toward a Secure and Private Cross-Chain Protocol Based on Encrypted Communication. *Electronics* **2024**, *13*, 3116. https://doi.org/10.3390/electronics13163116

Academic Editor: Hung-Yu Chien

Received: 22 May 2024
Revised: 20 June 2024
Accepted: 3 July 2024
Published: 7 August 2024

1. Introduction

1.1. Background and Motivation

With the development of information technology, privacy information protection is increasingly important [1]. The blockchain was first introduced accompanied by Satoshi Nakamoto's publication of the Bitcoin white paper in 2008 [2]. Blockchain has attracted attention from various sectors of society and major scientific institutions due to its characteristics of decentralization, public maintenance, transparency, tamper resistance, and traceability [3,4]. These features make the blockchain more suitable for data management systems with high-security requirements [5,6]. Blockchain technology and asset trading are closely related. We are currently in the era of blockchain 3.0, represented by Consortium Blockchain, and blockchain technology has been widely applied. Hyperledger Fabric [7], led by the Linux Foundation and founded by 30 initial enterprise members, including IBM, is the largest Consortium Blockchain platform. With the continuous development of blockchain technology, more and more blockchain projects have emerged [8]. There are many differences in the underlying technologies among chains, including their consensus and data structures. This leads to the inability to interact between chains, resulting in data islands. To make each blockchain no longer like an island in the ocean, it is urgent to address the issues of asset transfer and data exchange between heterogeneous blockchains.

Currently, mainstream cross-chain technologies include notary schemes, relays/sidechains, and Hashed Timelock Contracts [9,10]. The current research status of cross-chain technologies was analyzed, as described below. Notary schemes are cross-chain schemes created based on the interledger [11] protocol, which introduces a third party that is jointly trusted by both parties to act as a notary. Notaries are usually appointed independent nodes or institutions, and they undertake the tasks of data collection, transaction confirmation, and verification simultaneously. The notary scheme utilizes smart contracts to operate between chains, which is convenient. Still, it is prone to a centralized mode, which conflicts with the decentralized blockchain concept. In relays/sidechains, a sidechain refers to another blockchain system with independent functions that can actively perceive the primary chain's information and take corresponding actions. The relay acts as a communication channel when the main and sidechains exchange value and information. A relay refers to combining a sidechain and notarization mechanism to complete the functions of collecting, verifying, and forwarding messages. Westerkamp et al. [12] facilitate reliable cross-chain proofs of soundness by implementing a chain relay to verify the block headers of proof-of-work blockchains. For proof-of-stake [13] and proof-of-work [14] blockchains, Yin et al. [15] proposed an efficient sidechain structure with fast cross-chain transmission speed and small certificate size through a new cross-chain certificate generation process and committee election method. However, the technical implementation of the sidechain is complex, and due to the additional complexity of relays/sidechains, the transaction speed could be faster.

The atomic swap provides cross-chain asset exchange without involving any trusted third parties. The HTLC can achieve atomic swaps in cross-chain asset transactions and is the most widely used asset transaction control algorithm. In 2018, Maurice Herlihy [16] proposed atomic cross-chain swaps based on a strongly connected graph model. To protect the identity privacy of cross-chain transaction parties, Cai et al. [17] proposed the Paillier Timelock Contract (PTLC) in their paper, which uses Paillier homomorphic encryption (PHE) instead of hash encryption. Monika et al. [18] proposed a solution for implementing an atomic swap between public blockchains using an HTLC. They also formulated the timelock equations using the confirmation time of probabilistic blockchains to be used in the HTLC. Barbara [19] proposed Multi-Protocol HTLC (MP-HTLC), which makes multiple users' exchange tokens on different blockchains in a single instantiation of the protocol without any leadership elections. Among the existing HTLC cross-chain protocols, the Mid-Account HTLC (MA-HTLC) protocol proposed by Liu et al. [20] is the most advanced. They creatively introduced the concept of an account on Fabric, set up different mid-accounts in the transfer for asset custody and transfer, and promptly destroyed them after the completion of the transaction. This protocol applies to the Consortium Blockchain represented by Fabric and the public chain represented by Bitcoin and Ethereum, expanding the HTLC cross-chain protocol application scenario. However, there are also some shortcomings in the protocol. The transaction-locking information is transmitted in clear text. If an adversary steals it, the adversary can masquerade as the client, communicate with the blockchain, and steal the client's assets, significantly reducing the transaction's security and resulting in the low anti-attack ability of the protocol. Homomorphic encryption allows computations to be performed on encrypted data without decrypting them, preserving privacy and security. This enables secure data processing in cloud environments and protects sensitive information during analysis.

1.2. Contributions

To solve the above problems, we designed and introduced ExchangeSC, a decentralized system that ensures clients' identity privacy while facilitating secure information exchange. By combining smart contracts with Paillier homomorphic encryption [21], this protocol can be easily implemented in practical scenarios. The primary contributions of this study are summarized as follows.

- Enhanced Security and Privacy: By employing the Paillier homomorphic encryption algorithm, we ensure that the locked transaction information is securely encrypted before being uploaded to the blockchain. This encryption protects the data from unauthorized access and tampering during transmission.
- Identity Privacy Protection: The integration of an identity authentication module prevents adversaries from masquerading as legitimate clients, thereby safeguarding the participants' identities throughout the transaction process.
- Decentralization and Practical Deployment: ExchangeSC maintains the decentralized nature of blockchain technology and is designed for easy deployment in real-world scenarios. The system includes a reward and punishment mechanism to mitigate malicious behaviors by nodes, ensuring the integrity and reliability of cross-chain transactions.
- Performance Efficiency: Implementing ExchangeSC within the Ethereum Virtual Machine (EVM) has shown reasonable performance, with the time cost of cross-chain transactions being only marginally higher than that of the original protocol. Moreover, ExchangeSC outperforms established projects such as BitXHub in terms of security and time efficiency.

In addition to Section 1, the remainder of this paper proceeds as follows: Section 2 provides an overview of the related knowledge, such as blockchain technology, smart contracts, the MA-HTLC, and Paillier homomorphic encryption. Section 3 introduces the ExchangeSC in detail. The security and performance evaluations of ExchangeSC are presented in Sections 4 and 5, respectively. Finally, Section 6 provides the conclusion.

2. Related Knowledge

2.1. Blockchain Technology

The blockchain utilizes a distributed data storage model with decentralization, transparency, tamper resistance, and traceability, and it records all transactions in a peer-to-peer network [22]. In a blockchain system, each node maintains a blockchain replica. Figure 1 demonstrates how the information of the transaction is publicly uploaded into the matching block with the *Merkle* tree and how this block points to its previous block by recording its hash. Thus, the information on the blockchain cannot be altered. Additionally, millions of incentive-driven nodes are expected to confirm the legitimacy of transactions as they become publicly recorded. The consensus mechanism implemented in the blockchain system makes it mandatory for all nodes to follow uniform rules. Then, the consistency of the content of records belonging to different bookkeeping nodes is stored in a distributed accounting system.

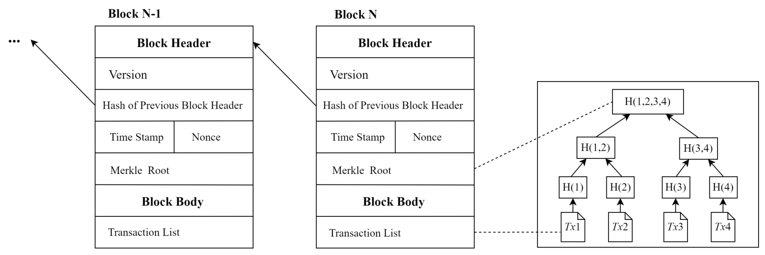

Figure 1. Blockchain structure.

2.2. Smart Contract

The smart contract is a digital contract based on blockchain technology, and it is a transaction protocol implemented by computer code provided to distributed participants to follow. A smart contract can be used to build a digital rule to automatically execute transactions under specific conditions. Specifically, a smart contract is a program created through contract transactions, and it allows participants to automatically execute transactions. With smart contracts, transactions can be completed automatically after specific conditions have been satisfied. As the contract code is stored on a public and tamper-proof blockchain, smart contracts are trustworthy and reliable. The smart contract in Ethereum can be regarded as an external account with a contract address. When a contract is invoked, the contract parameters are used as input, and then the transaction is sent to the contract address after paying the contract with ETH. Gas is the execution cost of each operation in Ethereum. Each process in the Ethereum Virtual Machine (EVM) consumes gas. The more data are computed and stored, the more gas is consumed. Each transaction invoking the contract must include a gas limit and a gas price, where the gas limit is the maximum gas consumed in a transaction, and the gas price is determined by the users and the nodes.

2.3. MA-HTLC

The HTLC protocol, derived from the lightning network [23], facilitates decentralized conditional payments among multiple users, eliminating the necessity for a trusted third-party intermediary. It is widely used in atomic swap and cross-chain transactions. Hash locking entails that a commitment is valid if the pre-image for a given hash value H is provided; otherwise, it becomes invalid. In case of transaction failure due to various reasons, the implementation of a timelock allows all parties to retrieve their funds and mitigate potential losses arising from fraudulent activities or transaction failures. This protocol introduces an accounting system into the Fabric blockchain and integrates smart contract technology to facilitate secure and seamless asset exchange between Ethereum and the Fabric network. During the transfer, distinct intermediary accounts are established for asset custody and transfer purposes, which are promptly eliminated upon transaction completion, thereby preserving the original cross-chain transaction rate while ensuring transaction security.

Figure 2 demonstrates that the protocol process consists of the following steps.

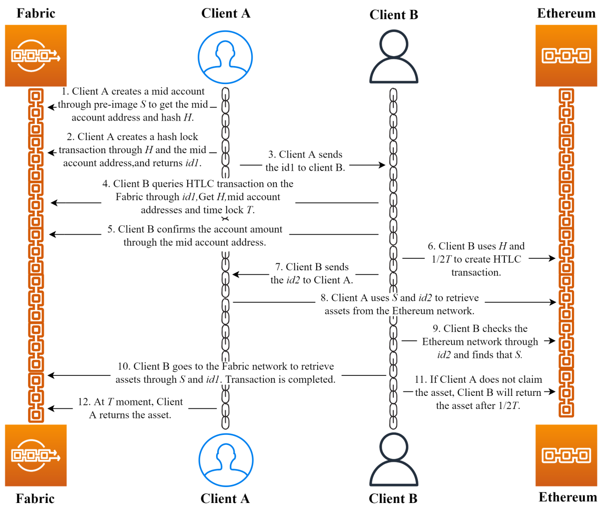
Fabric
Client A
Client B
Ethereum
1. Client A creates a mid account through pre-image S to get the mid account address and hash H.
2. Client A creates a hash lock transaction through H and the mid account address,and returns id1.
3. Client A sends the id1 to client B.
4. Client B queries HTLC transaction on the Fabric through id1,Get H,mid account addresses and time lock T.
5. Client B confirms the account amount through the mid account address.
6. Client B uses H and 1/2T to create HTLC transaction.
7. Client B sends the id2 to Client A.
8. Client A uses S and id2 to retrieve assets from the Ethereum network.
9. Client B checks the Ethereum network through id2 and finds that S.
10. Client B goes to the Fabric network to retrieve assets through S and id1. Transaction is completed.
11. If Client A does not claim the asset, Client B will return the asset after 1/2T.
12. At T moment, Client A returns the asset.
Fabric
Client A
Client B
Ethereum

Figure 2. Flowchart of the MA-HTLC.

Step 1. Client A creates a mid-account through a pre-image, receives the mid-account address *midadress* and hash $H(S)$, creates a hash-lock transaction on Fabric, returns a transaction lock id_1, and sends it to Client B.

Step 2. Client B receives the $H(S)$ and time-lock T by querying the locked asset transaction and confirms the account balance through *midadress*. Then, $H(S)$ and $T/2$ are used to create a hash timelock transaction on Ethereum, and the transaction lock id_2 is returned and sent to Client A.

Step 3. After receiving id_2, Client A will use id_2, and S will take out the assets from Ethereum. If Client A does not retrieve the assets after $T/2$, the transaction will be deemed as a failure, and Client B can retrieve the assets.

Step 4. Client B queries the locked asset status on Ethereum through id_2. If the asset has been taken, the query result returns pre-image S. Then, Client B communicates with Fabric through S and id_1 to retrieve the assets that Client A has locked on it. If Client B fails to take out the asset within the time limit, it is deemed that the asset exchange failed, and Client A can retrieve the asset.

2.4. Paillier Homomorphic Encryption

The Paillier homomorphic encryption algorithm is an encryption algorithm based on public-key cryptography. By performing operations in the ciphertext domain and decrypting the ciphertext, it can match the operations in the plaintext domain and encrypt the plaintext. Therefore, data privacy can be protected while searching and calculating ciphertext. The relevant theoretical knowledge of the algorithm is provided in the following.

2.4.1. Encryption and Decryption Process of the Paillier Algorithm

(1) Key Generation
Two large prime numbers p and q are randomly selected such that $\gcd(pq,(p-1)(q-1)) = 1$, where $\gcd(a,b)$ denotes the greatest common divisor of a and b. $n = pq$ and $\lambda = \mathrm{lcm}(p-1,q-1)$ are calculated, where $\mathrm{lcm}(a,b)$ denotes the least common multiple of a and b. $g \in Z^*_{n^2}$ is selected, and $\gcd\left(L\left(g^\lambda \,\mathrm{modn}\, n^2\right), n\right) = 1$ is satisfied. It is ensured that n divides the order of g by checking the existence of the following modular multiplicative inverse: $\mu = \left(L\left(g^\lambda \bmod n^2\right)\right)^{-1} \bmod n$, where $L(x) = \frac{x-1}{n}$. The public key pk is (n,g), and the private key sk is (λ,μ).

(2) Encryption
Let m be the plaintext; a random number $r \in Z^*_{n^2}$ is selected. m is encrypted to obtain ciphertext c.

$$c = g^m r^n \bmod n^2$$

(3) Decryption
Let c be the ciphertext, where $c \in Z^*_{n^2}$. c is decrypted to obtain plaintext m.

$$m = \frac{L\left(c^\lambda \,\mathrm{modn}\, n^2\right)}{L\left(g^\lambda \,\mathrm{modn}\, n^2\right)} \bmod n$$

2.4.2. The Additive Homomorphism Property of Paillier's Algorithm

The value of $\mathrm{Enc}(a+b)$ can be calculated by multiplying $\mathrm{Enc}(a)$ and $\mathrm{Enc}(b)$. The real value of $a+b$ can be obtained after decryption by the person who has pk.

In the Paillier algorithm, adding in the plaintext domain is equivalent to multiplying in the ciphertext domain.

$$\mathrm{Enc}(a+b) = \mathrm{Enc}(a) \cdot \mathrm{Enc}(b)$$

3. ExchangeSC Design

To address the issue of inadequate security and client privacy in the MA-HTLC protocol, we developed ExchangeSC, an information security exchange system based on Paillier homomorphic encryption. Additionally, we incorporated a blockchain depository function to document any transaction-related disputes. Conceptually, Figure 3 demonstrates that ExchangeSC implements a scheme for secure information exchange that safeguards client privacy. Clients are provided with an Application Programming Interface (API) for information exchange through the encryption API within ExchangeSC, which must interact with the smart contract via the authoritative node P_a. The authoritative node is either assumed or selected by the authoritative institution and possesses a reliable identity. The smart contract deployed on the blockchain serves as an incentive mechanism to ensure secure information exchange for all parties involved.

In this section, we describe the three-part structure of ExchangeSC. ExchangeSC implements a hybrid blockchain system that supports client information security exchange, client privacy protection, and exchange record supervision and storage through smart contracts and an information transmission scheme based on Paillier homomorphic encryption. In ExchangeSC, the three parts interact seamlessly; the client invokes the interface in the Paillier encryption API, and the algorithm in the Paillier encryption API is completed in the blockchain and through client participation.

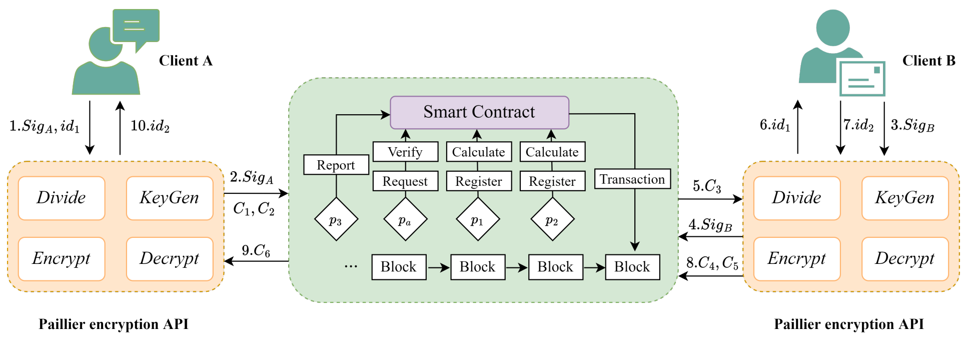

Figure 3. System architecture of ExchangeSC.

3.1. Client Part

The client part is composed of Client A, Client B, and the interaction with the Paillier encryption API. This part connects two types of entities: Client A and Client B. Both Client A and Client B interact directly with local encryption rather than directly with the blockchain part.

Client A: This is the initiator of information exchange and hopes to initiate a regulated information exchange.

Client B: This is the receiver of the information exchange. After receiving and verifying the correctness of the information sent by A, Client B sends their information to A.

The client invokes the API interface, converts the hexadecimal hash value into a decimal number, decomposes the decimal number into two numbers, and adds them through *Div*; then, they obtain the public–private key pair through *KeyGen*, encrypt the two numbers through *Enc*, and encrypt the identity information with the private key for generation. Finally, the signature and ciphertext information is uploaded to the authoritative node in the blockchain. When receiving the ciphertext information, it is decrypted with *Dec* to obtain the original plaintext information.

3.2. Paillier Encryption API Part

The Paillier homomorphic encryption API aims to encrypt information and meet client requirements. In addition, encryption is directly associated with the authoritative node, so the API can participate in the calling of the smart contract through the authoritative node. When the cross-chain transaction starts, the APIs of both clients are deployed and synchronized by a Key Generation Center (KGC) to ensure that the *KeyGen* generates the same public–private key pair. Below, we will introduce the working principle.

(1) The signature information uploaded by the client *Sig* is received and saved.
(2) A *Div* interface is provided. The client first inputs id_1/id_2 into the API, then invokes Div to convert the hexadecimal hash value into a decimal number, and randomly decomposes it into the following two numbers in the form of addition for the subsequent encryption.
(3) Clients are provided with the *KeyGen* interface to generate public and private key pairs (sk, pk) for Paillier encryption.
(4) An encryption interface *Enc* and decryption interface *Dec* are provided. Clients can encrypt $(id_1', id_1'')/(id_2', id_2'')$ to $(C_1, C_2)/(C_4, C_5)$ by invoking *Enc*, and they can decrypt C_3/C_6 to plaintext information by invoking *Dec*. Finally, the decimal plaintext information is converted into a hexadecimal hash value to obtain id_1/id_2.

3.3. Blockchain Part

The blockchain is the most essential part of the system, as it is used to exchange information between the two parties in a secure manner and to store and supervise the exchanged information. Nodes in the blockchain network are recruited, and the smart contract is invoked to multiply the ciphertext pair and send the result to P_a. The participating node P_i will only receive the ciphertext pair encrypted by the Paillier algorithm and cannot receive any other relevant information or any client information. Smart contracts are used to record registration information and transparently execute services, which are finally recorded by the blockchain. The smart contract consists of the following five algorithms.

(1) ***Registration***: In this part, we introduce the process of node registration (Algorithm 1). A node P_i in the blockchain obtains rewards by providing services calculated on the ciphertext domain. It must register by paying a deposit to the smart contract and filling in the registration information. The deposit is used to punish the improper behavior of a node. If a node provides an incorrect answer or fails to provide an answer, it will lose part of the deposit. The registration information is as follows.

- Registered deposit.
- Node address.
- The slowest service completion time.
- The number of completed services (initially 0, +1 after correct service completion, and -1 after incorrect service completion).

The registration information is recorded in the smart contract, and each entity in the blockchain network must comply with the smart contract protocol, so the smart contract acts as the intermediary between an entity and the blockchain network.

Algorithm 1 Registration

```
Input parameters: node address, deposit, slowest service time
Output: RegistrationInfor
1: function register  (node address, deposit, slowest service time)
2: contract.sendDeposit (node address, deposit)
3: contract.registerNode (node address, deposit, slowest service time)
4: end
5: return RegistrationInfor
```

(2) ***Request***: In this part, we will explain the process of the node P_a requesting services (Algorithm 2). When the authoritative node receives the transaction information, it requests a service from the smart contract and submits a service request, which consists of ciphertext information, the number of participating service nodes, a registration deposit demand, a service completion time demand, and a slowest completion time demand. Then, according to the service request, the smart contract selects the nodes whose registration deposit is greater than the registration deposit demand, whose service completion times are greater than the service completion time demand, and whose slowest completion time is less than the slowest completion time demand to participate in this service, and it sends the ciphertext information to the selected node. If the slowest completion time of two nodes is the same, to select a more reliable node, the smart contract should select the node with more registration deposits and more service times. To ensure the efficiency of the service, when the deposit for registration is the same and the number of completed services is the same, the nodes with the slowest completion time will be preferentially selected for service. If the client has no requirements for service time, the smart contract should choose a node with higher integrity for service. To select a more secure and reliable node and complete the service request more quickly, a greedy algorithm is used as a solution to the problem. The request will be terminated if a qualified node cannot be found among the existing nodes.

Algorithm 2 Request

```
Input: ciphertext, num participants, deposit requirement, service time requirement, slowest service time requirement
Output: $P_i$
1: function request(ciphertext, num participants, deposit requirement, service time requirement, slowest service time requirement)
2: nodes = contract.getNodes()
3: valid nodes = [ node for node in nodes if node.deposit >= deposit requirement node.service times >= service time requirement node.slowest service time >= slowest service time requirement ]
4: valid nodes.sort(key=$\lambda$ x:(x.slowest service time, -x.deposit, -x.service times, x.node address))
5: if participants = valid nodes[:num participants] then
6:     for i in range(len(participants)) do
7:         node = participants[i]
8:         contract.sendCiphertext(n.node address,ciphertext)
9:     end for
10: end if
11: return $P_i$
```

(3) ***Verification***: In this part, P_a verifies the digital signature of the clients to confirm their identity.

(4) ***Calculation***: In this part, P_i multiplies the ciphertext in the ciphertext field to calculate the result and sends the result to P_a.

(5) ***Report***: In this part, nodes that have not participated in the service but have paid the deposit can report the misconduct of the participating service nodes (Algorithm 3). Misconduct includes miscalculation and negative services (not calculated), which are rewarded through the smart contract. The two forms of misconduct are reported as follows.

- To deal with the improper behavior of miscalculation, we designed a service mode. When a service request is received, the smart contract selects the P_i with service number n. Assuming that the P_i returns two different results to P_a through calculation, both results will be sent to the client's local API. The client first decrypts the results of a larger number of calculations and decrypts another result if this fails. When the client completes this exchange service, it will broadcast the ciphertext information to the blockchain network. The node that does not participate in this service but has paid the deposit will calculate and send the report information to the smart contract. The report information includes the registered deposit and misconduct of the nodes. Similarly, the registration deposit of the reporting node is also more than the requirement, thus reducing the behavior of malicious reporting. Then, the smart contract check results show that if the P_i does have the improper behavior of miscalculation, part of its registration deposit will be deducted from the reporting node.
- To deal with the improper behavior of providing a negative service, the nodes in the blockchain network can detect the service record and whether the P_i returned the results within the shortest service time. If not, negative service behavior is found. The report information is sent to the smart contract, including the registered deposit and misconduct. Then, the smart contract checks the service record. If it is indeed as shown in the report information, the reporting node will obtain part of the deposit of the P_i. Otherwise, it will deduct part of its deposit.

If a node in the blockchain network does not provide report information within a certain period of time, the P_i will be rewarded. Note that the reporting process is carried out after the client obtains the ciphertext information, so it will not affect the efficiency.

The blockchain should meet the following three essential requirements to provide security, privacy, and fairness.

- ***Privacy***: The blockchain protects the privacy of the client's identity, and ordinary participating nodes cannot obtain relevant client information.
- ***Security***: P_a and P_i in the blockchain cannot deduce any information related to plaintext information through ciphertext.
- ***Fairness***: The blockchain's reward and punishment mechanisms ensure the calculation results' correctness. Malicious nodes will be detected after service, and their deposit will be deducted. At the same time, the blockchain also has a *Report* mechanism, and the whole process is recorded in the blockchain network, which means that malicious nodes that fail to provide services correctly for any reason will be detected by the smart contract and punished.

Algorithm 3 Report

```
Input: registration deposit, improper behavior
Output: reportInfor
1:  function report(registration deposit, improper behavior)
2:  nodes = contract.getNodes()
3:  report nodes = [node for node in nodes if node.deposit = registration deposit]
4:  if improper behavior == “improper computation” then
5:      for i in range(len(report nodes)) do
6:          node = report nodes[i]
7:          result1 = contract.getResult(node.node address, 1)
8:          result2 = contract.getResult(node.node address, 2)
9:          contract.sendReport(node.node address, “improper computation”, result1, result2)
10:     end for
11:     service records = contract.getServiceRecords()
12:     for i in range(len(service records)) do
13:         record = service records[i]
14:         if record.finishT-record.startT = record.slowest service time and not record.resultprovided then
15:             node = contract.getNode(record.provider address)
16:             if node.deposit = registration deposit then contract.sendReport(node.node address, “passive service”, None, None)
17:             end if
18:         end if
19:     end for
20: end if
21: return reportInfor
```

3.4. Workflow of ExchangeSC

The specific workflow of the system is divided into the following 11 steps. Figure 3 demonstrates further details.

Step 1. Client A inputs Sig_A and id_1 into the local API.

Step 2. The API divides id_1 as $id_1 = id_1' + id_1''$ and uses the Paillier homomorphic encryption algorithm to encrypt id_1' and id_1''; then, the ciphertext information C_1 and C_2 is obtained. The API uploads C_1, C_2, and Sig_A to P_a.

Step 3. Client B inputs Sig_A into the local API, which uploads Sig_A to P_a.

Step 4. P_a invokes *Verify* to decrypt Sig_A and Sig_B using their respective public keys, which verifies the identity of both.

Step 5. P_a sends C_1 and C_2 to P_i. P_i invokes Calculate to *calculate* the result of the multiplication operation on the ciphertext domain for C_1 and C_2 and then transmits the result of the respective calculation C_3 to P_a. After that, P_a records the information of P_i and sends C_3 to the local API for Client B.

Step 6. Client B uses the API to decrypt C_3 and verify the correctness of id_1.

Step 7. Client A inputs id_1 into the local API.

Step 8. The API divides id_2 as $id_2 = id'_2 + id''_2$ and uses the Paillier homomorphic encryption algorithm to encrypt id'_2 and id''_2; then, it receives ciphertext information C_4 and C_5. The API uploads C_4 and C_5 to P_a.

Step 9. P_a sends C_4 and C_5 to P_i. P_i invokes Calculate to *calculate* the result of the multiplication operation on the ciphertext domain for C_4 and C_5 and sends the result of the respective calculation C_6 to P_a. Then, P_a records the information of P_i and sends C_6 to the local API for Client A.

Step 10. Client B uses the API to decrypt C_6 to obtain id_2 and verify the correctness of id_2.

After the above steps are completed and no nodes participating in the service have invoked *Report*, the exchange process is considered to be completed, and the transaction information is packaged and uploaded to the blockchain for storage and supervision.

4. Correctness and Security Analysis

This section analyzes and demonstrates the correctness and security perspectives of ExchangeSC.

4.1. Correctness Analysis

The correctness of ExchangeSC is discussed in the following theorems.

Theorem 1. *Client A can obtain id_1 by decrypting C_3 using Dec.*

Proof. Select a random integer g where $g \in Z^*_{n^2}$, so $g^\lambda \equiv 1 \bmod n$ and $g^{n\lambda} \equiv 1\,\mathrm{mod n}\, n^2$. Then, select $id \in Z^*_{n^2}$, and let $g^\lambda = 1 + kn$. The private key (λ, μ) is used for decryption.

$$\begin{aligned}\mathrm{Dec}(C_3) =& L\left(C^\lambda \bmod n^2\right)\mu \bmod\ n \\ =& \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod\ n \\ =& \frac{L\left(\left(g^{id_1} r^n\right)^\lambda \bmod n^2\right)}{L(g^\lambda \bmod n^2)} \bmod\ n \\ =& \frac{L\left(\left(g^\lambda\right)^{id_1} \bmod n^2\right)}{L(g^\lambda \bmod n^2)}\end{aligned}$$

Substituting $g^\lambda = 1 + kn$ into the above equation, we find that

$$\begin{aligned}\mathrm{Dec}(C_3) =& \frac{L\left((1+kn)^{id_1} \bmod n^2\right)}{L(1+kn)} \\ =& \frac{id_1 \cdot k}{k} \\ =& id_1\end{aligned}$$

□

Theorem 2. *Client B can obtain id_2 by decrypting C_6 using Dec.*

Proof. The proof is omitted, as it is the same as that in Theorem 1. □

Theorem 3. *The client performs Div, $id = id' + id''$, and encrypts id in the plaintext domain. The node multiplies the ciphered id' and id'' in the ciphertext domain and obtains the same result as that after decryption.*

$$Enc(id) = Enc(id')Enc(id'')$$

Proof. Select a random integer g where $g \in Z^*_{n^2}$, so $g^{\lambda} \equiv 1\,\mathrm{modn}$ and $g^{n\lambda} \equiv 1 \bmod n^2$. Then, select an id and random numbers r_1, r_2. Here, $id \in Z^*_{n^2}$ and $r_1, r_2 \in Z^*_{n^2}$.

$$\begin{aligned}
\mathrm{Enc}\left(id'\right)\mathrm{Enc}\left(id''\right) &= \left(g^{id'} r_1^n \bmod \left(n^2\right)\right)\left(g^{id''} r_2^n \bmod \left(n^2\right)\right) \\
&= g^{id'+id''} (r_1 r_2)^n \bmod \left(n^2\right) \\
&= \mathrm{Enc}\left(id' + id''\right) \\
&= \mathrm{Enc}(id)
\end{aligned}$$

□

4.2. Security Analysis

ExchangeSC guarantees that only authenticated clients are entitled to ciphertext information through the blockchain node computing service.

Adversary Assumptions: We have made the following assumptions about the adversarial model.

- The key of the encryption API is generated by the KGC, and both sides of the encryption API are deployed and synchronized by the KGC, which is a trusted party; the key is not disclosed to anyone after it is generated.
- No party in the blockchain network can control the vast majority of computing power, i.e., no 51% attack can be launched, so all transactions and actions of nodes are publicly recorded in the blockchain and are tamper-evident.
- The communication channel between the crypto API and the blockchain network is protected by SSL/TLS and cannot be tampered with or leaked to nodes in the blockchain.

Adversary model: It is considered that there are two types of attackers in the system: internal and external. We assume that the internal attacker could be every node in the blockchain network, while the external attacker has not paid a deposit to become a registered node.

(1) *Internal Attackers*: An internal attacker can participate in the smart contract to perform a service, and it can obtain $(C_1, C_2)/(C_4, C_5)$. There are two ways to attack: committing misconduct in the service or reporting process but being rewarded and stealing the privacy of the client's identity. The proposed system implements security against both of these types of attacks.

- Since all actions of nodes are public in the blockchain network, their misbehaviors are recorded. The system's reporting mechanism can detect these misbehaviors and punish nodes that have done so.
- The client's identity information is not made public to the blockchain network but is sent by the API, which verifies the identity information. Therefore, an internal attacker cannot distinguish who it is actually providing services to, which effectively protects the privacy of the client's identity.

(2) *External Attackers*: An external attacker can eavesdrop on the ciphertext and access public transactions in the blockchain network. The goal of their attack is to obtain the ciphertext information and then decrypt it to gain id_1/id_2 and, thus, steal the assets that the client has locked on the chain.

The security of our system is based on the security of Paillier homomorphic encryption. The Paillier scheme meets the standard security definition of an encryption scheme: semantic security, that is, the indiscernibility of ciphertext under a chosen-plaintext attack (IND-CPA). The security of the scheme can be reduced to the decisional composite residuity assumption, which means that given a composite number n and an integer z, it is difficult to determine whether z is an nth-power residue under n^2. So far, no polynomial time algorithm can break through, and the security of the Paillier encryption scheme is considered reliable. So, even if the attacker eavesdrops on the ciphertext information, they cannot make inferences from the id_1/id_2 ciphertext information.

The security proof is given as follows.

Theorem 4. *If the original id is divided into n fragments, the id is indicated as a set* $A = \{a_1, a_2, \ldots, a_i, \ldots, a_j, \ldots, a_n\}, \forall a_i, a_j \in A$, $a_i \neq a_j$, $i < j$. *If* $\{a_i, \ldots, a_j\}$ *are extracted from A, thus obtaining* $A' = \{a_1, a_2, \ldots, a_{i-1}, a_{j+1}, \ldots, a_n\}$, *then the following formula holds.*

$$\mathrm{Dec}(\mathrm{Enc}(A - A')\,\mathrm{Enc}(A')) = \mathrm{Dec}(\mathrm{Enc}(A))$$

Proof. Assuming that the original formula is not tenable, the following formula holds.

$$\mathrm{Dec}(\mathrm{Enc}(A - A')\,\mathrm{Enc}(A')) \neq \mathrm{Dec}(\mathrm{Enc}(A))$$

It follows from the correctness of the homomorphism that this formula can be reduced to $\sim A' \cup A' \neq A$; this obviously contradicts the condition.

So, this assumption does not hold. □

Theorem 5. *If the original id is divided into n fragments, the id is indicated as a set* $A = \{a_1, a_2, \ldots, a_k \ldots, a_n\}$. *For any* $a_\tau \neq a_k$, *instead of* a_k, *and* $A' = \{a_1, a_2 \ldots, a_\tau \ldots, a_n\}$ *is obtained; then, the following formula holds.*

$$\mathrm{Dec}(\mathrm{Enc}(A)) \neq \mathrm{Dec}(\mathrm{Enc}(A'))$$

Proof. This mimics a substitution attack by an adversary. The correctness of the Paillier algorithm makes the sufficient condition for successful decryption the use of the correct set A. So, the replacement of the correct fragment obviously cannot result in the correct decryption result being obtained. □

Corollary 1. *If* $A = \{a_1, a_2, \ldots, a_{k-1}, a_k, a_{k+1} \ldots, a_n\}$, $1 < k < n$, *after deleting a certain* $a_k \in A$, $A' = \{a_1, a_2, \ldots, a_{k-1}, a_{k+1} \ldots, a_n\}$ *is obtained; then, the following inequality holds.*

$$\mathrm{Dec}(\mathrm{Enc}(A)) \neq \mathrm{Dec}(\mathrm{Enc}(A'))$$

Corollary 2. *If* $A = \{a_1, a_2, \ldots, a_n\}$, *adding* $\forall a_\tau \notin A$, $A' = \{a_1, a_2, \ldots, a_n, a_\tau\}$ *is obtained; then, the following inequality holds.*

$$\mathrm{Dec}(\mathrm{Enc}(A)) \neq \mathrm{Dec}(\mathrm{Enc}(A'))$$

According to Theorems 4 and 5 and their corollaries, the safety of the newly proposed ExchangeSC was demonstrated. If a customer's identity information is partially replaced, deleted, or added, it cannot be decrypted and verified. Thus, the security of the proposed ExchangeSC can be fully demonstrated in theory and application.

5. Experiment and Performance Evaluation

5.1. Experimental Environment

This prototype was deployed on 8 GB of Intel (R) Core (TM) i5-12500H memory CPU@2.54 on the GHz Ubuntu 18.04 system; the hard disk's storage space was 80 GB. The PHE API was coded in Python 3.7 and included four interfaces: *Div*, *KeyGen*, *Enc*, and *Dec*. The smart contract was coded using Remix IDE 0.42 [24] and the Solidity language [25]. The smart contract was designed with five functions: *Registration*, *Request*, *Verification*, *Calculation*, and *Report*. The smart contract was deployed, and the gas cost was tested on the official test network of Ethereum, Sepolia [26].

5.2. Encryption API Simulation

To evaluate the time overhead of the Paillier homomorphic encryption API, *Div*, *Enc*, and *Dec* were tested with the key size settings of 512 bits, 1024 bits, and 2048 bits, respectively. Repeated experiments of 200 operations were conducted to obtain the average time overhead, as shown in Table 1. For the process of encryption and decryption, the case of a 1024-bit key size was analyzed. The encryption took about 31.913 ms, and the decryption took about 9.648 ms, giving a total of 41.579 ms for the whole process, which shows that the time overhead is reasonable.

Table 1. Average time cost of 200 PHE operations (ms).

Operations	512-Bit	1024-Bit	2048-Bit
$Div(m) : m = m' + m''$	0.018	0.018	0.019
$Enc(m')Enc(m'')$	8.727	31.913	341.626
$Dec(Enc(m))$	1.597	9.648	71.619
$Overall$	10.342	41.579	413.264

5.3. Experiments on the Official Test Network

The ExchangeSC was deployed to the official Ethereum test network, Sepolia, which mimics a real production network, to demonstrate the practicality of ExchangeSC. The gas costs of *Registration*, *Request*, *Verification*, *Calculation*, and *Report* were tested separately. Figure 4 demonstrates that nodes invoking *Registration* cost roughly 0.00034 ETH, indicating that the cost of *Registration* is low enough that nodes do not abandon participation in the service due to the high registration fee. Similarly, the gas cost for the *Verification* and *Calculation* interfaces is also low at 0.00037 ETH and 0.00032 ETH, respectively. The gas cost is related to the complexity of the code, and the above three interfaces are simple, so the gas cost is low and the overhead is reasonable. *Request* requires the selection of participating service nodes. For one eligible node, it costs 0.00064 ETH. As it has a loop traversal operation, the gas cost increases linearly as the number of nodes increases, which is also reasonable. The code for the *Report* interface is relatively complex and requires a cost of 0.00098 ETH. Still, it only needs to be called in two situations: firstly, when the client receives two or more different calculation results; secondly, when the number of calculation results is less than the number of service nodes. Because there are few dishonest nodes in Sepolia, the *Report* mechanism is regarded as more of a threat.

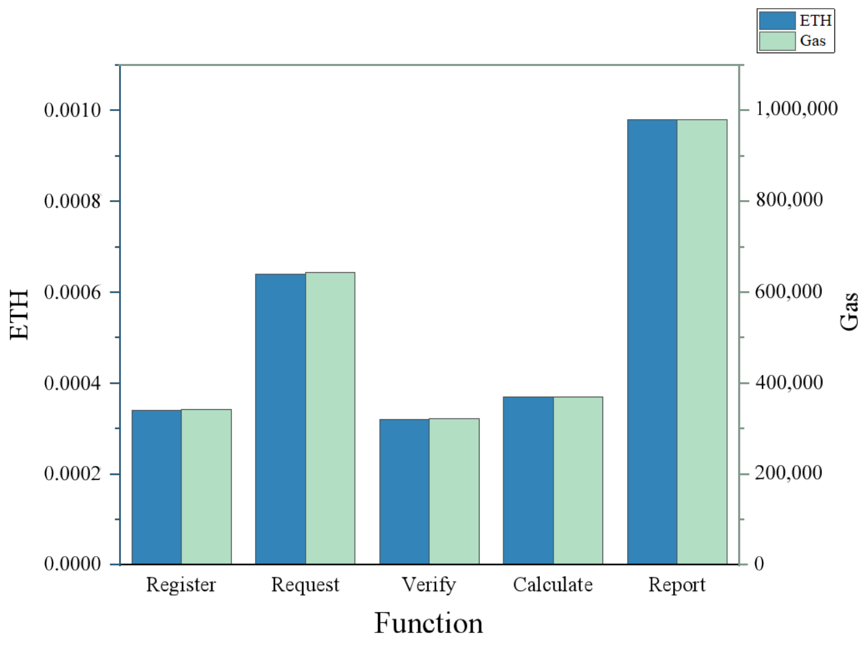

Figure 4. Gas costs in Sepolia.

5.4. Verification Experiment on the Theorems

We introduced some important fields and collections, as shown in Table 2, and then validated the theorems' correctness by conducting a series of experiments. Firstly, an *id* was generated, and then the API was utilized to divide it into four fragments. Subsequently, four experiments were conducted separately, and the parameters were changed four times in each experiment. As shown in Tables 3–6, the correctness of the theorems was verified by the experiments.

Table 2. Fields and collections.

Field	Description
id	input 22
A	$id\ fragments$ set
A'	$id\ fragments$ changing set
a_k	$id\ fragments = \{2, 5, 6, 9\}$
a_τ	an element used to replace or add
$Enc(id)$	$Enc(22) = 625\ldots132$

Table 3. Verification of Theorem 4.

Test	$\alpha = Enc(A')$	$\beta = \text{Enc}(\sim A')$	$Dec(\alpha \cdot \beta)$	True and False
1	546...158	448...549	22	Ture (✓)
2	846...113	244...647	22	Ture (✓)
3	715...484	184...754	22	Ture (✓)
4	436...122	137...641	22	Ture (✓)

Table 4. Verification of Theorem 5.

Test	$a_\tau \rightarrow a_k$	$Enc(A')$	$Dec(A')$	True or False
1	$3 \rightarrow 9$	386...754	16	False (×)
2	$8 \rightarrow 5$	911...521	25	False (×)
3	$1 \rightarrow 2$	315...775	21	False (×)
3	$7 \rightarrow 6$	476...667	23	False (×)

Table 5. Verification of Corollary 1.

Test	$a_\tau \rightarrow a_k$	$Enc(A')$	$Dec(A')$	True or False
1	9	228...697	13	False (×)
2	5	927...200	17	False (×)
3	2	466...454	20	False (×)
4	6	872...155	16	False (×)

Table 6. Verification of Corollary 2.

Test	$a_\tau \rightarrow a_k$	$Enc(A')$	$Dec(A')$	True or False
1	3	331...751	25	False (×)
2	4	275...312	26	False (×)
3	7	355...162	29	False (×)
4	8	390...661	30	False (×)

5.5. Cross-Chain Simulation

In this part, a full cross-chain transaction was run with the slowest time to complete the service set to 8 s to evaluate and analyze the performance of the protocol added to ExchangeSC. The experiments tested the time overhead of the proposed protocol, MA-HTLC, and BitXHub. Figure 5 demonstrates that the time cost of cross-chain transactions using the protocol is only 19.6% higher than that of MA-HTLC, and the additional time overhead is mainly used for secure exchanges in the ExchangeSC. This is a reasonable time cost to enhance the security of the protocol. The time cost of cross-chain transactions using this protocol is lower than that of the BitXHub project, which is a high-level improvement and shows that the improved protocol is efficient while meeting security requirements.

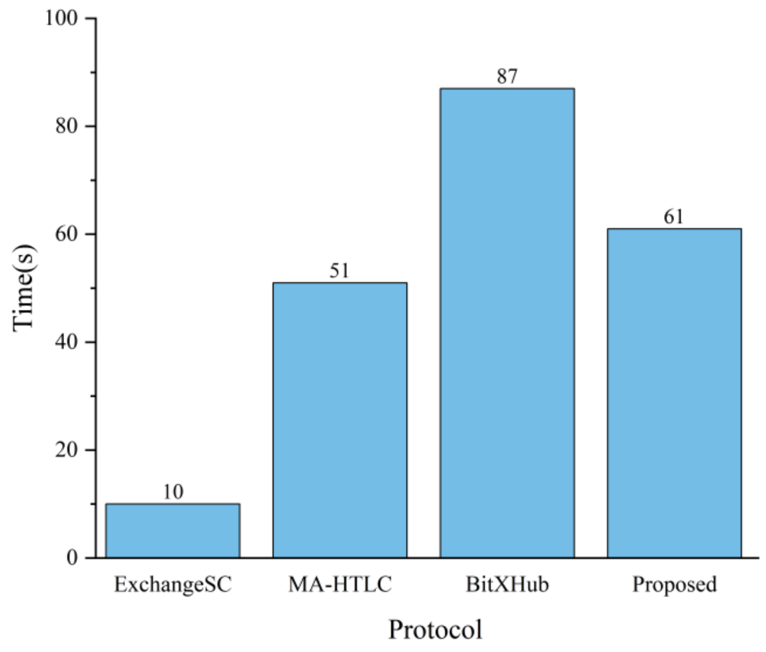

Figure 5. Time cost comparison.

To verify the robustness of this scheme and the correctness of the theorem, 50 nodes were set up, and 50 message exchanges were initiated. The percentages of malicious nodes were set to 20%, 40%, 60%, 80%, and 98%. Figure 6 demonstrates the success rate of message exchange in ExchangeSC. It can be observed that the transaction success rate of this scheme always remains at a high level and does not decrease with the increase in malicious nodes. In contrast, the success rate of other schemes decreases with the increase in malicious nodes. This robustness is achieved because the proposed scheme sends the calculation results of all nodes to the client, which decrypts the same number of results in order from most to least. Therefore, the client may still access the actual data even if 49 of the 50 nodes are malicious.

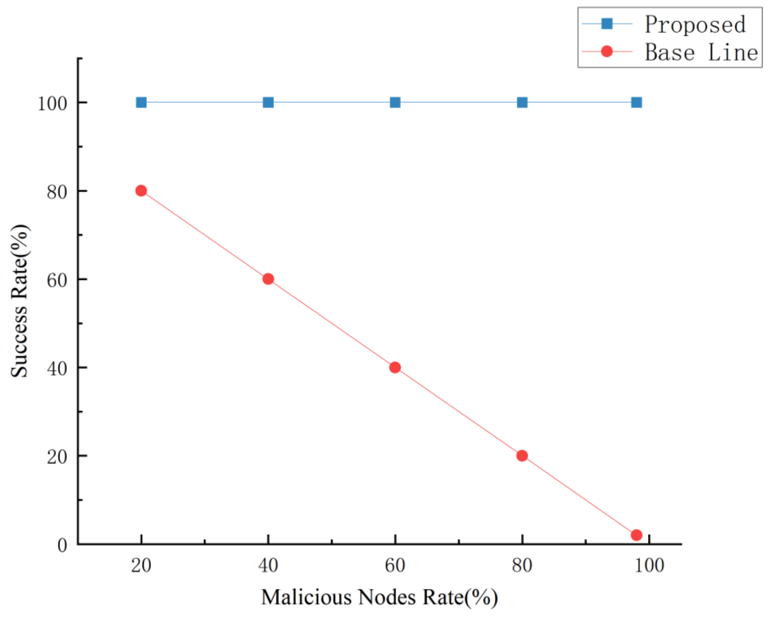

Figure 6. Effect of malicious nodes.

6. Conclusions and Further Work

In this paper, we examine the current mainstream cross-chain techniques and their problems. In order to solve the problem of the insufficient security and privacy of the HTLC cross-chain technology, we propose ExchangeSC and integrate it into the MA-HTLC. ExchangeSC combines smart contracts with Paillier homomorphic encryption for locking the secure and private exchange of information and introduces a reward and punishment mechanism to design an evidence storage function that follows in case of disputes between the parties. Then, a concrete implementation is demonstrated. The experimental results show that ExchangeSC has a reasonable gas cost. Compared to the original protocol, the newly proposed scheme improves the security of cross-chain transactions at the cost of a small time overhead and outperforms the established BitXHub project in terms of efficiency. Moreover, the scheme has high fault tolerance, and malicious nodes do not affect the success rate of information exchange.

Author Contributions: Funding acquisition, B.M. and Y.X.; Methodology, Y.W., B.M. and Y.X.; Project administration, Y.W. and B.M.; Software, Z.C. and R.M.; Writing—original draft, Z.C.; Writing—review and editing, Y.W., R.M., B.M., Y.X. and Q.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Key R&D Program of China (2021YFC3340600, 2021YFC3340602), the National Natural Science Foundation of China (62272255, 62302248, 62302249), the Shandong Provincial Natural Science Foundation (ZR2022LZH011, ZR2023QF018), the Shandong Provincial Higher Education Institutions Youth Creative Technology Supported Foundation (2022KJ124, 2023KJ131), the Taishan Scholar Program of Shandong (tsqn202306251), the QLUT Integration Pilot Project of Science Education Industry (2023PX006), the QLUT First Talent Research Project (2023RCKY143), the Ability Improvement Project of Science and Technology SMES in Shandong Province (2022TSGC2485, 2023TSGC0217), and the Jinan "New 20 Universities" Project of Introducing Innovation Team (202228016).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ma, B.; Tao, Z.Q.; Ma, R.H.; Wang, C.P.; Li, J.; Li, X.L. A High-Performance Robust Reversible Data Hiding Algorithm Based on Polar Harmonic Fourier Moments. *IEEE Trans. Circuits Syst. Video Technol.* **2024**, *34*, 2763–2774. [CrossRef]
2. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Bus. Rev.* **2008**.
3. Leng, J.; Zhou, M.; Zhao, J.L.; Huang, Y.; Bian, Y. Blockchain security: A survey of techniques and research directions. *IEEE Trans. Serv. Comput.* **2020**, *15*, 2490–2510. [CrossRef]
4. Tian, Y.; Li, T.; Xiong, J.; Bhuiyan, M.Z.A.; Ma, J.; Peng, C. A blockchain-based machine learning framework for edge services in IIoT. *IEEE Trans. Ind. Inform.* **2021**, *18*, 1918–1929. [CrossRef]
5. Cortes-Goicoechea, M.; Franceschini, L.; Bautista-Gomez, L. Resource analysis of Ethereum 2.0 clients. In Proceedings of the 2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), Paris, France, 27–30 September 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–8.
6. Aggarwal, S.; Kumar, N. Hyperledger. In *Advances in Computers*; Elsevier: Amsterdam, The Netherlands, 2021; Volume 121, pp. 323–343.
7. Ou, W.; Huang, S.; Zheng, J.; Zhang, Q.; Zeng, G.; Han, W. An overview on cross-chain: Mechanism, platforms, challenges and advances. *Comput. Netw.* **2022**, *218*, 109378. [CrossRef]
8. Neisse, R.; Hernández-Ramos, J.L.; Matheu-Garcia, S.N.; Baldini, G.; Skarmeta, A.; Siris, V.; Lagutin, D.; Nikander, P. An interledger blockchain platform for cross-border management of cybersecurity information. *IEEE Internet Comput.* **2020**, *24*, 19–29. [CrossRef]
9. Scheid, E.J.; Kiechl, P.; Franco, M.; Rodrigues, B.; Killer, C.; Stiller, B. Security and standardization of a notary-based blockchain interoperability API. In Proceedings of the 2021 Third International Conference on Blockchain Computing and Applications (BCCA), Tartu, Estonia, 5–16 November 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 42–48.
10. Xiong, A.; Liu, G.; Zhu, Q.; Jing, A.; Loke, S.W. A notary group-based cross-chain mechanism. *Digit. Commun. Netw.* **2022**, *8*, 1059–1067. [CrossRef]
11. Siris, V.A.; Nikander, P.; Voulgaris, S.; Fotiou, N.; Lagutin, D.; Polyzos, G.C. Interledger approaches. *IEEE Access* **2019**, *7*, 89948–89966. [CrossRef]

12. Westerkamp, M.; Eberhardt, J. zkrelay: Facilitating sidechains using zksnark-based chain-relays. In Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Genoa, Italy, 7–11 September 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 378–386.
13. Saleh, F. Blockchain without waste: Proof-of-stake. *Rev. Financ. Stud.* **2021**, *34*, 1156–1190. [CrossRef]
14. Sliwinski, J.; Wattenhofer, R. Better incentives for proof-of-work. In Proceedings of the International Symposium on Stabilizing, Safety, and Security of Distributed Systems, Clermond Ferrand, France, 15–17 November 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 314–328.
15. Yin, L.; Xu, J.; Tang, Q. Sidechains with fast cross-chain transfers. *IEEE Trans. Dependable Secur. Comput.* **2021**, *19*, 3925–3940. [CrossRef]
16. Herlihy, M. Atomic cross-chain swaps. In Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, Washington, DC, USA, 25–27 July 2018; pp. 245–254.
17. Cai, J.; Zhou, Y.; Hu, T.; Li, B. PTLC: Protect the Identity Privacy during Cross-Chain Asset Transaction More Effectively. In Proceedings of the 2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C), Guangzhou, China, 5–9 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 70–78.
18. Monika; Bhatia, R.; Jain, A.; Singh, B. Hash time locked contract based asset exchange solution for probabilistic public blockchains. *Clust. Comput.* **2022**, *25*, 4189–4201. [CrossRef]
19. Barbàra, F.; Schifanella, C. MP-HTLC: Enabling blockchain interoperability through a multiparty implementation of the hash time-lock contract. *Concurr. Comput. Pract. Exp.* **2023**, *35*, e7656. [CrossRef]
20. Liu, F.; Zhang, J.; Zhou, J.; Li, M.; Kong, D.; Yang, J.; Qi, J.; Zhou, A. Novel hash-time-lock-contract based cross-chain token swap mechanism of blockchain. *China J. Comput. Sci* **2022**, *49*, 336–344.
21. Jhanwar, M.P.; Venkateswarlu, A.; Safavi-Naini, R. Paillier-based publicly verifiable (non-interactive) secret sharing. *Des. Codes Cryptogr.* **2014**, *73*, 529–546. [CrossRef]
22. Guerrero, J.; Chapman, A.C.; Verbič, G. Decentralized P2P energy trading under network constraints in a low-voltage network. *IEEE Trans. Smart Grid* **2018**, *10*, 5163–5173. [CrossRef]
23. Zhao, Z.; Zhou, L.; Su, C. Systematic Research on Technology and Challenges of Lightning Network. In Proceedings of the 2021 IEEE Conference on Dependable and Secure Computing (DSC), Fukushima, Japan, 30 January–2 February 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–8.
24. Remix-Project. Available online: https://github.com/ethereum/remix-project (accessed on 2 July 2024).
25. The Solidity Contract Oriented Programming Language. Available online: https://github.com/ethereum/solidity (accessed on 1 June 2023).
26. Sepolia: Ethereum Official Testnet. Available online: https://sepolia.dev (accessed on 1 June 2023).