mathematics

Article

The Data Privacy Protection Method for Hyperledger Fabric Based on Trustzone

Wen Gao [1], Xinhong Hei [1,2] and Yichuan Wang [1,2,*]

[1] School of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048, China; 2181220027@stu.xaut.edu.cn (W.G.); heixinhong@xaut.edu.cn (X.H.)
[2] Shaanxi Key Laboratory for Network Computing and Security Technology, Xi'an 710048, China
[*] Correspondence: chuan@xaut.edu.cn

Abstract: Hyperledger Fabric is a distributed ledger solution platform based on a modular architecture. The cryptographic algorithm is the core of the platform to ensure the security and tamper-resistant of the data on the chain. However, the original Fabric platform lacks the protection of user's keys and cryptographic operations. To this end, this paper proposes a data privacy protection method for Hyperledger Fabric based on Trustzone technology, which places the user's key and the cryptographic operation process of private data in the trusted execution environment for protection. The experimental results based on the existing blockchain network show that the scheme can effectively ensure the security of data encryption process and key static storage, greatly reduce the trusted computing base and the attack surface. The performance loss is within an acceptable range.

Keywords: Hyperledger fabric; Trustzone; privacy

MSC: 68P27

check for updates

Citation: Gao, W.; Hei, X.; Wang, Y. The Data Privacy Protection Method for Hyperledger Fabric Based on Trustzone. *Mathematics* **2023**, *11*, 1357. https://doi.org/10.3390/math11061357

Academic Editors: Ding Wang, Qi Jiang, Chunhua Su and Antanas Cenys

Received: 27 January 2023
Revised: 20 February 2023
Accepted: 8 March 2023
Published: 10 March 2023

1. Introduction

In recent years, blockchain technology has developed rapidly around the world, and its application has extended to the field of Internet of Things (IoT). Moreover, leading IoT companies have begun to deploy blockchain technology. IBM, Microsoft, Amazon, and SAP all provide blockchain services on their respective cloud platforms to deliver elastic resource pools for future massive IoT device access. As a distributed and trustless technology [1], blockchain can provide usable, transparent, traceable, and non-tamperable data-storage protection measures for the IoT; however, this also brings with it issues regarding data privacy disclosure. To this end, Hyperledger Fabric, an enterprise-level commercial blockchain project alliance chain platform, designed a BCCSP (Blockchain Cryptographic Service Provider)module to provide encryption algorithm-related services for the SDK of the client to protect user privacy [2,3]. That is, before data are uploaded to the chain, the client is encrypted by calling the chaincode to ensure that the hash or ciphertext of the original data is stored on the chain. This method provides high independent privacy, since only offline authorized users can view the original data. However, there are certain security risks, as shown in Figure 1.

As shown in Figure 1, for the current fabric blockchain node, the smart contract runs in a rich execution environment (REE) with a low-security level, and the contract logic may be maliciously tampered with, resulting in untrustworthy changes to the state data. In addition, data encryption on the REE side lacks root key protection, and malicious applications may monitor the private data-processing process, thereby threatening the security of private data.

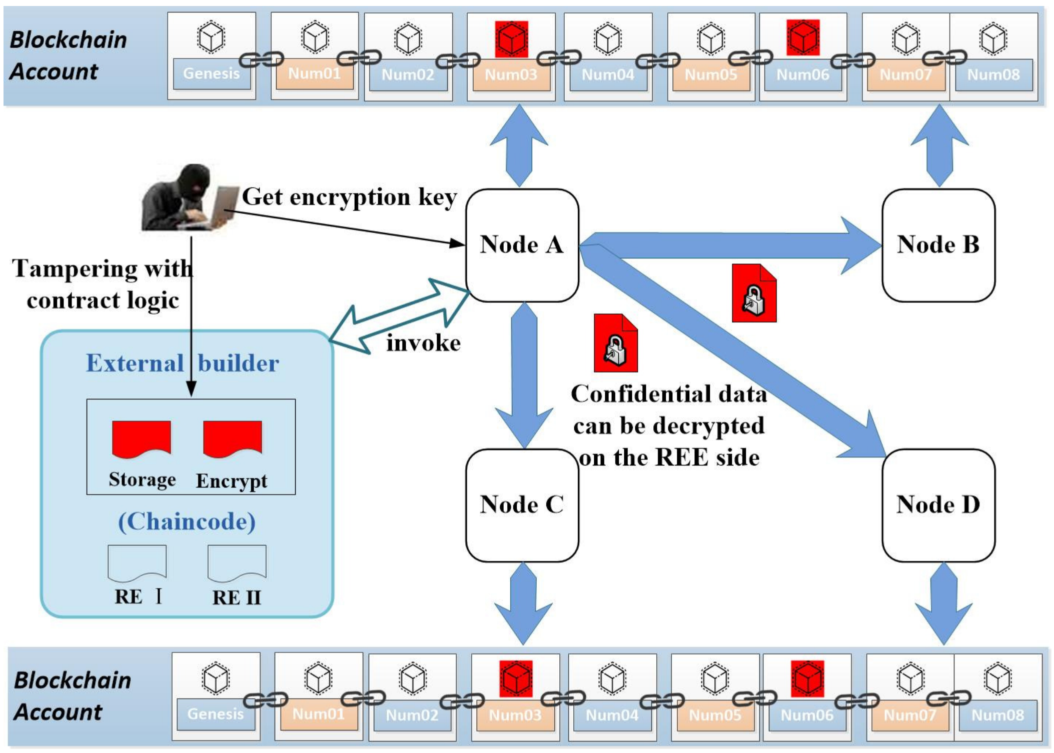


Figure 1. The security risks present in ordinary smart contract engines.

1.1. Related Works

Aiming at the privacy disclosure problems existing in Fabric, researchers have proposed some privacy protection schemes for Fabric in recent years. Brandenburger et al. [4,5] used Trusted Execution Environment (TEE) to protect the privacy of chaincode data computation from potentially untrusted peers square influence. Cheng et al. [6] proposed the Ekiden model, which adopts the mechanism of separating computation and consensus, and computing nodes complete the computation of private data in an offline trusted execution environment. Fan et al. [7] placed the sensitive private information-processing process in the blockchain transaction process in the Intel Software Guard Extensions (SGX) security area for protection. Results presented from the above research works can help protect the data privacy of Fabric to a certain extent, but these are only applicable to the server platform of the X86 architecture and not suitablefor IoT devices.

Zhou [8] implemented a blockchain network for IoT systems called BeeKeeper 2.0. It uses homomorphic encryption technology to protect data. However, this method is less efficient. Dai et al. [9] designed a safe and readily available digital wallet based on Trustzone technology; however, the logic of the smart contract used in this method can easily to be tampered with. Müller et al. [10] designed a prototype for Hyperledger Fabric chaincode execution with Trustzone; however, it does not provide secure key storage.

1.2. Our Goal and Contribution

We summarized some relevant work in recent years, as shown in Table 1. We can see that most of the current research employs SGX technology to protect data privacy in Fabric and is more effective. However, this requires the node to have an Intel processor. When Fabric is applied to the IoT system, there is no perfect solution to ensure the privacy and security of data. Therefore, this paper proposes a data-privacy protection method of Fabric blockchain based on Trustzone technology for IoT devices. It uses the trusted execution environment to undertake the operation of chaincode, and places the user key and privacy data-processing process in the trusted execution environment for protection. This method can resist malicious-application monitoring in the process of chaincode encryption and improve the security of key static storage in Fabric. In addition, this method safely conceals the cryptographic details of Fabric from users, greatly reducing the trusted computing base and attack surface, thus reducing the risk of private data disclosure.

Table 1. Summary of some related works.

Scheme	Research Work	Resistance to Attacks
SGX	Brandenburger et al. [4,5]	Privacy stealing and rollback attack
	Cheng et al. [6]	Privacy stealing and key disclosure attack
	Fan et al. [7]	Privacy stealing and replay attack
	Yannick et al. [11]	Privacy stealing, key disclosure, image replacement and MITM attack
Trustzone	Dai et al. [9]	Key disclosure attack
	Müller et al. [10]	Privacy stealing attack
	This work	Privacy stealing, key disclosure, image replacement attack

In summary, our main contributions are as follows:

1. Aiming at solving data confidentiality issues when applying Fabric blockchain on IoT devices, we propose a data-privacy protection method of Fabric blockchain based on Trustzone technology, using TEE to provide Fabric with encryption algorithm services to protect data privacy.
2. According to the TEE standard specification, we store the key in the trusted execution environment by designing the directory file and key encryption file, which improves security of the key stored in the Fabric blockchain node.
3. In order to reduce the impact of the trusted execution environment on the performance of data-encryption operations, we improve the data-copying stage, changing from data transfer to pointer transfer to optimize program performance.

The rest of the paper is organized as follows: Section 2 introduces the relevant knowledge involved in this paper. Section 3 presents our overall design. In Section 4, a detailed description of the scheme is provided. Section 5 presents an experimental configuration and analysis. Lastly, we provide concluding remarks.

2. Background

2.1. Hyperledger Fabric

Hyperledger Fabric is a consortium blockchain. Obtaining identity authentication and organizational permission is a prerequisite for accessing the Fabric network. After joining the network, members jointly maintain a distributed ledger. All data operations are public, and will be permanently recorded and cannot be tampered with. In Hyperledger Fabric, the communication security of each network node is guaranteed by the Public Key Infrastructure (PKI), which signs and authenticates user identities and communication messages [12–15].

2.2. Security and Cryptographic Services for Fabric Networks

The technical architecture of the Fabric network is shown in Figure 2. The blockchain cryptographic service provider (BCCSP) module is most closely related to the research work of this paper. Based on BCCSP, the common security and cryptographic services of the Fabric network can be implemented, providing secure and pluggable membership service management (MSP), consensus services and chaincode services for upper-layer applications [16,17].

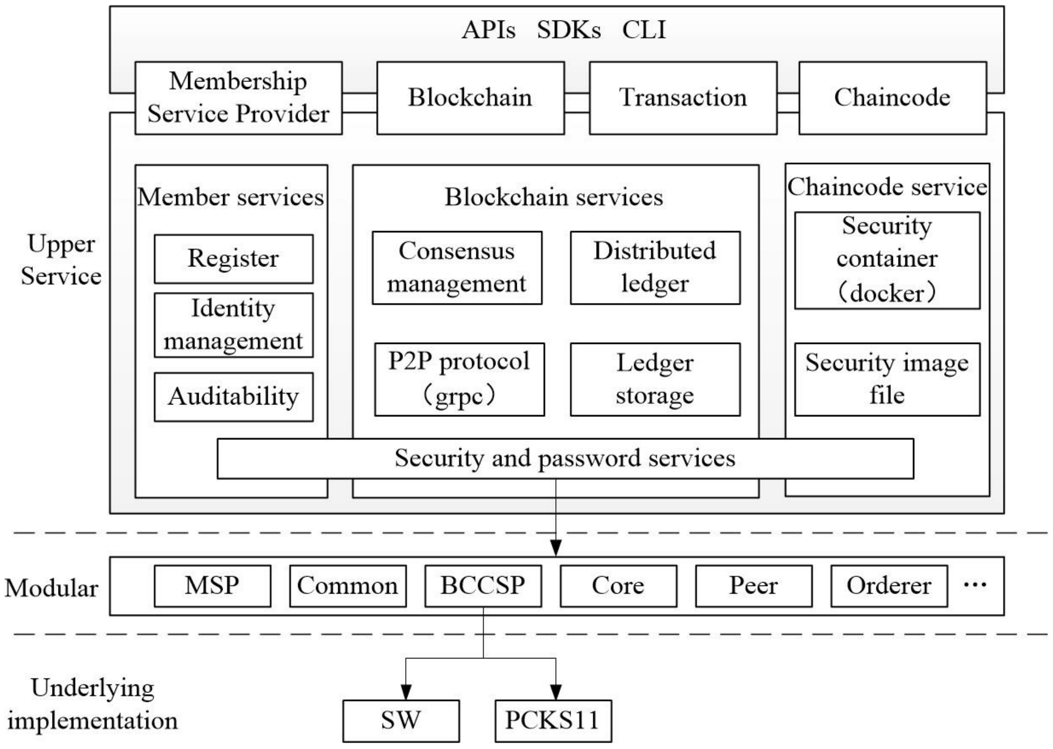


Figure 2. The technological architecture of the Fabric alliance chain network.

The BCCSP module encapsulates two types of BCCSP instances, SW based on software implementation and PCAS11 based on hardware implementation. Each type of instance provides four types of functional interfaces including key management, hashing, signature verification and encryption and decryption for upper-layer service calls. The cryptographic algorithms supported by the BCCSP module mainly include hash cryptographic algorithms, symmetric cryptographic algorithms and asymmetric cryptographic algorithms. The instance selected by BCCSP can be specified through the CSP option to provide corresponding support for upper-layer applications.

2.3. Smart Contract

A smart contract [18,19] is a set of digital commitments that can be triggered for automatic and trusted execution. By deploying it on the blockchain, the smart contract code can be triggered by authorized members, and the credibility of the execution result will be determined by each nodes to guarantee [20].

In Hyperledger Fabric, smart contracts are referred to as a chaincode, which supports Go, Node.js or JAVA and other multilingual programming. Chaincode status includes system chaincode and user chaincode. The system chaincode runs in the peer node and is assigned to perform specific system tasks, such as processing functions related to the life cycle of smart contracts, processing the configuration of user node side channels, and providing account book query APIs such as obtaining blocks and transactions. User chaincode is a user-defined chain code, which is executed on the docker container side. According to the business needs of users performing different tasks, the execution logic is encapsulated into the contract code.

2.4. Trustzone Technology

In order to implement a trusted execution environment in mobile devices, ARM introduced Trustzone technology in the ARM v6 architecture, which can provide protection and isolation of hardware resources at the chip level. The ARM core that supports the Trustzone technology divides the working state into two states: safe state (corresponding to trusted execution environment) and non-safe state (corresponding to rich execution environment) [21,22].Among them, the application running in the trusted execution environment is called TA (Trusted Application), and the application running in the rich execution environment is called CA (Client Application). The system switches between the safe and non-safe state of the ARM core by calling the SMC (Safety Monitor Mode Call) instruction. When the processor core is in a safe state, the processor can not only run the code on the TEE side, but also has permission to access the resources on the REE side. When it is in a non-safe state, it can only run the code on the REE side, and can only obtain those specific data and call specific functions on the TEE side through the previously defined interface. A block diagram of the system software level of Trustzone technology used in ARMv7 architecture is shown in Figure 3.

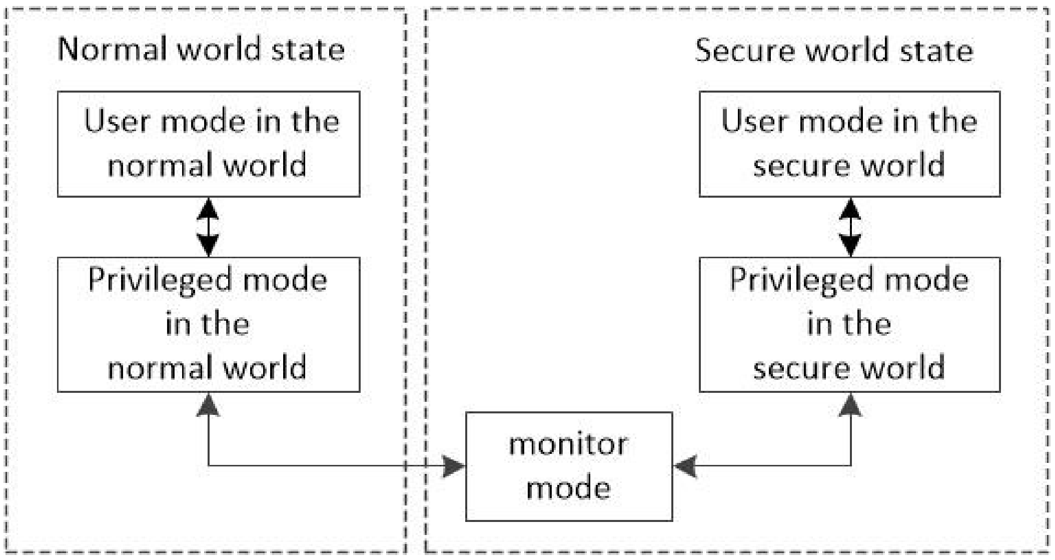

Normal world state
User mode in the normal world
Privileged mode in the normal world
Secure world state
User mode in the secure world
Privileged mode in the secure world
monitor mode

Figure 3. System software framework for Trustzone technology in ARMv7.

As shown in Figure 3, in addition to the user mode and privileged mode that typically exist in the two worlds, the secure world can also run in a third mode called monitor mode. When the user mode of the normal world wants to obtain services from the secure world, it needs to enter the privileged mode of the normal world first, and then invoke the security monitor call instruction. The processor enters the monitor mode, stores the context of the normal world in the monitor mode, then loads the privileged mode of the secure world, and finally loads the user mode of the secure world and provide corresponding services.

In this paper, OP-TEE [23] is selected as TEE. OP-TEE is a trusted execution environment based on Trustzone technology, and is developed according to GP specifications [24].

3. System Architecture

The system architecture proposed in this paper is shown in Figure 4. It includes an order, a peer, and a client. Chaincode_proxy runs as a chaincode on the same node as the peer node. In addition, in our structure, there is also a node for performing cryptographic operations in the secure world of OP-TEE. This node needs to have an ARM processor and support OP-TEE. CA runs in the normal world of OP-TEE and is responsible for processing and forwarding messages from chaincode_proxy and TA. A detailed description of the components we added is as follows.

- Chaincode_proxy. The chaincode_proxy is installed and instantiated as a chaincode, which is deployed on Fabric network nodes, runs in an isolated sandbox (currently a Docker container), and interacts with corresponding peer nodes through the gRPC protocol to manipulate data in the distributed ledger. it is responsible for forwarding chaincode encryption request to the CA or receiving processing responses from CA. After installation and instantiation, the UUID corresponding to the TA must be passed to the chaincode_proxy so that it can communicate with the TA that provides cryptographic services.

- CA. The CA for cryptographic operations runs in the normal world REE and is used to forward messages between chaincode_proxy and TA. Communication between the chaincode_proxy and the CA is achieved through gRPC remote procedure calls and the session between the CA and the TA is achieved through the TEE Client API function.
- TA. The TA for cryptographic operations runs in the TEE secure world and is called by CA. It replaces the BCCSP module in Fabric to provide encryption services for Fabric, including key generation, key storage, symmetric encryption, hash calculation, digital signature and other modules.

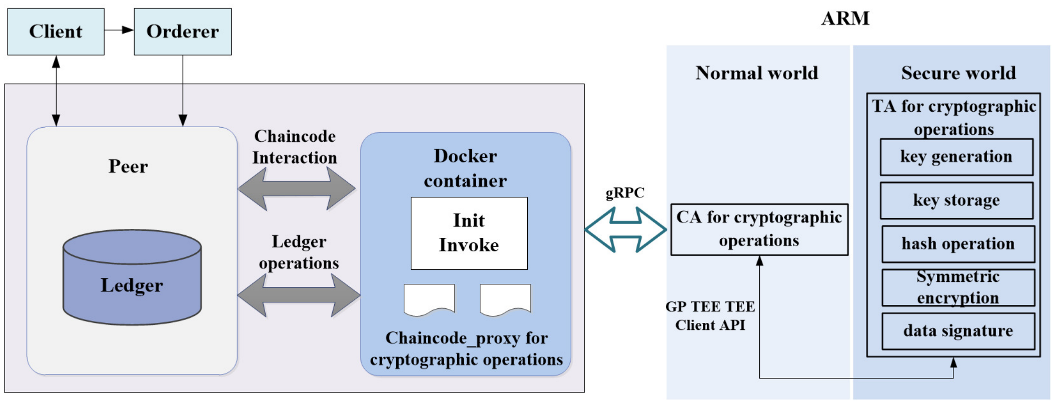


Figure 4. Proposed architecture in this paper.

4. Implementation Details

4.1. Interactive Process

The interaction process between the various parts of this paper is shown in Figure 5. Among them, Figure 5a shows the process of invoking encryption transaction to encrypt ledger data, and Figure 5b presents the process of invoking query transaction to decrypt ledger data.

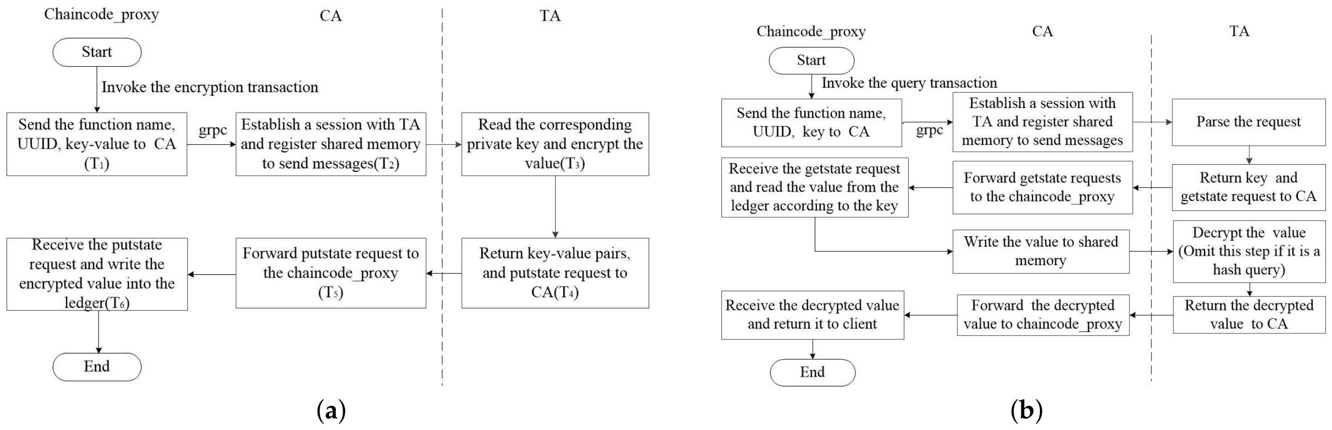

Chaincode_proxy
CA
TA
Start
Invoke the encryption transaction
Send the function name, UUID, key-value to CA (T1)
grpc
Establish a session with TA and register shared memory to send messages(T2)
Read the corresponding private key and encrypt the value(T3)
Return key-value pairs, and putstate request to CA(T4)
Forward putstate request to the chaincode_proxy (T5)
Receive the putstate request and write the encrypted value into the ledger(T6)
End
(a)
Chaincode_proxy
CA
TA
Start
Invoke the query transaction
Send the function name, UUID, key to CA
grpc
Establish a session with TA and register shared memory to send messages
Parse the request
Return key and getstate request to CA
Forward getstate requests to the chaincode_proxy
Receive the getstate request and read the value from the ledger according to the key
Write the value to shared memory
Decrypt the value (Omit this step if it is a hash query)
Return the decrypted value to CA
Forward the decrypted value to chaincode_proxy
Receive the decrypted value and return it to client
End
(b)

Figure 5. The interaction process between the various parts: (**a**) invoking encryption transaction; (**b**) invoking query transaction.

As shown in Figure 5a, the communication between the chaincode_proxy and the CA is achieved through gRPC remote procedure calls [25]. In order to enable both the chaincode_proxy and the CA to send data streams to each other, we adopt the bidirectional streaming RPC mode, the protocol used is the buffer language syntax proto3 [26]. The chaincode_proxy acts as the gRPC client, and the CA acts as the gRPC server, listening on port 50051. Whenever the client calls the chaincode_proxy, the chaincode_proxy creates a gRPC client data stream, and sends the function name, UUID , data (key–value) and related parameters to the CA, and then waits for the CA's message.

When the CA receives the message, the CA will call the TEEC_OpenSession function to open the session, establish a connection with the corresponding TA in the TEE according to the received UUID value and call the TEEC_RegisterSharedMemory function to register a shared memory, which is then used for the communication between the CA and the TA. Subsequently, the CA writes function name, data to be encrypted and parameters into shared memory, and calls the TEEC_InvokeCommand function to send it to the corresponding TA. TA initializes the session context when the initial function is called according to the received function name and the data to be operated, and then performs the specified operation on the data (see Sections 4.3 and 4.4 for details). When the TA execution is completed, the TA puts the key and the value after the operation into the shared memory, adds the putstate request to the parameter and returns it to the CA, indicating that the execution has been completed. The CA forwards the response to the chaincode_proxy, and then uses the TEEC_CloseSession and TEEC_FinalizeContext functions to close the session with the TA and release resources.The chaincode_proxy receives the putstate request and write the encrypted value into the ledger. This encryption process is complete.

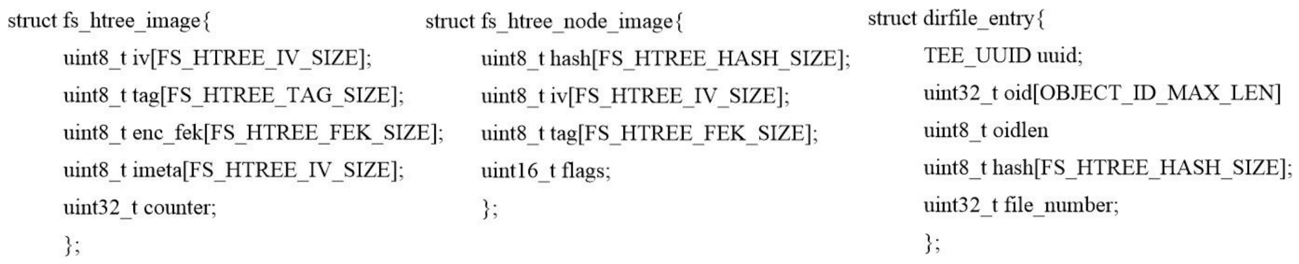
In addition, the biggest difference between the query transaction in Figure 5b and the encrypted transaction in Figure 5a is that it needs one GetState and no PutState.

4.2. Key Storage Module in TA

In native Fabric, the keystore.go file in the BCCSP directory defines the KeyStore interface, which provides key management and storage functions. We redesign these functions and implement them in a trusted environment to improve key security.

4.2.1. Creation of Directory Files and Key Encrypted Files

Before creating the file to store the user key, we need to create a directory file to manage the key encryption file first . The directory file is divided into three areas, which are used to save the file header, node, and data, respectively. The file header part stores the content of the fs_htree_image structure.The node part stores the content of the fs_htree_node_image structure. The data area stores the relevant information of all encrypted files, using dirfile_entry structure represents the basic structure of the encrypted file. The definition of the three part structure is shown in Figure 6.



```
struct fs_htree_image{
    uint8_t iv[FS_HTREE_IV_SIZE];
    uint8_t tag[FS_HTREE_TAG_SIZE];
    uint8_t enc_fek[FS_HTREE_FEK_SIZE];
    uint8_t imeta[FS_HTREE_IV_SIZE];
    uint32_t counter;
    };

struct fs_htree_node_image{
    uint8_t hash[FS_HTREE_HASH_SIZE];
    uint8_t iv[FS_HTREE_IV_SIZE];
    uint8_t tag[FS_HTREE_FEK_SIZE];
    uint16_t flags;
    };

struct dirfile_entry{
    TEE_UUID uuid;
    uint32_t oid[OBJECT_ID_MAX_LEN]
    uint8_t oidlen
    uint8_t hash[FS_HTREE_HASH_SIZE];
    uint32_t file_number;
    };
```

Figure 6. Structure definition.

After the directory file is created, we start to create the key encryption file. The format of the encrypted file is roughly the same as that of the directory file, except that the data block stores the user's key. The process of creating an encrypted file is shown in Figure 7.

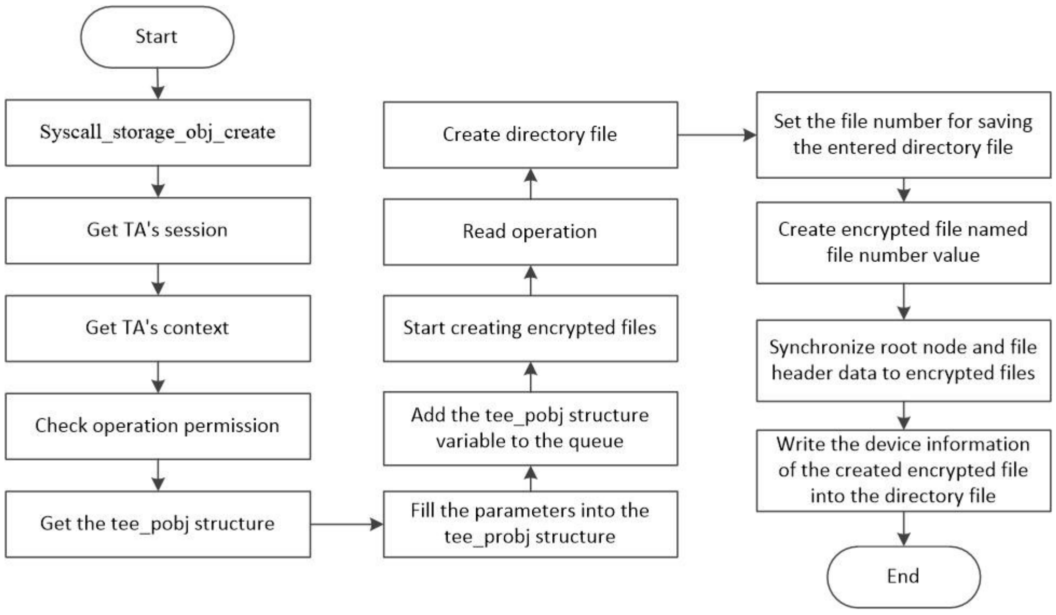

Start
Syscall_storage_obj_create
Get TA's session
Get TA's context
Check operation permission
Get the tee_pobj structure
Fill the parameters into the tee_probj structure
Add the tee_pobj structure variable to the queue
Start creating encrypted files
Read operation
Create directory file
Set the file number for saving the entered directory file
Create encrypted file named file number value
Synchronize root node and file header data to encrypted files
Write the device information of the created encrypted file into the directory file
End

Figure 7. Creation process of encrypted files.

When creating an encrypted file, the data area of the encrypted file will be initialized, and then the tee_fs_htree_node_image area of the encrypted file and the tee_fs_htree_image area saved in the file header will be updated. In order to enable the directory file to find the encrypted file later, it is also necessary to update the contents of the encrypted file information defined by the data area dirfile_entry in the directory file.To here, the creation process of the encrypted file ends.

4.2.2. Writing and Reading of Key Encryption File

Before writing and reading the key encrypted file, we need to open it first, so we need to find the storage number of the encrypted file by reading the dirfile_entry structure corresponding to the encrypted file in the data area of the directory file. For the write operation, we call the TEE_WriteObjectData function, which will obtain the TA's session ID and running context, check the permissions, and then complete the data writing through the ree_fs_write function. Similarly, for the read operation, we call the ree_fs_write function, which will call the ree_fs_read function to read the encrypted file. The ree_fs_read function calculates which data block the read data belongs to, then finds the node ID corresponding to the block, so as to obtain the IV value used in the encryption and decryption of the data block, and finally decrypts the read plaintext data through FEK and IV.

4.3. Symmetric Encryption and Signature Module in TA

In the native Fabric, BCCSP provides data encryption in AES-CBC mode, the specific types supported are AES128/AES192/AES256, and the padding method used is: pkcs7Padding, Therefore, we implement the above three encryption types in the trusted execution environment to complete chaincode encryption instead of BCCSP. The specific implementation steps are as follows:

1. When the TA receives the TEEC_InvokeCommand command request from the CA, it calls the TEE_AllocateOperation function to allocate an operation handle for cryptographic operations, and sets the algorithm type and mode to determine whether the cryptographic algorithm is encrypted or decrypted. Where the mode parameter is set to EN_MODE_CBC.
2. Call the TEE_AllocateTransientObject function to allocate an uninitialized temporary space. After the allocation operation is successful, call the TEE_InitRefAttribute and TEE_PopulateTransientObject functions to set the key object parameters, and then call

the TEE_SetOperationKey function to save the contents of the object storing the key to the operation handle.
3. Call the TEE_CipherInit function to initialize the configuration.
4. Call the TEE_CipherUpdate function interface to transmit the data to be calculated.
5. Call the TEE_CipherDoFinal function to complete the AES encryption and decryption operation, and return the operation result.

In addition, the signature algorithm used in the native Fabric network is Elliptic Curve Digital Signature Algorithm (ECDSA), which mainly supports ECDSAP384/ECDSAP256. Since optee's trusted execution environment provides two interfaces, TEE_ALG_ECDSA_P256 and TEE_ALG_ECDSA_P384, we can implement the two signature algorithms in the trusted environment through interface calls, and the steps are similar to the AES encryption method.

4.4. Data Encryption and Signature Module in TA

The hash algorithms used in the native Fabric are SHA-256, SHA-384, SHA3-256, and SHA3-384. Similarly, we implement the above algorithms in a trusted execution environment to replace BCCSP to complete the hash operation. The implementation steps are as follows:

1. When a link is established between the TA and the CA through the OpenSession operation, the TEE_AllocateOperation function is called to specify the type of digest algorithm (for example: SHA-256, SHA-384, etc.)
2. Call the TEE_DigestDoFinal function to send the data to be calculated (this function can be called continuously to send the data).
3. Call the TEE_DigestUpdate function to perform the digest calculation operation, and feed the data back to the TA.
4. Finally, the functions of CloseSession and FinalizeContext are called in turn to release resources and avoid resource occupation caused by data.

5. Experimental Configuration and Analysis

5.1. Experimental Environment

The blockchain server configuration employed in this paper: Intel Core i5-8400 CPU @ 2.80 GHz 2.81 GHz. This paper uses qemu-v8 to simulate IoT devices to run OP-TEE system. We set the core number -smp of qemu-v8 to 4, the trusted thread CFG_NUM_THREADS to 4, and the NUM_CQS (the number of queues at the gRPC server which will listen to incoming RPCs) to 1. The detailed configuration of the experiment is shown in Table 2.

Table 2. Experimental environment configuration.

Environment	Version	Notes
operating system	Ubuntu 18.04 server x64	-
blockchain network	Hyperledger Fabric1.4.3	Open source blockchain architecture
gRPC	v.1.20.0	For remote procedure calls
OP-TEE	v3.8.0	Provide the trusted environment
Go language	Go 1.14.6	Smart contract development language

5.2. TCB Size

The Trusted Computing Base (TCB) [27] is a collection of all implementation strategies and mechanisms to achieve data security. The smaller the TCB, the fewer corresponding errors and the more secure the system. In order to calculate the TCB of our scheme, we use the code statistics tool–CLOC, and the results are shown in Table 3.

Table 3. Lines of code for each component.

	Go	C	C++	Protocol Buffer	Total
chaincode_proxy	191	0	0	40	231
CA	0	0	388	40	428
TA	0	896	0	0	896

According to Table 3, since the CA, the chaincode_proxy is not trusted, and the TA in the TEE is trusted. Therefore, the TCB of our scheme consists of TA, i.e., 896 lines of trusted code. For native Fabric, the number of lines of code related to BCCSP module is about 5600 lines. By providing cryptographic algorithm services for Hyperledger Fabric only in the trusted environment, instead of completing cryptographic operations through the BCCSP interface in the whole peer, the trusted computing base and attack surface are greatly reduced.

5.3. Security Analysis

5.3.1. Analysis of Potential Attack Scenarios

This section mainly analyzes the current attacks and privacy disclosure risks suffered by Fabric, and introduces the defense of our scheme against these attacks.

- Key disclosure attack

Currently, the user's private key of Fabric is encrypted and stored in the Keystore. If the attacker attempts to delete this file directly after obtaining the root permission of the terminal device, the user will not be able to encrypt and decrypt the private data normally. In this scheme, the access and operation requests for the directory file and key encrypted file in TEE will be subject to the strict security verification of the security extension component, and the encrypted files in TEE are independent of each other.The key required for the encrypted file is only randomly generated in the trusted memory during the operation of the code in the Trustzone and will not be disclosed to the outside. Therefore, even an attacker with root privileges cannot destroy the encrypted file.

- Privacy Stealing Attack

Privacy stealing attack means that in the scenario where the attacker invades the Fabric container, the chaincode running in the original Fabric is not protected. Since the plaintext of data and keys will be involved in the operation process, malicious applications can steal user data and key information by monitoring encryption processes. In this scheme, the operation process of providing cryptographic services for Fabric is completed inside TA. Through the memory isolation, interrupt isolation and other mechanisms of Trustzone, it is ensured that the program in REE cannot access the instructions and data of TEE. Therefore, the malicious application on the REE side cannot perform malicious monitoring and other operations on the cryptographic operation process, which can effectively ensure the security of the execution process.

- Image Replacement Attack

When the Fabric container is created, it relies on the image tag to create an operating environment for the chaincode. Image replacement attack means that since the image is stored locally, an attacker can use a malicious image with a backdoor to replace the original image and use the label of the original Fabric. In this scheme, the startup process of the IoT device is secure booted by the ATF [9]. The ATF will verify the kernel image before starting the image of the next stage to ensure that the boot kernel image has not been illegally tampered with, and then OPTEE_OS verify the TA when loading the trusted application into the secure memory to ensure that the TA is legal and not the replaced application of the same name, thus ensuring the logical security of the TA.

5.3.2. Compared with E-Fabric

We compare our scheme with E-Fabric [11]. E-Fabric is a method to build a trusted execution environment for lightweight fabric chaincode based on SGX. It is improved on the basis of Ref. [5] and provides better privacy protection capability for Fabric. The comparison results are shown in Table 4.

Table 4. Comparison of E-Fabric and our scheme.

Feature	E-Fabric	Our Scheme
Technology	Intel SGX	ARM Trustzone
Resist root attacker	√	√
Chaincode confidentiality	√	√
Remote attestation	√	×
Data encryption	√	√
Key safety storage	×	√

Both our scheme and E-Fabric can ensure the security of the chaincode encryption process. E-Fabric is applicable to nodes with Intel processors, while our scheme is applicable to nodes with ARM processors. Our scheme provides key management and storage functions for Fabric by designing directory file and key encryption file, while E-Fabric does not provide secure storage of keys.

In E-Fabric, peer nodes can remotely authenticate and exchange keys with the chaincode container to confirm that the transaction proposal is executed by the authorized chaincode enclave. However, since Trustzone itself does not provide remote authentication support, our scheme loses the verification of chaincode execution by an authorized TEE.

5.4. Performance Evaluation

5.4.1. Performance of Hash Operation Module

First, we tested the time it took for the client to invoke the chaincode_proxy to perform a hash operation (That is, the time used for the whole process in Figure 5a). The results are shown in Figure 8. Among them, Figure 8a shows the latency of a single client invoking the chaincode_proxy to perform sha256 and sha384 operations on different data sizes. Figure 8b shows the average latency of different numbers of clients repeatedly invoking the chaincode_proxy in parallel to perform the SHA256 operation on 64 KB data within 30 s.

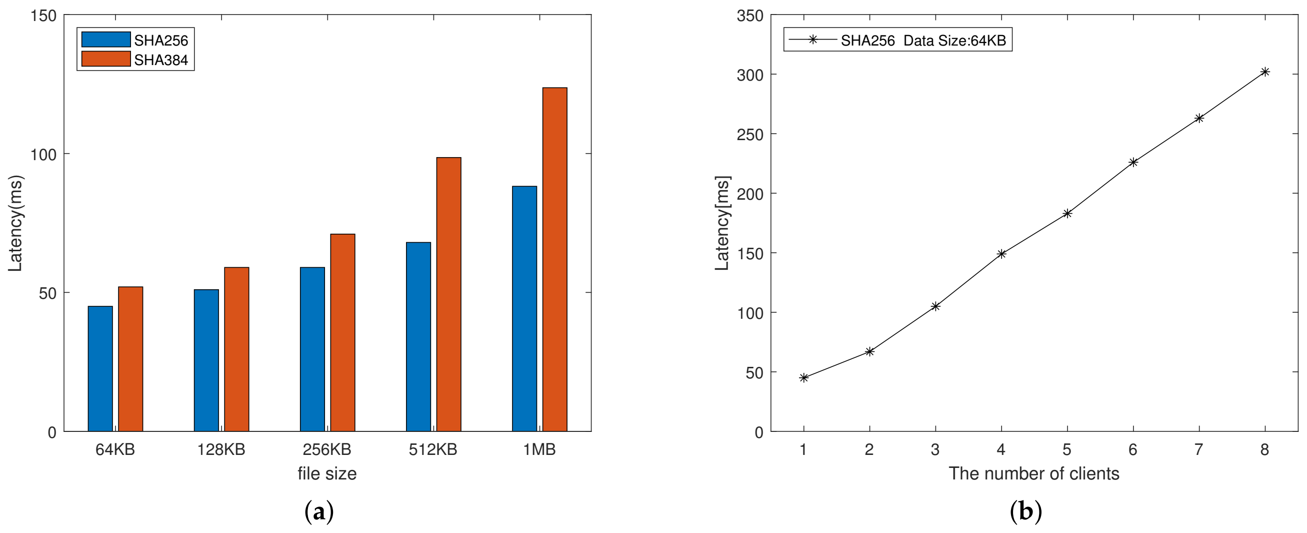


Figure 8. Latency for the execution of the chaincode hash transaction invoked by client: (**a**) one single client; (**b**) multiple clients.

From Figure 8a, it can be seen that for small data lengths (less than 256 KB), the overall time spent performing SHA256 and SHA384 operations is not significantly different. With the increase in data length, the time cost of SHA384 increases significantly faster than that of SHA256. Furthermore, we observe from Figure 8b that the average latency of all single transactions increases with an increasing number of clients.

In order to understand the duration of each subsection in the above process, we take the process of the client calling the chaincode_proxy to perform sha256 hash calculation on 64 KB data as an example. We measured the time required for each part (as shown in Figure 5a), where T_1 represents the initialization time of chaincode_proxy and the time to send data, function name and other parameters to the CA. T_2 includes the initialization of the CA, the time to establish a connection with the corresponding TA in the TEE and copy the data to the shared memory through the TEEC_InvokeCommand function, T_3 is the time for the TA to perform cryptographic operations, T_4 is the time when TA returns key–value pairs and putstate request to CA. T_5 is the time when the CA forwards putstate request to the chaincode_proxy. T_6 is the time when the chaincode_proxy writes data to the ledger. The running time of each part is shown in Table 5.

Table 5. The running time of each part.

	Average (ms)	Min (ms)	Max (ms)
T_1	9.46	7.37	13.24
$T_2 + T_3$	30.36	26.13	34.24
T_4	1.61	1.52	1.68
T_5	0.02	0.02	0.02
T_6	0.47	0.41	0.61
sum	41.92	—	—

From Table 5, it can be seen that the average time of the whole process is 41.92 ms, of which $T_2 + T_3$ accounts for the largest proportion of the whole process. That is, the main time-consuming part of the whole process is during the stage when CA calls TA to complete the hash operation.

In order to reduce the impact of T_2 on the whole process, we study the impact of data copying in shared memory in T_2 on program performance. In T_2, CA reads the original data into the CA data area first. When calling and executing TA, CA copies the required data into the shared memory, and then copies the shared memory data into the TA data area. In the whole process, multiple data memory copies are faced, which reduces program performance, and the larger the amount of copied data, the more program overhead, and the lower the program execution efficiency.

To this end, we change data passing to pointer passing to optimize program performance. Specifically, after the CA registers and allocates the shared memory, the original data is directly read into the shared memory, and only the pointer of the data block in the shared memory is passed when calling the execution TA to pass the parameters. When performing hash operations in TEE, data is directly read from shared memory to decrease the times of memory data copy.

As can be seen from Figure 9, when allocating different shared memory sizes, hash operations are performed on 64 KB, which reduces the number of memory copies. As the allocated shared memory gradually increases, the efficiency improvement can be maintained at about 8%.

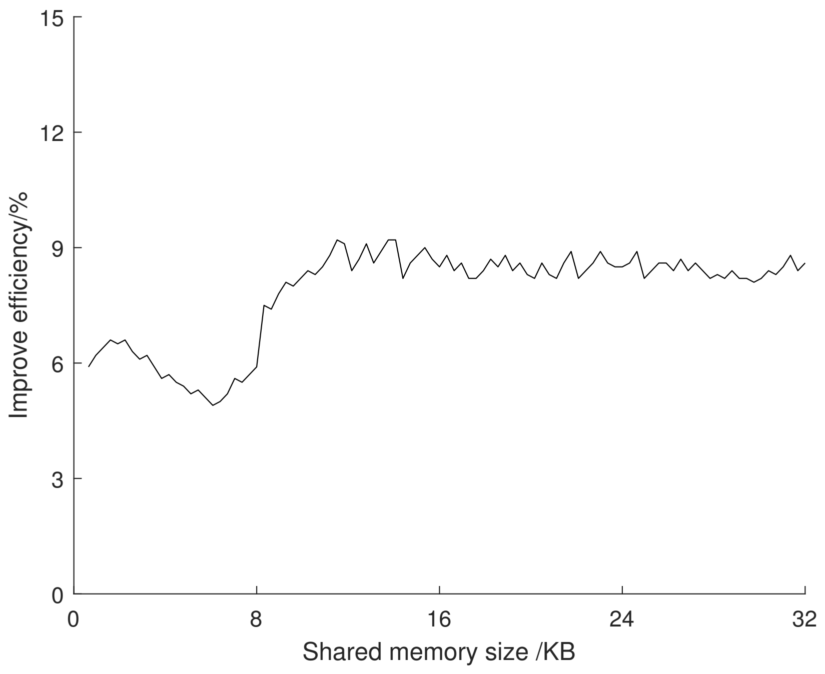


Figure 9. Performance improvement brought by reducing memory copies under different shared memory sizes.

After writing the hash calculation results into the ledger, we measured the throughput of query transaction in our method, the control group and the native fabric. The measurement is completed by repeatedly querying the results on the chain in parallel for different number of clients (1–8) over a period of time. In order to verify the impact of TEE on system throughput, we set the control group. The control group implements the logic of TA in this scheme by CA, that is, it does not interact with TEE. The results are shown in Figure 10.

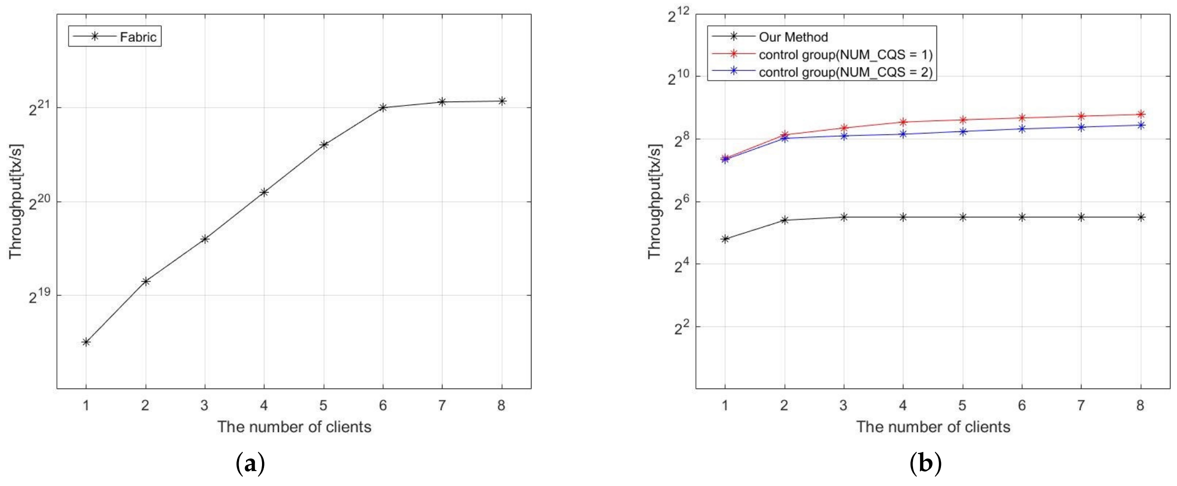


Figure 10. Throughput of parallel calls to query transactions by different number of clients: (**a**) measurements are done for the plain Hyperledger Fabric; (**b**) measurements are done for our method and control group.

We found that for our method, the throughput increased slightly from 1 client to 2 clients (factor < 1.2). When there are more than two clients, the throughput is stagnant. For native fabrics, the throughput will increase significantly as the number of clients increases, until the number of clients equals 6 (cores). Therefore, the Go program does not affect the throughput of our scheme. In addition, for the control group, when NUM_CQS is set to 1 or 2, the throughput increases 1.6 times and 1.72 times from 1 client to 2 clients, respectively.Compared with 1 client, 8 clients increased 2.14 times and 2.64 times, respectively, Therefore, gRPC is not the main reason that affects the throughput of our scheme. From this, we can conclude that the execution of chaincode in the TEE is mainly responsible for the throughput of our scheme.

5.4.2. Read and Write Performance of Key Encrypted Files

We tested the read and write performance of the key-encrypted files in this paper, as shown in the following Table 6.

Table 6. Read and write performance of key encrypted files.

Key Size	Writing (ms)	Reading (ms)
128 B	58.27	36.61
192 B	61.33	38.07
256 B	65.15	39.36

From the Table 6, it can be seen that the reading and writing of secure storage does incur some time overheads. However, because the objects stored are relatively small, the key size to be saved is 128/192/256 bytes, so the time performance loss is deemed acceptable. Furthermore, the write operation will be carried out only when the user creates it. Only read operations are performed during transaction execution.

5.4.3. Performance of Symmetric Encryption Module

We perform AES128-CBC encryption and decryption on plaintext data in the TA of this paper, in the enclave of SGX and in the native Fabric, respectively. We measure its running time, and convert it into throughput rate to observe the changes in computing performance overhead. Among them,the AES encryption and decryption functions in the SGX SSL extension library (based on OpenSSL) are used in the enclave of SGX. The results are shown in the following Figure 11.

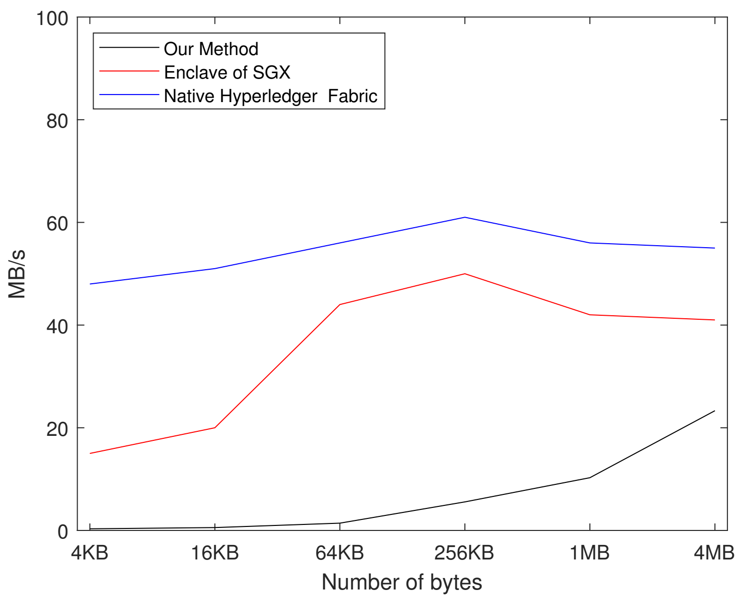


Figure 11. Comparison of AES encryption efficiency.

According to the results of the experiment, when the length of plaintext is short, the performance loss of this paper will be more obvious. This is because when the length of plaintext is short, the additional overheads such as allocating shared memory and world switching are too obvious compared with the encryption and decryption overheads. However, with the increasing number of bytes, the time-consumption of encryption and decryption operations gradually becomes dominant, and the performance gradually improves.

In addition, the trusted execution environment provided by Intel SGX technology is suitable for the server platform of X86 architecture, and the method proposed in this paper is aimed at IoT devices of ARM architecture, so its performance is weaker than SGX. However, the computing time is still within the millisecond range, indicating that it is possible to protect blockchain data privacy through IoT devices.

6. Conclusions

This paper proposes a data-privacy protection method of Hyperledger Fabric based on Trustzone for IoT devices. The method replaces the BCCSP module in the native Fabric to complete the chaincode encryption by designing and implementing modules such as key storage, hash operation, and symmetric encryption in TEE. The experimental results show that this method can resist malicious application monitoring in the process of chaincode encryption and improve the security of key static storage in Fabric. In addition, this method safely conceals the cryptographic details of Fabric from users, greatly reducing the trusted computing base and attack surface, thus reducing the risk of privacy data disclosure. Compared with the chaincode-encryption scheme of native Fabric, the performance loss of this method lies within an acceptable range.

Author Contributions: Conceptualization, Y.W.; methodology, W.G.; validation, Y.W.; formal analysis, X.H.; investigation, Y.W.; writing—original draft preparation, W.G.; writing—review and editing, Y.W.; supervision, X.H.; funding acquisition, X.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research work is supported by the National Natural Science Founds of China (62072368, U20B2050), Key Research and Development Program of Shaanxi Province (2022GY-040), Basic Research in Natural Science and Enterprise Joint Fund of Shaanxi (2021JLM-58).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

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

References

1. Sousa, J.; Bessani, A.; Vukolic, M. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Luxembourg, 25–28 June 2018; pp. 51–58.
2. Wutthikarn, R.; Hui, Y.G. Prototype of blockchain in dental care service application based on hyperledger composer in hyperledger fabric framework. In Proceedings of the 22nd International Computer Science and Engineering Conference, Chiang Mai, Thailand, 21–24 November 2018; pp. 1–4.
3. Ferrag, M.A.; Derdour, M.; Mukherjee, M.; Derhab, A.; Maglaras, L.; Janicke, H. Blockchain technologies for the internet of things: Research issues and challenges. *IEEE Internet Things J.* **2019**, *6*, 2188–2204.. [CrossRef]
4. Brandenburger, M.; Cachin, C.; Kapitza, R. Trusted computing meets blockchain: Rollback attacks and a solution for hyperledger fabric. In Proceedings of the 38th Symposium on Reliable Distributed Systems (SRDS), Lyon, France, 1–4 October 2019; pp. 32400–32409.
5. Brandenburger, M.; Cachin, C.; Kapitza, R.; Sorniotti, A. Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric. *arXiv* **2018**, arXiv:1805.08541.
6. Cheng, R.; Zhang, F.; Kos, J.; He, W.; Song, D. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy, Stockholm, Sweden, 17–19 June 2019; pp. 185–200.

7. Fan, J.; Chen, J.; Shen, R. Sgx-based approach for blockchain transactions security and privacy protection. *J. Appl. Sci.* **2021**, *39*, 17–28.
8. Zhou, L.; Wang, L.; Ai, T.; Sun, Y. BeeKeeper 2.0: Confidential blockchain-enabled IoT system with fully homomorphic computation. *J. Abbr.* **2008**, *10*, 142–149. [CrossRef] [PubMed]
9. Dai, W.; Wang, Q.; Wang, Z.; Lin, X.; Jin, H. Trustzone-based secure lightweight wallet for hyperledger fabric. *J. Parallel And Distrib. Comput.* **2021**, *149*, 66–75. [CrossRef]
10. Müller, C.; Brandenburger, M.; Cachin, C.; Felber, P.; Gttel, C.; Schiavoni, V. Tz4fabric: Executing Smart Contracts with ARM TrustZone. In Proceedings of the 2020 International Symposium on Reliable Distributed Systems, Shanghai, China, 21–24 September 2020; pp. 31–40.
11. Yannick, K.; Yi, W.; Wang, J. A lightweight trusted execution environment construction method for fabric chaincode based on sgx. *Netinfo Secur.* **2022**, *22*, 73–83.
12. Zhu, Y.; Song, W.; Wang, D.; Ma, D.; Chu, C. Ta-spesc: Toward asset-driven smart contract language supporting ownership transaction and rule-based generation on blockchain. *IEEE Trans. Reliab.* **2021**, *99*, 1255–1270. [CrossRef]
13. Sharma, A.; Schuhknecht, F.M.; Agrawal, D.; Dittrich, J. Blurring the lines between blockchains and database systems: The case of hyperledger fabric. In Proceedings of the 2019 International Conference on Management of Data, Amsterdam, The Netherlands, 1–5 July 2019; pp. 105–122.
14. Gorenflo, G.; Lee, S.; Golab, L.; Keshav, S. FastFabric: Scaling hyperledger fabric to 20000 transactions per second. In Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Seoul, Korea, 14–17 May 2019; pp. 455–463. [CrossRef]
15. Wan, S.; Li, M.; Liu, G.; Wang, C. Recent advances in consensus protocols for blockchain: A survey. *Wirel. Netw.* **2020**, *26*, 5579–5593. [CrossRef]
16. Guo, Q.; Zhang, D.; Liang, C.; Liu, X.; Song, J. Design and implementation of proxy-protected proxy signature based on sm2. *Chin. J. Netw. Inf. Secur.* **2017**, *3*, 47–54.
17. Androulaki, E.; Manevich, Y.; Muralidharan, S.; Murthy, C.; Laventman, G. Hyperledger fabric: A distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15.
18. Christidis, K.; Devetsikiotis, M. Blockchains and smart contracts for the internet of things. *IEEE Access* **2016**, *4*, 2292–2303. [CrossRef]
19. Badruddoja, S.; Dantu, R.; He, Y.; Upadhayay, K.; Thompson, M. Making smart contracts smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 24–28 October 2016; pp. 254–269.
20. Vacca, A.; Sorbo, A.D.; Visaggio, C.A.; Canfora, G. A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. *J. Syst. Softw.* **2020**, *174*, 110891. [CrossRef]
21. Hei, X.; Gao, W.; Wang, Y.; Zhu, L.; Ji, W. From hardware to operating system: A static measurement method of android system based on TrustZone. *Wirel. Commun. Mob. Comput.* **2020**, *4*, 8816023. [CrossRef]
22. Shepherd, C.; Arfaoui, G.; Gurulian, I.; Lee, R.P.; Markantonakis, K.; Akram, R.N.; Sauveron, D.; Conchon, E. Secure and trusted execution: Past, present, and future-a critical review in the context of the internet of things and cyber-physical systems. In Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA, 2, Tianjin, China, 3–26 August 2016; pp. 168–177.
23. Linaro, Op-Tee Documentation. Available online: https://optee.readthedocs.io/ (accessed on 27 December 2022).
24. GlobalPlatform Device Specifications: Trusted Execution Environment. Available online: https://globalplatform.org/ (accessed on 27 December 2022).
25. Foschini, L.; Gavagna, A.; Martuscelli, G.; Montanari, R. Hyperledger fabric blockchain: Chaincode performance analysis. In Proceedings of the 2020 IEEE International Conference on Communications, Guangxi, China, 28–31 October 2020; pp. 1–6.
26. Developers, G. Protocol Buffers. Available online: https://developers.google.com/protocol-buffers/(accessed on 27 December 2022).
27. Brandenburger, M.; Cachin, C.; Lorenz, M.; Kapitza, R. Rollback and forking detection for trusted execution environments using lightweight collective memory. In Proceedings of the 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Denver, CO, USA, 31 October 2017; pp. 157–168.