Data Privacy Protection of Industrial Blockchain

Huaqiu Long[1](✉), Jun Hou[2], Qianmu Li[3], Na Ma[3], Jian Jiang[4], Lianyong Qi[5], Xiaolong Xu[6], and Xuyun Zhang[7]

[1] Intelligent Manufacturing Department, Wuyi University, Jiangmen 529020, China
502080285@qq.com
[2] Nanjing Institute of Industry Technology, Nanjing 210023, China
[3] School of Cyber Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China
[4] Jiangsu Zhongtian Internet Technology Co., Ltd., Nantong 226009, China
[5] School of Information Science and Engineering, Qufu Normal University, Jining 272000, China
[6] Nanjing University of Information Science and Technology, Nanjing 210023, China
[7] Department of Computing, Macquarie University, Sydney, Australia

Abstract. This paper studies the data privacy protection of industrial block chain. Aiming at the privacy leakage problem of industrial blockchain, combining symmetric encryption and homomorphic encryption, The data privacy protection method is proposed to ensure the confidentiality and privacy of industrial block chain and improve the privacy security of industrial enterprises. This paper designs common basic chain code, security service chain code and privacy protection chain code in the chain code layer, USES AES algorithm to encrypt sensitive information of industrial enterprises, and USES Paillier algorithm to encrypt maintenance cost. The chain code of privacy protection can be used for security access and fault event audit to guarantee the privacy security of industrial enterprises. In addition, in order to achieve the unity of call chain code, the call chain code is encapsulated to provide support for fast and highly concurrent upload data and access data. Finally, the existing sensitive and private data storage methods of industrial block chain are analyzed, and the experimental comparison with the method in this paper proves the effectiveness of this method.

Keywords: Blockchain · Data privacy protection · Fault diagnosis

1 Introduction

Blockchain technology is a distributed, non-tamper-proof sharing technology that can realize the common storage of data and achieve decentralization in a distributed environment without mutual trust [1]. It is an emerging solution to improve the traditional centralized architecture. In the blockchain platform of remote industrial fault diagnosis, users' fault logs and diagnostic data are stored in clear text in LevelDB or CouchDB nodes [2–6]. Users can directly obtain all the information of fault events, which lacks privacy and confidentiality guarantee and is easy to leak the privacy of enterprise users.

L. Qi et al. (Eds.): CloudComp 2020, LNICST 363, pp. 83–99, 2021.
https://doi.org/10.1007/978-3-030-69992-5_7

In order to avoid the leakage of industrial enterprise privacy caused by the public storage of data, this paper adopts the fusion protection mode of AES algorithm and Paillier algorithm to design a data privacy protection scheme. The schema is shown in Fig. 1.

(1) **Client-side privacy protection middleware** [7–9]. The user calls the specified chain code function by executing the command on the client side, and interacts with the ledger for data. Each call requires the operation-related parameters to be written into the complex command. Therefore, in order to reduce the performance degradation caused by complex chain code invocation and satisfy fast and highly concurrent client requests, the process of chain code invocation is encapsulated in the client privacy protection middleware based on Node.js to provide a unified interface for the connection between client applications and chain code. The client-side privacy protection middleware is divided into fault event submission SDK and fault event query SDK. The fault event submission SDK is responsible for submitting the data of the client, uploading the request, and executing the key-value uploading operation. The FAULT event query SDK is responsible for submitting the data query request of the client and performing the fault event history query or fault event status query operation.

(2) **Chain code** [10–12]. In order to satisfy the security storage of different types of fault logs and diagnostic data, this paper combines symmetric encryption algorithm and homomorphic encryption algorithm, designs common security service chain code and privacy protection chain code based on Golang language, and provides confidentiality guarantee for uploading, querying and auditing of fault logs and diagnostic data. Among them, the security service chain code provides symmetric encryption interface and homomorphic encryption interface, which is designed based on AES algorithm, and homomorphic encryption interface is designed based on Paillier cipher system, supporting key generation, data encryption and decryption, data homomorphic operation and other functions. The chain code of privacy protection needs to invoke the interface in the chain code of security service to carry out different levels of privacy protection on the data request submitted by the client, so as to protect the writing, reading and auditing of data privacy protection and improve the privacy security of industrial enterprises in the distributed storage environment. In order to reduce the performance degradation caused by chain code execution, the basic chain code is designed to provide common data storage and query, as well as the common data processing function of the chain code layer.

(3) **Books** [13]. In this scheme, CouchDB is used as the data repository to store the fault log and diagnosis data submitted by the user and the mixed key information of the user.

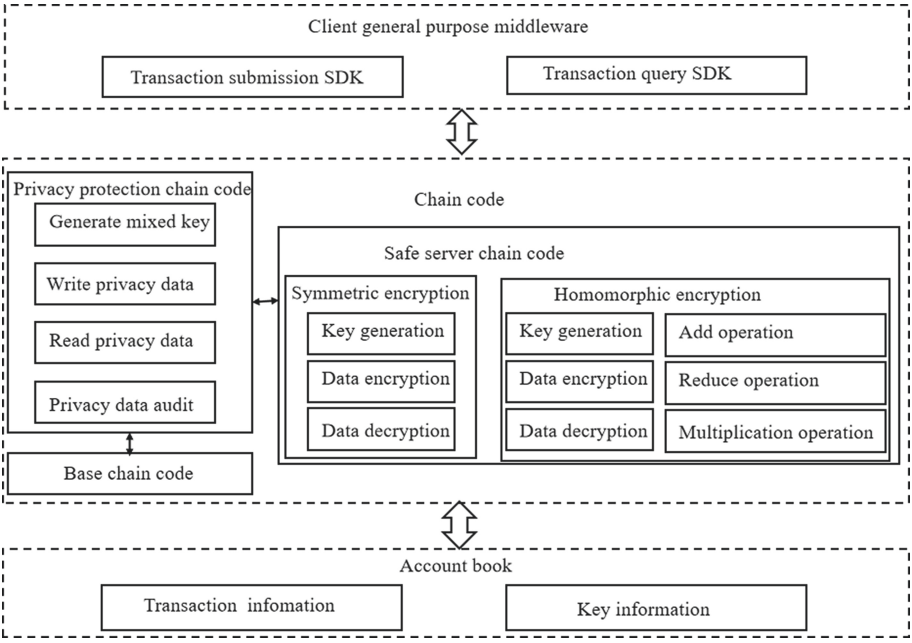

Fig. 1. The architecture of block chain data privacy protection scheme based on fusion protection

2 Design of Chain Code with Privacy Protection Function

The chain code function class diagram of block chain data privacy protection scheme is shown in Fig. 2. BaseChaincode relies on the shim API at the bottom of the blockchain, providing basic data read-write chain code functions, such as read(), Write(), and general data processing interfaces, such as getJSON(), getCurrnentUser(), etc. The security service chain code relies on the Golang native API, which supports AES symmetric encryption and Paillier homomorphic encryption, provides key generation, data encryption and decrypting, and specific mathematical operation interfaces, such as GeneratePaillierKey(), EncryptPaillier(), EncryptAES(), Add(), etc. The privacy protection chain code inherits the basic chain code and extends the chain code function to meet the specific data processing requirements. In addition, the privacy chain code relies on the security service chain code to invoke the corresponding interface for specific data encryption, decryption, and auditing functions, including genHK(), writeByEncrypt(), readByDecrypt(), writePvt(), readPvt(), Compute(), and auditPvt().

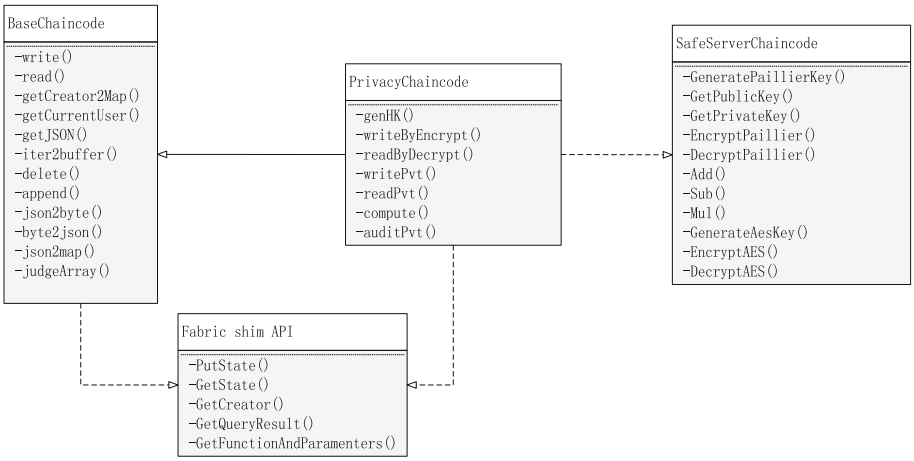

Fig. 2. Class diagram of chain code function.

2.1 Chain Code Interaction Process

Chain code is an important component connecting the client and the underlying ledger, which is divided into system chain code and user chain code. The chain code involved in this paper is user chain code. System chain code is responsible for logical processing of fault events, system configuration, etc., and is solidified in the system; The user chain code is written by the user, responsible for executing the custom processing logic, running in the Docker container, connecting with Peer nodes through gRPC, and communicating with each other through sending ChaincodeMessage messages. The interactive process is shown in Fig. 3, and the specific steps are as follows:

(1) After the gRPC connection is created, the user chain code calls the shim.Start() method, sends a ChaincodeMessage_REGISTER to the Peer node for registration, and waits to receive the response from the Peer node, at which time the state is created;
(2) After the Peer node receives the ChaincodeMessage_REGISTER, it is registered in the local Handler structure and returns the ChaincodeMessage_REGISTERED to the chain code. At this time, the state of the Peer node is ESTABLISHED. After the chain code receives the ChaincodeMessage_REGISTERED, the update state is established and the chain code registration is completed.
(3) Peer node sends ChaincodeMessage_READY to the chain code, and the update status is Ready. The chain code is also updated to Ready after receiving ChaincodeMessage_READY.
(4) Peer node sends ChaincodeMessage_INIT message to chain code and initializes the chain code.
(5) After receiving the ChaincodeMessage_INIT message, the chain code container calls Init() method to initialize, and returns the ChaincodeMessage_COMPLETED message to the Peer node after success. The chain code initialization is complete, and the chain code state is callable at this time.

(6) Peer node sends ChaincodeMessage_TRANSACTION message to chain code and executes chain code call.
(7) After receiving the ChaincodeMessage_TRANSACTION message, the chain code calls the Invoke() method, executes the specific chain code function logic, and sends the database operation message to the Peer node.
(8) After receiving the database operation message, the Peer node executes the corresponding data operation and returns the ChaincodeMessage_RESPONSE message to the chain code.
(9) Send ChaincodeMessage_COMPLETE to Peer node after the chain code call is completed.
(10) During the above message interaction, Peer nodes and chain codes regularly send ChaincodeMessage_KEEPALIVE messages to each other to ensure that they are online.

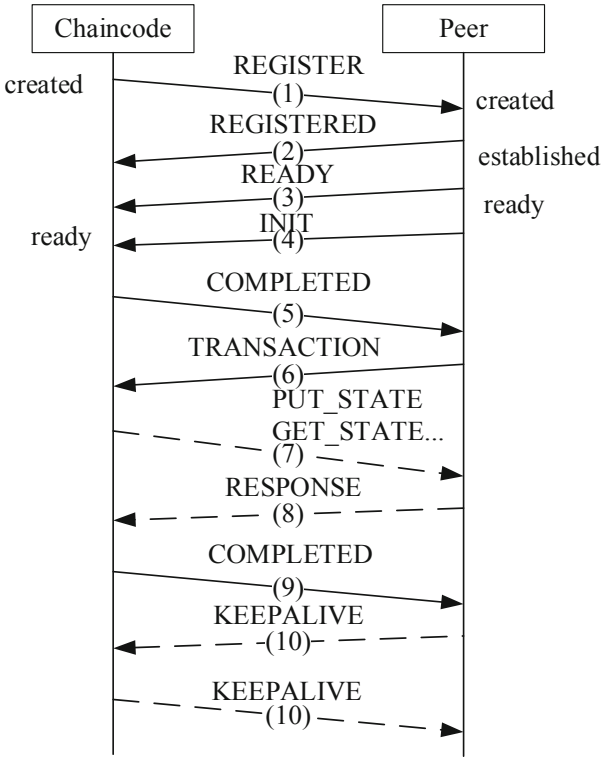

Fig. 3. Message interaction between the chain code and the Peer node.

The middle layer of chain code and Peer node interaction is called Shim layer. The Shim layer provides a series of Shim apis to interact with the underlying ledger for developers to write user chain codes. In this paper, the Shim API on the basis of the relevant chain code design.

2.2 Basic Chain Code

The write() function is responsible for storing individual key-value data in the ledger. This function takes the key name K and the corresponding value V as arguments, calls the PutState() interface of the Shim layer, adds a pair of key values to the ledger, or updates the value of the specified key name. The written key is appended to Writes first, and as soon as the billing node validates, the key is written to the ledger. In the process of writing data, errors may occur, resulting in the failure of data writing, so the resulting errors need to be handled (all chain code functions in this article need to be monitored when performing write/read data operations, this step is omitted in the related chain code function algorithm later, but it is essential in the actual operation). The implementation logic of the write() function is shown in Algorithm 1.

Algorithm 1 write() function

Input: The key name needs to be written k, the value v corresponding to the key name

1: **function** write(k, v)
2: $err_{PS} \leftarrow PutState(k,v)$
3: **if** err_{PS} *is not null* **then**
4: **reture** $Error(err_{PS}.Error())$
5: **end if**
6: **reture** *Success(null)*
7: **end funtion**

The read() function is responsible for querying the value of the specified key from the ledger. This function takes the K that needs to be queried as an argument, calls the GetState() interface of the shim layer of the blockchain, and reads the value corresponding to K from the ledger. The read() function also needs to return the error information that occurred during the query to the client, and if there are no errors, return the query results to the client. The implementation logic of the read() function is shown in Algorithm 2.

The data processing interface of the basic chain code is responsible for the data processing logic of the chain code layer and provides the calling interface for the chain code function, including the following interfaces: getCreator2map(), getCurrentUser(), getJSON(), Iter2Buffer(), Append(), Delete(), etc. GetCreator2map() is used to obtain and parse the identity certificate of the current client user from the ledger. GetCreator2map() extracts the user's identity certificate from signatureheader.Creator from signedProposal by calling the fault event submitter interface at the bottom of the blockchain GetCreator(), parses the properties of the certificate, and returns. CurrentUser() is used to obtain the current client user ID from the ledger identity certificate (in this case, the user ID is the unique ID that the user registers in the blockchain, which is the user name). Other data processing interface functions in the basic chain code are shown in Table 1.

Algorithm2 read() function

Input: The key name needs to be read k

Output: look up the read value $kBytes$

1 : **function** read(k)

2 : $kBytes, err_{GS} \leftarrow GetState(k)$

3 : **if** err_{GS} *is not null* **then**

4 : **reture** $Error(err_{GS}.Error())$

5 : **end if**

6 : **reture** $Success(kBytes)$

7 : **end funtion**

Table 1. Partial data processing interface in basic chain code.

Interface	Functional description
getJSON	The JSON string data is read from the account book, converted into a JSON object, and called by the chain code function
iter2buffer	Cache the iterator data in the buffer
append()	Append elements to an array
delete()	Deletes elements from an array
json2byte()	JSON objects are converted into bytes
byte2json()	The byte is converted to a JSON object
json2map()	JSON objects are converted to maps
judgeArray()	Determine whether the elements in the array exist

(1) Safe Server Chain Code

SafeServerChaincode supports AES symmetric encryption algorithm and Paillier cryptography system, combined with Golang native interface, provides key generation, data encryption, data decryption, data homomorphism operation and other interfaces. Among them, symmetric encryption provides key generation, data encryption, data decryption and other interfaces, and the functional description of each interface is shown in Table 2. Homomorphic encryption provides functions such as key generation, key acquisition, data encryption, data decryption, data addition operation, subtraction operation and multiplication operation. The functional description of each interface is shown in Table 3.

(2) Privacy Chain Code

PrivacyChaincode inherits the basic chain code, relies on Shim API and security service chain code, and provides the storage and query functions of data using different encryption types and fusion protection.

Table 2. Symmetric encryption interface

Interface	Functional description
GenerateAESKey()	Generation of AES session key by salting mechanism
EncryptAES()	Use AES to encrypt the plaintext
DecryptAES()	Use AES to decrypt the ciphertext

Table 3. Homomorphic encryption interface

Interface	Functional description
GeneratePaillierKey()	Generate the Paillier key
GetPublicKey()	Get public key
GetPrivateKey()	Get private key
EncryptPaillier()	Data encryption
DecryptPaillier()	Data decryption
Add()	Ciphertext addition
Sub()	Ciphertext subtraction
Mul()	Ciphertext multiplication

In order to ensure the privacy security in the fault log and diagnosis data submitted by industrial enterprises, each registered user in the scheme in this paper holds mixed key information and encrypts the fault log and diagnosis data submitted by users. Hybrid key includes three parts: PubKey Paillier the public key and secret key PriKey Paillier, AES initial key AesKey PubKey responsible for encrypt plaintext data, based on Paillier encryption logic PriKey responsible for unlock data, based on Paillier decryption logic AesKey is responsible for providing initial key value of AES encryption algorithm, when the initial key when the need for AES encryption salt processing, can guarantee the real key to encrypt and decrypt every time different. The structure of mixed key data based on Golang is as follows:

```
type HybridKey struct {
    PriKey      string   `json:"prikey"`
    PubKey      string   `json:"pubkey"`
    AesKey      string   `json:"aeskey"`}
```

The writePvt() function is responsible for writing fault logs and diagnostic data to the ledger based on fusion protection. This function inherits the data storage logic of the write() function and can protect the privacy of data with industrial enterprise privacy.

First, the parameters received are converted into the specified data privacy protection structure (such as the structure of the privacy order data in Sect. 5 of this article). The data privacy protection structure contains four types of attributes: privacy attribute priAttr, non-privacy attribute pubAttr, maintenance cost money, and failure event type TXType. Privacy attribute refers to the sensitive data in the failure event, such as mobile phone number; Non-private attributes refer to data that can be exposed to others, such as time of failure events; Maintenance cost refers to the expenditure or income of completing a failure event; Failure event types are used to count a certain type of failure event. The writePvt() function supports different encryption methods for different attributes: AES encryption privacy attributes, Paillier encryption maintenance costs. The implementation logic of writePvt() function is shown in Algorithm 3, and the general process is as follows:

Algorithm 3 writePvt() function

Input: The key name needs to be written *key*, the relevant parameters needs to be written *args*

```
1: function writePvt(key,args)
2: ret ← PrivateData(args)
3: uid ← getCurrentUser()
4: hk ← getJSON("HK" + uid)
5: if ret['priAttr'] is not null then
6:     s ← DecryptPaillier(hk.PriKey,hk.AesKey)
7:     aeskey ← GenerateAESKey(s)
8:     for k,v in range(ret['priAttr'] ) do
9:         ret['priAttr'][k] ← EncryptAES(v,aeskey)
10:    end for
11: end if
12: if ret['money'] is not null then
13:    ret['money'] ← EncryptPaillier(hk.PubKey,ret['money'])
14: end if
15: retByte ← json2byte(ret)
16: PutState(key,retByte)
17: end function
```

(1) Convert the input fault log and diagnostic data into the specified data privacy protection PrivateData structure;
(2) Get the user ID of the current client through getCurrentUser() function, and get the user's mixed key according to the user ID;

(3) If the fault event has privacy attributes, salt the decrypted AES initial key and encrypt all privacy attributes via the EncryptAES() function;
(4) EncryptPaillier() encryption using Paillier public key if there is maintenance fee for the failure event;
(5) PutState(), a data storage interface based on the Shim layer of block chain, stores encrypted fault logs and diagnosis data into the ledger.

The readPvt() function is responsible for reading the fault log and diagnostic data of a given user from the account book in a fusion protection-based manner, returning plaintext data.This function inherits the data query logic of the read() function, invokes the security service chain code interface to decrypt the obtained fault event ciphertext, and returns the decrypted data to the client.The logic of the readPvt() function is similar to that of the writePvt() function. The implementation logic is shown in Algorithm 4, and the general process is as follows:

Algorithm 4 readPvt() function

Input: User ID *uid*， specified key name *k*

```
1: function readPvt(uid, k)
2: res ← getJSON(k)
3: ret ← json2map(res)
4: hk ← getJSON('HK' + uid)
5: if ret['priAttr'] is not null then
6:     s ← DecryptPaillier(hk.PriKey,hk.AesKey)
7:     aeskey ← GenerateAESKey(s)
8:     for k, v in range(ret['priAttr']) do
9:         ret['priAttr'][k] ← DecryptAES(v,aeskey)
10:    end for
11: end if
12: if ret['money'] is not null then
13:    ret['money'] ← DecryptPaillier(hk.PriKey,ret['money'])
14: end if
15: retBytes ← json2byte(ret)
16: reture Success(retBytes)
17: end function
```

(1) Query the fault log and diagnostic data of the specified key name from the account book;
(2) Query the mixed key of the designated user from the account book;

(3) If the fault log and diagnostic data contain the privacy attribute(priAttr), DecryptAES() decrypts all the privacy attribute with the salted AES initial key;
(4) If there is a repair cost for the failure event, the repair cost is encrypted with Paillier public key through EncryptPaillier();
(5) Returns the decrypted fault log and diagnostic data to the client.

3 Design of Fault Detection Middleware for Client Privacy Protection

The fault detection middleware for client privacy protection is written based on Node.js. On the block chain Node SDK, it encapsulates common chain code functions for client users, including fault event submission SDK and fault event query SDK. Before performing the failover detection middleware, the user must ensure that the client is connected to the blockchain. In the scheme of this paper, users connect to the block chain by setting the network configuration file, in which the network configuration file information includes CA server address, database address, channel path, chain code name, Peer node server address, Orderer node server address and other configuration parameters. After the successful connection, the chain code can be called to conduct data interaction with the blockchain ledger.

3.1 Failure Event Submission SDK

The implementation interface of the FAULT event submission SDK is the invokeTx() method, which provides the function of uploading or updating the fault log and diagnosing data for users on the client side, and invokes the chain code function of the fault event category in the chain code layer.

The invokeTx() method implementation logic is shown in Algorithm 5. After the client executes the invokeTx() method, it will write or update the key value to the ledger, and new fault event or block will be generated during the execution process. Therefore, it is necessary to specify the endorsement node, add the node endorsement and fault event to the block, and then update the local ledger through the node. Before the invokeTx() method is executed, the connection configuration file must be specified to connect the block chain. In the connection configuration file, CA server, database, server, database, channel, chain code, Peer node, Orderer node and other network configuration parameters of the block chain need to be set. After the successful connection, the module interface in the Blockchain Node SDK is called with the chain function name and chain function as the input parameters of the invokeTx() method to obtain the relevant information of the current blockchain network, and the channel module interface sendTransactionProposal sends the fault event to the Node and returns the submission result of the fault event. The results include txID (fault event ID), blockNum (block number of fault event), isDelay, and so on, which can be used by the client user to analyze the execution of the fault event.

Algorithm 5 invokeTx() method

Input: Specified chain codefunction name *ccf*, required parameters of chain codefunction *args*

Output: transaction submission results *invokeResult*

```
1: function invokeTx(ccf,..,args)
2: if ccf is null then
3:     reture "Missing chain codefunction name"
4: end if
5: tgs ← [] // endorsement node
6: if len(this.targets) < 1 then // this refers to the current fabric network
7:     tgs.push(this.peer)
8: else
9:     tgs ← this.targets
10: end if
11: tid ← this.client.newTransactionID()
12: this.txids.set(tid.getTransactionID(),null)
13: req ← {targets:tgs,
14:         chaincodeId : this.chaincode_id,
15:         fcn : ccf,
16:         args : args,
17:         chainId : this.channel_id,
18:         txId : tid}
19: invokeResult ← this.channel.sendTransactionProposal(req)
20: reture invokeResult
21: end funtion
```

3.2 Failure Event Query SDK

The implementation interface of the FAULT event query SDK is queryTx() method, which provides the client with the unified fault log and diagnostic data query function for users, invokes the chain code function of non-indorsed fault event category in the chain code layer, and returns the chain code invocation result to the client.

The queryTx() method reads key-value data from the ledger, and execution does not result in a new failure event or block, so there is no need to endorse a failure event.

Before execution queryTx() method, the client should also with block chain network connection, with chain code function name as an input parameter, call blocks in the chain of module interface to get information about the current industrial chain network fault detection blocks, queryByChaincode interface module using channels to submit failure event query request, after the query returns the query results, the results content depends on the chain code function returns the result. The queryTx() method implementation logic is shown in Algorithm 6.

Algorithm 6 queryTx() method

Input: Specified chain code function name *ccf*, required parameters of chain code function *args*

Output: transaction query results *queryResult*

```
1: function queryTx(ccf,..,args)
2: if ccf is null then
3:     reture "Missing chain code function name"
4: end if
5: tgs ← []
6: tgs.push(this.peer)
7: tid ← this.client.newTransactionID()
8: req ← {targets:tgs,
9:         chaincodeId : this.chaincode_id,
10:        fcn : ccf,
11:        args : args,
12:        chainId : this.channel_id,
13:        txId : tid}
14: queryResult ← this.channel.queryByChaincode(req)
15: reture queryResult
16: end funtion
```

4 Blockchain Sensitive Privacy Data Storage Method

Block chain, introduced the concept of sensitive private data collection. Based on SideDB (lateral database) mechanism to save the failure events of sensitive private data, allow specific organization node access to sensitive private data, other members without permission is only know the sensitive private data exist, don't know the real content of sensitive private data.

Sensitive privacy data set includes data in two parts: sensitive privacy data entity and hash value of sensitive privacy data. Sensitive private data entities are accessible only to

nodes with specific permissions, stored in the node's private state database (PrivateDB), and transmitted between nodes with permissions via the Gossip protocol, with no sorting services involved. The hash value of sensitive private data is written into the public State database (PublicDB) of each node in the channel after endorsement and sorting as the proof of the existence of failure events.

To execute sensitive private data storage, complex chain code commands are required to define sensitive private data set configuration files, which include the following information: Collection name (Name), signature policy, minimum recognized node (requiredPeerCount), maximum number of Peer nodes (maxPeerCount), and blockToLive (duration of sensitive private data retention). The collection name indicates where the data is stored in the ledger. The signature policy defines an organization member that persists the collection data, such as "OR('org1msp.member', 'org2mSP.member')"; The minimum number of approved nodes means the minimum number of Peer nodes that need to distribute sensitive private data before the endorsement signature is returned to the client. The maximum number of Peer nodes refers to the number of Peer nodes that endorse nodes to distribute sensitive private data to save data redundancy. BlockToLive should be set to 0 if it is necessary to keep sensitive private data indefinitely. BlockToLive should be set to 0 if it is necessary to keep sensitive private data indefinitely. After the configuration is complete, the sensitive private data storage interfaces PutPrivateData() and the sensitive private data query interface GetPrivateData() interact with the ledger.

When the data needs to be kept secret among members in the channel and propagated among all organizations, certain members are required to have access to part of the data of the failure event, and the Orderer node needs to be kept secret, then the block chain USES sensitive private data storage method to improve the security of the fault log and diagnostic data.

Although the blockchain sensitive and private data storage method can guarantee sensitive data privacy in fault events, it still has shortcomings, which are mainly reflected in two aspects: (1) fault log and diagnostic data are still stored in the ledger of the node in clear text, and attackers can easily obtain fault event information directly from the ledger; (2) Node access authorization needs to provide complex operations. Fault logs and diagnostic data are stored in two state libraries, which can easily reduce the performance of the block chain itself.

5 Experiment and Analysis

The block chain data privacy protection scheme proposed in this paper is mainly aimed at industrial fault detection block chain users to protect privacy among users. Based on high concurrent request and high load, this experiment USES three methods, namely, the proposed scheme, the blockchain sensitive private data storage method and the original industrial fault detection blockchain (without privacy protection), to process client requests, obtain performance test results, and verify the feasibility of the proposed scheme by evaluating the test results. The test measures are Transactions Per Second (TPS) and average Time Per Transaction (TPT) Per Transaction. TPS is used to evaluate the number of data requests that can be processed Per unit Time, and TPT is used to evaluate the Time required to process a data request under a specified amount of concurrency.

This experiment mainly tests the upload data request and access data request submitted by the client, executes the client-side middleware invokeTx() method and queryTx() method in batches, and invokes different chain code functions. Among them, the chain code functions called when uploading and accessing data are: writePvt() and readPvt(); the chain code functions called when uploading and accessing data by native industrial fault detection block chain are write() and read(); the chain code interfaces called when uploading and accessing data by sensitive private data storage method of industrial fault detection block chain are PutPrivateData() and GetPrivateData() respectively. In the experimental environment, the host computer with CPU Intel(R)Core(TM) i7–6700@3.40 GHz and four Core eight threads was used to simulate data requests for 10 rounds with different concurrency values, calculate the average TPS and average TPT under each concurrency value, and finally analyze the results.

(1) Upload data to request test results

The experimental results are shown in Fig. 4. The horizontal axis represents the concurrency of the upload data request, while the vertical axis represents the average TPS under each concurrent amount. The concurrency value ranges from 0 to 2000 with intervals of 100. Can be seen from the Fig. 4, under the same concurrency, this scheme can handle the upload data request number per unit time is basic block chain, close to the native industrial fault detection when the concurrency value is less than 700, this scheme can handle the upload data request number per unit time is far more than the industrial chain of fault detection block sensitive private data storage methods. Therefore, the blockchain data privacy protection scheme proposed in this paper does not reduce the performance of industrial fault detection blockchain when executing the upload data request.

(2) Access data request test results

The experimental results are shown in Fig. 5. The horizontal axis represents the amount of concurrency of access data request, while the vertical axis represents the average TPS under each amount of concurrency, with the range of concurrency increasing successively from 0 to 4000, with the interval of 100. As can be seen from Fig. 5, with the same amount of concurrency, compared with the number of requests that can be processed per unit time in the original industrial fault detection block chain, the scheme in this paper has been reduced, slightly higher than the sensitive private data storage method of the industrial fault detection block chain. Therefore, the data privacy protection scheme of industrial fault detection blockchain based on fusion protection proposed in this paper slightly reduces the performance of industrial fault detection blockchain when executing access data request, and is superior to the sensitive privacy data storage method of industrial fault detection blockchain.

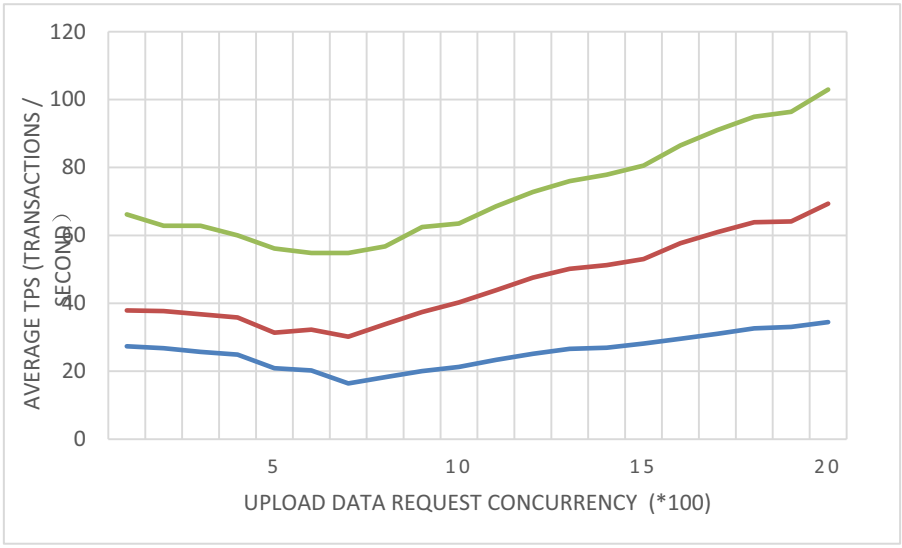

Fig. 4. Average TPS test results for upload data requests.

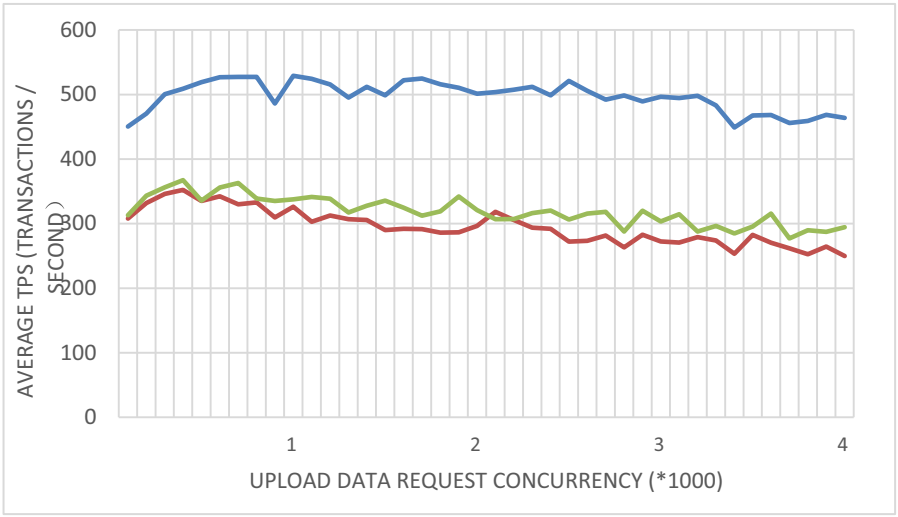

Fig. 5. Average TPS test results for access data requests.

6 Conclusion

This paper studies the data privacy protection storage of block chain, proposes a data privacy protection scheme based on fusion protection, introduces encryption method to partially encrypt the data uploaded to block chain, and realizes the data confidentiality storage and audit in distributed ledger. The usability of the proposed scheme is verified

by experimental comparison with the blockchain sensitive private data storage method and the native blockchain.

Funding. This work was supported in part by the Key Research Base of Philosophy and Social Sciences in Jiangsu Universities "Huang Yan-Pei Vocational Education Thought Research Society Academic Center", 2019 Industrial Internet Innovation and Development Project from Ministry of Industry and Information Technology of China, 2018 Jiangsu Province Major Technical Research Project "Information Security Simulation System", Fundamental Research Funds for the Central Universities (30918012204), 2019 Jiangmen Basic and Theoretical Scientific Research Projects "Research on Feature Selection and Data Fusion of Industrial Big Data for Intelligent Manufacturing".

References

1. Li, Q., Song, Y., Zhang, J., Sheng, V.S.: Multiclass imbalanced learning with one-versus-one decomposition and spectral clustering. Expert Syst. Appl. **147**, 113152 (2020). https://doi.org/10.1016/j.eswa.2019.113152
2. Li, Q., Yin, X., Meng, S., Liu, Y., Ying, Z.: A security event description of intelligent applications in edge-cloud environment. J. Cloud Comput. **9**(1), 1–13 (2020). https://doi.org/10.1186/s13677-020-00171-0
3. Hou, J., Li, Q., Tan, R., Meng, S., Zhang, H., Zhang, S.: An Intrusion tracking watermarking scheme. IEEE Access **7**, 141438–141455 (2019). https://doi.org/10.1109/ACCESS.2019.2943493
4. Hou, J., Li, Q., Cui, S., Meng, S., Zhang, S., Ni, Z., Tian, Y.: Low-cohesion differential privacy protection for industrial Internet. J. Supercomput. **76**(11), 8450–8472 (2020). https://doi.org/10.1007/s11227-019-03122-y
5. Li, Q., Tian, Y., Wu, Q., Cao, Q., Shen, H., Long, H.: A cloud-fog-edge closed-loop feedback security risk prediction method. IEEE Access **8**(1), 29004–29020 (2020)
6. Li, Q., et al.: Safety risk monitoring of cyber-physical power systems based on ensemble learning algorithm. IEEE Access **7**, 24788–24805 (2019)
7. Li, Q., Meng, S., Wang, S., Zhang, J., Hou, J.: CAD: command-level anomaly detection for vehicle-road collaborative charging network. IEEE Access **7**, 34910–34924 (2019)
8. Li, Q., Meng, S., Zhang, S., Hou, J., Qi, L.: Complex attack linkage decision-making in edge computing networks. IEEE Access **7**, 12058–12072 (2019)
9. Li, Q., Wang, Y., Ziyuan, P., Wang, S., Zhang, W.: A time series association state analysis method in smart internet of electric vehicle charging network attack. Transp. Res. Rec. **2673**, 217–228 (2019)
10. Cui, S., Li, T., Chen, S.C., Shyu, M.-L., Li, Q., Zhang, H.: DISL: deep isomorphic substructure learning for network representations. Knowl.-Based Syst. **189** (2020). https://doi.org/10.1016/j.knosys.2019.105086
11. Meng, S., Li, Q., Zhang, J., Lin, W., Dou, W.: Temporal-aware and sparsity-tolerant hybrid collaborative recommendation method with privacy preservation. Concurrency Comput.-Pract. Exp. **32**(2) (2020). https://doi.org/10.1002/cpe.5447
12. Li, Q., Hou, J., Meng, S., Long, H.: GLIDE: a game theory and data-driven mimicking linkage intrusion detection for edge computing networks. Complexity **2020** (2020). https://doi.org/10.1155/2020/7136160. Article no. 7136160, 18 pages
13. Hou, J., Li, Q., Meng, S., Ni, Z., Chen, Y., Liu, Y.: DPRF: a differential privacy protection random forest. IEEE Access **7**, 130707–130720 (2019). https://doi.org/10.1109/ACCESS.2019.2939891