Cipherchain: A Secure and Efficient Ciphertext Blockchain via mPECK

Hailin Chen[1], Gang Xu[3], Yuling Chen[2], Xiubo Chen[1, 2, *], Yixian Yang[1, 2], Ruibin Fan[4], Kaixiang Zhang[4] and Huizhong Li[4]

Abstract: Most existing blockchain schemes are based on the design concept "openness and transparency" to realize data security, which usually require transaction data to be presented in the form of plaintext. However, it inevitably brings the issues with respect to data privacy and operating performance. In this paper, we proposed a novel blockchain scheme called Cipherchain, which can process and maintain transaction data in the form of ciphertext while the characteristics of immutability and auditability are guaranteed. Specifically in our scheme, transactions can be encrypted locally based on a searchable encryption scheme called multi-user public key encryption with conjunctive keyword search (mPECK), and can be accessed by multiple specific participants after appended to the globally consistent distributed ledger. By introducing execution-consensus-update paradigm of transaction flow, Cipherchain cannot only make it possible for transaction data to exist in the form of ciphertext, but also guarantee the overall system performance not greatly affected by cryptographic operations and other local execution work. In addition, Cipherchain is a promising scheme to realize the technology combination of "blockchain+cloud computing" and "permissioned blockchain+public blockchain".

Keywords: Blockchain, Cipherchain, cloud computing, mPECK.

1 Introduction

Bitcoin was proposed as the first truly decentralized digital currency in 2008. In the past, digital currency schemes were mostly based on the trust of currency issuers [Chaum (1983); Rivest (1997); Yang and Hector (2003)], while the Bitcoin and other decentralized cryptocurrency schemes are based on the blockchain technology. Blockchain maintains a continuously growing list of ordered transactions called blocks.

[1] Information Security Center, State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China.

[2] Guizhou University, Sate Key Laboratory of Public Big Data, Guizhou, 550025, China.

[3] North China University of Technology, School of Computing Science and Technology, Beijing, China.

[4] Webank, Shenzhen, 518000, China.

[*] Corresponding Author: Xiubo Chen. Email: flyover100@163.com.

Received: 29 November 2019; Accepted: 14 December 2019.

Essentially, this emerging underlying technology is an open, transparent and distributed replicated database.

Blockchain achieves the data characteristics of auditability and immutability. It has been proven theoretically robust with respect to data security [Garay, Kiayias and Leonardos (2015); Miller and LaViola Jr (2014)]. However, Bitcoin's sensitive data is processed and stored in the form of plaintext, which is the same in most other public blockchain systems. This so-called "openness and transparency" labeled as blockchain's advantage is at the expense of data being accessible to anyone. Bitcoin employs the "pseudonym pattern" to let users send transactions to the blockchain network with public key addresses as their identities, intended to eliminate the relationship between users' real identities and transactions logic. However, Meiklejohn et al. [Meiklejohn, Pomarole, Jordan et al. (2013)] have demonstrated that the map between Bitcoin transactions and identities is traceable. The Bitcoin protocol recommends users create new pseudonymous public keys for each transaction. Nevertheless as shown in Reid et al. [Reid and Harrigan (2013)], transactions with multiple inputs are generally generated with the same user signature. What's worse, user habits also lead to the loss of transaction privacy information. Koshy et al. [Koshy, Koshy and McDaniel (2014)] also demonstrate that it is possible to identify ownership relationships between Bitcoin addresses and IP addresses.

In order to realize the transaction privacy protection in the blockchain system, various mechanisms have been proposed or applied [Karame and Androulaki (2016)]. Especially in the field of cryptocurrency, it can be roughly divided into three kinds of schemes: mixed-coin scheme [Bonneau, Narayanan, Miller et al. (2014); Duffield and Diaz (2015)], off-chain transaction scheme [Heilman, Alshenibr, Baldimtsi et al. (2017); Green and Miers (2017)] and cryptography scheme [Miers, Garman, Green et al. (2013); Sasson, Chiesa, Garman et al. (2014)]. Most schemes intend to achieve unlinkability of multiple inputs and outputs of transaction against the Unspent Transaction Outputs (UTXO) model. These solutions are feasible for a cryptocurrency system with a simple payment transfer function. However, in order to achieve the expansion of the blockchain function rather than just limited to digital currency applications, the transaction data type and execution logic will be more diverse and complicated. The amount of information brought will make the transaction inputs and outputs have a stronger correlation. This may lead to more complicated improvement schemes, and make system performance and even privacy or security not guaranteed. In addition, permissioned blockchains typically remove digital currency attributes for consideration of policies or applications. The scheme that can be applied to previous public blockchains may not be able to compete in the permissioned blockchains without digital currency [Vukolić (2017)].

With the advent of consortium blockchain (as a category of permissioned blockchains), transaction privacy has become a more significant issue to be addressed. Therefore, we have witnessed some novel improvement schemes employed out of the consideration of engineering implementation and operating efficiency. Multi-channel technology is adopted in Hyperledger Fabric v1.0 [Androulaki, Barger, Bortnikov et al. (2018)]. In R3 Corda [Hearn (2016)], transactions will be only spread and "witnessed" between relevant parties. Hence, both Fabric and Corda adopt the physical isolation against distributed ledger to split the transaction flows of the entire blockchain network. This abandons the design of distributed general ledger, where system reaches global consensus on transactions and store them in the same ledgers. This seems to have lost the concept and characteristics of the "original" blockchain. The security feature require that transaction cannot be tampered and forged once it is packed into the block structure. However, if blockchain systems adopts physical isolation scheme, transaction will be agreed upon only in a limited node group, which is still questionable in terms of security compared with previous public blockchains [Gervais, Karame, Wüst et al. (2016)].

In this paper, we propose a novel ciphertext blockchain scheme called Cipherchain. In Cipherchain, we retain the design of the distributed ledger technology (DLT) generally used in the public blockchain. All newly generated transactions need to go through the same consensus process and eventually be stored on a globally consistent distributed ledger. The biggest difference between Cipherchain and the previous blockchain schemes is that newly generated transactions need to be encrypted and then appended in the form of ciphertext to the ledger. Specifically, the mPECK technology [Hwang and Lee (2007)] is employed to process transaction data.

We reinterpreted the concept "openness and transparency", which is generally used to describe the basic properties of blockchain technology. Furthermore, Cipherchain can serve as an immutable and consistent ledger, even if transaction data is presented not in the form of plaintext. It can perform auditing, verification, sharing, etc. over transaction ciphertext between authorized participants.

Unlike all previous blockchain systems following the order-execute architecture, Hyperledger Fabric v1.0 [Androulaki, Barger, Bortnikov et al. (2018)] creates a novel execute-order-validate paradigm. Previous blockchain systems cannot implement the data presented directly in the form of ciphertext. All nodes play the same role in executing and ordering transactions. Hence while receiving a transaction in the form of ciphertext, nodes cannot verify whether the transaction is legitimate. Thus in Cipherchain, we followed the design idea of Fabric and some adjustments were made in accordance with the special requirements of our scheme. We call Cipherchain the execution-consensus-update architecture, which allows the transaction data to be encrypted after it is executed, then reached global consensus and eventually updated to the general distributed ledger. Furthermore, local cryptographic computation overhead does not have much impact on system performance because of the modular design.

Cipherchain leverages both advantages of the public and the permissioned blockchains to serve our consideration of security and efficiency. The main contributions of our work can be summarized as follows:

- We proposed the first ciphertext blockchain scheme that can realize the privacy protection of transaction data directly. By introducing mPECK technology, our scheme can realize the transaction data in the form of ciphertext, making the transaction data and execution logic only shared between authorized participants while retaining the original attributes of previous Bitcoin-like blockchains.
- We adopt newly execution-consensus-update paradigm and two-level state database scheme. These make the independent consensus module affecting overall system performance be implemented efficiently.

- Our scheme can achieve the technology combination of "blockchain+cloud computing". Specifically, it takes advantage of cloud computing in data retrieval and storage to address the issue of data explosion that most blockchains may face. The process is illustrated in Fig. 1.

The remaining of the paper is organized as follows: We begin by introducing background and related work in Section 2, preliminary and assumption in Section 3. Section 4 provides an overview about the architecture design of Cipherchain. The detailed architecture design is described in Section 5 and necessary evaluation is showed in Section 6. Besides, some promising applications are described in Section 7. Finally, Section 8 concludes.

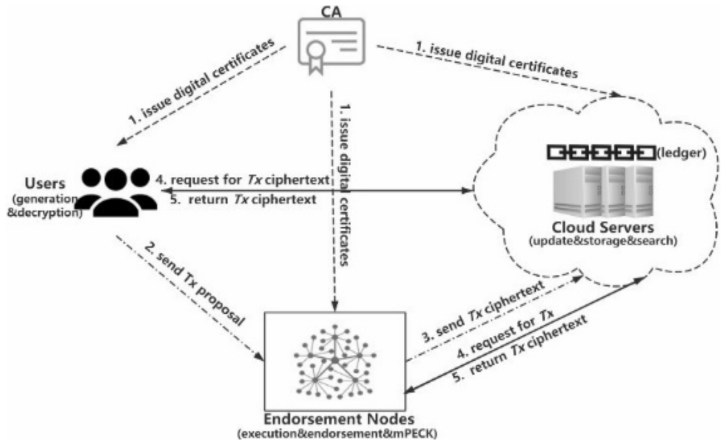


Figure 1: Cipherchain network composed of "cloud computing+permissioned blockchain"

2 Background and related work

2.1 Blockchain

In 2008, Nakamoto first proposed the blockchain in the Bitcoin White Paper [Nakamoto (2008)]. The miners in the blockchain network run the Proof of Work (PoW) consensus mechanism with their strong computing power. This maintains the normal operation and security of the system [Garay, Kiayias and Leonardos (2015)].

In recent years, many novel blockchain projects have emerged. The smart contract is added to blockchain while Ethereum [Buterin (2014)] came online. Blockchain technology is also moving towards commercial applications, which has ignited the rise of the consortium blockchain. In 2015, the Linux Foundation launched the open source project Hyperledger. Corda is the blockchain platform developed by R3 [Hearn (2016)], one of the leading blockchain consortiums. Unlike other blockchain schemes, Corda set "notary" nodes to maintain and update the ledger.

In most blockchain schemes (here we take Bitcoin as a reference), nodes are required to perform the same programs, including transaction execution, verification, consensus and storage. However, its shortcomings are also obvious:

Performance. Bitcoin-like schemes adopt the PoW mechanism or other similar ones. Although PoW is different from the previous consensus mechanisms such as Paxos [Lamport (1978)] and BFT, its idea is to select a qualified "leader" as well. However, leader is selected in the Byzantine environment [Lamport, Shostak and Pease (1982)], while the generation of valid blocks faces various complex network environments. At the same time, considering that the blockchain may face high transmission latency under the premise that the network quality cannot be guaranteed, the public blockchain protocols require the distributed data to reach the eventual consistency. Therefore, the blockchain system suspends the processing of newly generated transactions during the consensus phase. The result is that such a system cannot process the verification, consensus, and execution of transactions in parallel, which has become a bottleneck affecting the overall performance of the blockchain system.

After the advent of Ethereum [Wood (2014)], the smart contract was realized. However, the execution of smart contract make performance issues even more critical. A node may execute a long or even infinite loop of contract code while the program cannot automatically terminate, the system performance definitely drops dramatically. To address this issue, Ethereum adopted the "gas" mechanism based on its cryptocurrency. The amount of code executed is proportional to the amount of cryptocurrency that needs to be spent. Obviously, this mechanism is not adequate for the permissioned blockchain not based on cryptocurrency.

Privacy. In most public blockchains, data is exposed to all participants and stored in the form of plaintext. Since each transaction needs to be executed at all nodes, transaction logic and state update (if necessary) are transparent to them [Vukolić (2017)]. Consequently, analysis of transactions may yield useful information. Many of the privacy protection schemes mentioned above are based on the concept of "openness and transparency". They attempt to weaken the correlation between inputs and outputs, or between inputs (outputs) and transaction participants. However, the privacy protection cannot be guaranteed under the premise of the data presented in plaintext, according to the conclusions in [Meiklejohn, Pomarole, Jordan et al. (2013); Reid and Harrigan (2013); Koshy, Koshy and McDaniel (2014)] mentioned above. Besides, the inputs and outputs of the previous public blockchains are simply currency values. In the process of extending the blockchain function, it may be necessary to introduce multiple parameters into the input or output fields of the transaction as the logic becomes more complex.

2.2 Hyperledger fabric

As a consortium blockchain project, Hyperledger Fabric aims to promote the commercial application of blockchain across industries. Fabric system innovatively employs execute-order-validate mode [Androulaki, Barger, Bortnikov et al. (2018)], rather than the order-execute mode used by most public blockchains.

In Fabric, it can be divided into three kinds of blockchain nodes by function: endorsers, orderers, and committers. The endorser completes the simulation and endorsement of transaction proposals; the orderer packs and orders transactions according to a certain consensus strategy; the committer maintains and updates the ledger on a per-channel basis. The endorser and the committer are usually on the same physical server.

The transaction proposal is generated by a client with execution logic (chaincode), which is then broadcast to the blockchain network. Unlike previous broadcasts to all nodes, Fabric employs a multi-channel design, where the transaction is sent to a specific endorser group according to the endorsement policy determined by the chaincode and chaincode itself. For endorsers and other types of nodes that are not in the channel, it is impossible to receive or access the transaction. When specified endorsers receive the transaction in the channel, they simulate the proposal according to the transaction logic of the chaincode, and signs the valid transaction, thereby completing endorsement function. After completing the above operation, the endorsers will return the execution results and signatures to the client. While the client collects "sufficient" signatures from endorsers, it will submit the executed transaction and signatures to orderers. The orderers then order received transactions and pack them into blocks, which then will be sent to committers. The committers evaluate endorsement policy and check read-write conflict of each transaction, which is eventually appended to the local ledger.

As can be seen from the execution of the transaction flow, Fabric adopted a storage isolation strategy. The transaction data and execution logic will only be accessed by specific nodes. In summary, Fabric is based on the trust of the endorsers, more precisely in the premise of trusting the endorsement policy, to achieve the privacy and security attributes of system. Due to the multi-channel design, Fabric is usually a multi-chain system-that is, one channel corresponds to one blockchain structure.

Different from the design of Fabric, we still apply the global consensus mechanism commonly used in previous public blockchains to Cipherchain, as well as the global distributed ledger.

2.3 Searchable encryption

Traditional information retrieval methods are based on the plaintext system, where the server is fully aware of stored information. Searchable encryption is a secure search technology that provides ciphertext retrieval on semi-honest or malicious servers.

The searchable encryption was first proposed by Song et al. [Song, Wagner and Perrig (2000)], who established the first symmetric searchable encryption scheme. While symmetric searchable encryption is limited to single-user scenarios, public key searchable encryption addresses the whole issues well, and realizes the data sharing among multiple participants. Then in 2004, Boneh et al. [Boneh, Di Crescenzo, Ostrovsky et al. (2004)] proposed the first public-key encryption with keyword Search (PEKS), and gave a PEKS construction scheme based on anonymous IBE. This solution is suitable for implementing mail routing in an untrusted mail system.

The above scheme is mainly for single keyword search. Yet, in practical applications, this cannot cope with the case where a specific file is accurately located when the file data is large. Then conjunctive keyword search technology came into being. In 2005, Golle et al. [Golle, Staddon, Waters et al. (2004)] first proposed a searchable encryption scheme based on conjunctive keyword. Park et al. [Park, Kim and Lee (2004)] proposed a public key encryption with conjunctive field keyword search (PECK), and presented two construction schemes. Boneh et al. [Boneh and Waters (2007)] also proposed a scheme for conjunctive keyword search over encrypted data in 2007. In the blockchain application scenario based on the account model, the data-sharing mode is often one-to-many. That is, a single data owner publishes data, and multiple participants share, verify, or audit transaction data. The first one-to-many mode searchable encryption was constructed by Curtmola et al. [Curtmola, Garay, Kamara et al. (2011)] in 2006 based on Naor's broadcast encryption technology. The privacy protection mechanism of blockchain transaction employed in Cipherchain is based on the multi-user PECK scheme proposed by Hwang et al. [Hwang and Lee (2007)]. The idea is to use multi-receiver PKE [Baudron, Pointcheval and Stern (2000)] and randomness reuse [Kurosawa (2002)] to reduce the communication and computation cost in multi-user scenarios. Besides, it does not need a trusted third party.

3 Preliminary and assumption

3.1 Bilinear maps

We assume that G_1 and G_2 are the two multiplicative cyclic groups of order p, which is a certain large prime. The bilinear map $e: G_1 \times G_1 \to G_2$ is between these two groups. The bilinear map should be satisfied the following properties:

Bilinear: A map $e: G_1 \times G_1 \to G_2$ is bilinear if $e\left(u^x, v^y\right) = e(u,v)^{xy}$ for all $u, v \in G_1$ and any $x, y \in Z_p^*$.

Non-degenerate: The map e does not send all pairs in $G_1 \times G_1$ to the identity in G_2. If g is a generator of G_1 then $e(g,g)$ is a generator of G_2.

Computable: There is an efficient algorithm to compute $e(u,v)$ for any $u, v \in G_1$

3.2 Assumption

To facilitate the description of the scheme and simplify the details, we will make the following premise or assumptions:

1) Assume that after each transaction is generated, there are l fixed keyword fields without two identical ones.

2) The certificate authority (CA) is a fully trusted party, and all blockchain participants have been registered with the CA.

4 Overview

In this section, we will give an overview of the architecture design and explain the operation of the transaction flow. Here, we refer to the transaction data ciphertext and keyword set ciphertext mentioned later as the transaction ciphertext.

In Cipherchain, blockchain entities can generally be divided into four parts {*Tx*, CA, *N*, *U*}, where *Tx* is the transactions data. The CA authorize operations such as registration, authorization, or cancellation of all nodes and users. *N* is a set of blockchain nodes performing different programs and tasks, while *U* represents the participating users who generate transaction proposals. To facilitate the following description, we collectively refer to the users and associated endorsement nodes as part of *N*, as the transaction participants.

The blockchain system is operated and maintained by participating nodes. The formation and transmission of transaction flow require these nodes with different tasks for full participation. In Cipherchain, blockchain nodes can be divided into the following three categories according to processing manner of transaction flow:

Endorsement node. These nodes are responsible for executing transaction proposals submitted by senders, verifying the validity of the transactions, and the core mission is to sign them for endorsement. That is, only transactions that meet the endorsement policies specified in the smart contracts will be considered valid. In addition, there is an endorsement node among partial nodes called proxy node, which is additionally responsible for collecting the transaction execution results and signatures of other endorsement nodes. Then it broadcasts valid transactions to consensus nodes.

Consensus node. The function performed by the consensus nodes is relatively not complicated, but it is a core factor affecting the transaction throughput of system. All nodes are required to collect newly received transaction ciphertext and run the same consensus mechanism.

Storage node. The storage nodes are responsible for receiving the block after consensus phase and maintaining full ledger data. Besides, they need to respond to the demand for transaction retrieval.

Physically, some types of nodes may belong to the same server. We will introduce later. The main process of transaction flow is illustrated in Fig. 2.

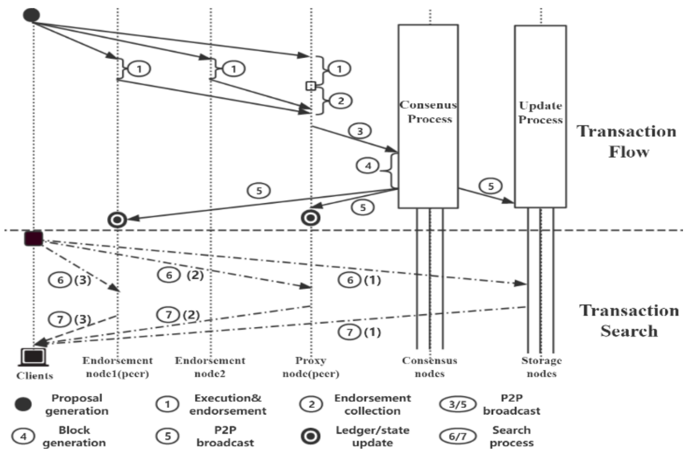


Figure 2: High-level transaction flow in Cipherchain

Different from the blockchain architecture of execute-order-validate implemented in Hyperledger Fabric, our proposed Cipherchain is more accurately described as execution-consensus-update architecture. That is, the distributed ledger update can be performed directly by storage nodes. Since we introduce the mPECK in Cipherchain, we briefly describe the blockchain architecture model with several polynomial time algorithms as follows:

1) ***Params***: Authorized participants registered with the CA will have their own key pairs for transaction encryption and decryption, as well as the public key certificate assigned by the CA. In addition, necessary system parameters are generated.

2) ***TpEndorse*****:** The sender generates a transaction proposal *Tp* and corresponding keyword set, broadcasts them to specific endorsement nodes. These nodes perform a series of operations on *Tp* when recognizing them as valid to generate a transaction *Tx*. Then they sign the simulation result for endorsement, which will be subsequently delivered along with their signatures to the proxy node. Each endorsement node has a local state database (mentioned later) that is used to record the state information of the associated transaction participants and the user contract.
3) ***mPECK+TxEnc*****:** When the valid signatures of endorsement nodes reach the required number, the proxy node introduces the public keys of the transaction participants, to perform mPECK operation on *Tx* and the keyword set, thus generate a transaction ciphertext.
4) ***Consensus*****:** The proxy node broadcasts transaction ciphertext together with the signatures to consensus nodes. After performing necessary validation, consensus nodes order and pack the received transactions into blocks, then participate in the consensus process of the whole blockchain network. In the consensus phase, these nodes do not need to know the transaction content, so there is no need to store the blockchain ledger. Each consensus node is built with a global state database that records the state information of overall endorsement nodes.
5) ***Update*****:** The valid block after consensus phase is delivered to storage nodes, which then perform the update operation on their respective local ledgers. The endorsement nodes involved and all consensus nodes need to update state databases.
6) ***Trapdoor*****:** When a member of transaction participants needs to extract a transaction stored in the blockchain ledger, it can use the private key of the corresponding transaction to act on the keyword set to generate the trapdoor and deliver it to the storage nodes for ciphertext retrieval.
7) ***Test*****:** The storage node executes the *Test* algorithm after Step 6 has been executed. Once the operation is successful, the transaction data ciphertext corresponding to the keyword set is then transmitted to the member; otherwise, the search failure message is returned.
8) ***TxDec*****:** After receiving the transaction data ciphertext, the authorized member decrypts the ciphertext with its own private key, and eventually obtains target transaction information.

Obviously, the first five algorithms above describe the formation of the transaction flow, including the whole process from the generation of a transaction to the eventual ledger update. The latter three algorithms introduce a transaction search method based on trapdoor query.

5 Architecture design

In this section, we will elaborate on how Cipherchain works according to the overview in Section 4, and explain the details of the execution-consensus-update architecture. Before doing this, we introduce the concept of smart contract and state database designed in Cipherchain, which are mentioned frequently below.

Smart Contract

The smart contract specifies the transaction logic and endorsement policy, which exists in the form of program code in the transaction. Generally, the transaction logic is the confidential data of business. Since the transaction exists in the form of ciphertext, this makes the smart contract more private and secure. In Cipherchain, smart contract can be categorized into user contract and system contract explained as follows:

User contract (USC). User contracts are developed by users and presented in the form of transactions. Before sending an ordinary transaction, the user must declare which user contract the transaction is associated with. The transaction execution result will be considered valid only if it conforms to the logic of user contracts. In addition, each user contract points to the system contract by default at the time of initial creation, parameterizing the function of the endorsement policy in the system contract, specifying the ID, quantity, and number threshold parameters, etc. of the endorsement nodes.

System contract (SSC). System contract is developed by the system administrator. It acts on all generated transactions in Cipherchain and affects the normal operation of the system and transaction flows. The main purpose of the system contract is to develop abstract endorsement policies. As can be seen from the above, each user contract must be associated with the system contract. If the parameterized endorsement policy in the user contract is not satisfied with the system contract requirements, the contract creation request will be rejected.

The system contract should have been in Cipherchain at the beginning of the system operation. Meanwhile, before users obtain blockchain service, they must have a valid user contract in the ledger. Of course, user contracts may also be updated, such as adjusting the endorsement policy, changing the transaction logic, etc., which requires the transaction participants to develop or verify the new contract.

State database

Each node in Ethereum has a dynamic global state trie in addition to the static blockchain structure. The state trie is specifically implemented using “the modified Merkle Patricia tree (trie)” [Matthew (2017)]. It records state data for all accounts in a key-value mode. The *nonce* field in the transaction serves as part of the account state and contribute to prevent “double spending” attack. This makes the traditional UTXO model replaced by the account model. Our solution is based on the account model as well.

The account state in Ethereum is globally consistent, that is, each node holds all the account information in the blockchain network. However, as the volume of transactions increases, the performance of Ethereum will also decline rapidly [Zhang, Jin and Cui (2018)]. In Cipherchain, a two-level state database scheme is proposed to conform to the modular design of the system. The lower level has local attributes, while the higher one is global consistent.

The local state database saved by specified endorsement nodes maintains the state information of associated transaction participants and the user contract. The term “specified” above refers to the nodes encoded in the user contract, which are generally only a subset of overall endorsement nodes. Moreover, an endorsement node can run multiple different user contracts, but a corresponding state database needs to be established for each user contract. Ordinary transactions drive the update of the low-level database, which should be synchronized between specified endorsement nodes.

Consensus nodes maintain the global state database. This globally consistent database maintains state information for all endorsement nodes in the blockchain network.

Compared with Ethereum, the design of two-level state greatly reduces the storage burden of blockchain nodes, and the construction is more scalable and flexible. When a user contract is revoked, the corresponding low-level state database can be removed from local memory. Besides, operations on user contracts such as invocation and inheritance are not described in detail here.

5.1 Execution phase

The execution phase is the core process of the Cipherchain, which consists of three algorithms: *Params, TpEndorse* and *mPECK+TxEnc*.

Algorithm 1. $\boldsymbol{Params\left(1^k\right)}$ **:** Given the security parameter 1^k , it returns parameter set $params=\left(G_1,G_2,e,H_1,H_2,g\right)$, where g is the generator G_1 . The hash functions $H_1=\{0,1\}^{log\omega}\to G_1$ $H_2=\{0,1\}^{log\omega}\to G_1$ are related to the generation of keyword set ciphertext and trapdoor. The sender and the n other participants respectively generate random value $s_0,s_1\ldots s_n\in Z_p^*$, and compute $t_i=g^{s_i}$, that is, generate respective public-private key pairs for data encryption and decryption.

$\left(pk_i,sk_i\right)=\left(t_i,s_i\right)$

It should be noted that among the n participants, there are not only the participating users, but also specified endorsement nodes.

Each transaction participant usually has at least two key pairs with different functions: one is allocated by CA for network access and identity authentication; the other is to encrypt or decrypt data, usually it can be generated by the transaction participant itself or by the CA. We identify the key pair of the former as:

$\left(pk_i^{ID},sk_i^{ID}\right)$

Write the latter as:

$\left(pk_i^{Enc},sk_i^{Enc}\right)$

Remark: Since the Cipherchain is a permissioned blockchain, it is necessary to provide PKI-based identity management and enforce transaction management. All users and nodes participating in the blockchain network need to be registered with the CA specified by the system. In addition, unlike previous blockchains, the keyword set field in Cipherchain's transactions is employed to facilitate ciphertext retrieval mentioned later.

Algorithm 2. $\boldsymbol{TpEndorse\left(Tp,USC\right)}$**:** Executed by endorsement nodes. Take as input the transaction proposal *Tp* and associated user contract, output the transaction simulation result *Tx* and specified endorsement nodes' signature set of Sig_{Tx}. The generated *Tx* will not be written to the blockchain ledger in this phase.

The sender first needs to generate the transaction proposal *Tp* and fill in the keyword set $W = \{w_1, \dots w_l\}$. Some necessary fields of a transaction proposal is shown in Tab. 1.

Table 1: The partial content of a transaction proposal

Transaction content	Mainly refers to the operation set of the transaction; if the transaction is to create a user contract, it includes the contract code.
Keyword set	The sender needs to fill in the fixed keyword set $W = \{w_1, \dots w_l\}$ corresponding to the transaction in advance.
Timestamp	Time when the transaction proposal was generated.
ID and signature	Sender's identity information registered in CA, and the signature of the proposal.
User contract address	The execution of the transaction relies on associated smart contracts. If the contract address is 0, it means that the transaction is to create a user contract.
Nonce	Identifies the transaction ID. The nonce value will be incremented by 1 each time after a transaction is executed. "User Contract Address+Nonce" constitutes the unique identifier of the transaction in the blockchain network.

Since each ordinary transaction is associated with a specific user contract, a valid user contract needs to be created before ordinary transactions. The sender formulates a user contract *Tp*, which must be broadcast to the endorsement nodes specified in the contract. This special contract also specifies a proxy node as one of these endorsement nodes. The sender just wait for the proxy node to inform it whether the contract transaction has been appended to the ledger.

We assume that the user contract has been updated into the ledger. Each *Tp* is also delivered to all specified endorsement nodes. They verify whether the timestamp is valid, the user's identity is authentic and whether there is permission to submit the transaction based on the ID and signature provided by the sender.

The endorsement nodes need to access the contract transaction ciphertext according to the contract address provided in *Tp*, and decrypts to obtain the contract content with its private key sk_i^{Enc}. As for how to obtain the ciphertext, we will explain it in Section 5.3. If the ciphertext cannot be obtained, or the decryption fails, it indicates that the contract address provided is invalid, or the endorsement nodes are not specified by the contract. Then the endorsement nodes will return an error message to the proxy node. Next, the endorsement nodes will check whether the keyword set meets the specification of the keyword format.

The *Tp* submitted will be executed respectively according to the transaction logic on each specified endorsement node. After the execution is completed, the endorsement nodes generate a well-formed transaction structure *Tx*, and sign the result, then deliver the signature information Sig_{Tx} to the proxy node. During the execution phase, the endorsement nodes should retain the context of the execution results and not update the account state until *Tx* is eventually appended to the ledger. Generally, the hashes of transaction structures received by the proxy node are the same, which indicates that the *Tp* generated by the sender is "correct". This "correct" indicates that the execution process and results of the code are unambiguous.

The formation of a user contract is not explained in detail above, so we briefly describe it here. The sender creates a user contract *Tp* and sends it to specified endorsement nodes, which will do the following additional work:

1) Check if the sender has permission to create a contract with a certain type.
2) Check whether the user contract meets the endorsement policies in the system contract.
3) Check the authenticity and authority of user contract creators' identities specified in the contract (optional).
4) Verify that the contract transaction logic satisfies the system contract specifications.

Here we will explain the Step 3 above. Due to the diversity of business types, the initiation of a user contract transaction may be actively performed by a single sender or multiple parties together. If the individual has permission to create an ordinary transaction, the endorsement nodes only need to verify the identity information of the sender. While multiple parties participate together, each endorsement node will wait for receiving all users' signatures.

A user contract is eventually considered valid only if overall endorsement nodes specified in the contract (including the proxy node) endorse the contract. Then each specified endorsement node creates an initial state database locally that maps to the contract.

Remark: If the smart contract in Cipherchain is written in a similar way to the Ethereum programming language, each proxy node always receive the same transaction execution results. While writing code in a more flexible programming language other than Solidity, this makes transaction logic more powerful and decentralized applications more scalable. However, programming languages like C/C++ or Java may bring result uncertainty, which in turn destroys the consistency of both the transaction results and the account state with respect to most blockchains. In Cipherchain, the proxy node is selected to play a critical role in transaction execution. It compare the results of asynchronous execution to obtain sufficient matching responses from fixed endorsement nodes. In the worst case, the user need to resubmits a new *Tp*. According to the description of Fabric's future solutions, it may introduce CRDTs [Shapiro, Preguiça, Baquero et al. (2011)] to enhance the liveness semantics under contention.

Algorithm 3. *TxEnc* (y_0, y_1 … y_n, *Tx*)+*Mpeck* (y_0, y_1, … y_n, *W*): Executed by the proxy node. Take as input the public keys pk_i^{Enc} of transaction participants, the transaction plaintext *Tx*, and the keyword set $W = \{w_1, \dots w_l\}$, and output the transaction ciphertext C_{Tx}. For simplicity, here y_i is used instead of pk_i^{Enc} in the certain equations later.

When the proxy node verifies the *Tx* structures and signatures set of Sig_{Tx} are valid, and the number or proportion of the same result meets the endorsement policy, the *Tx* is recognized as valid.

Here we assume that all specified endorsement nodes correctly execute the *Tp*; meanwhile, they deliver the execution results to the proxy node. A secure multi-receiver public key encryption scheme [ElGamal (1985)] is employed to generate transaction data ciphertext E_{Tx}. The proxy node selects two random values $r_1, r_2 \in Z_p^*$:

$$E_{Tx} = H(e(g,g)^{r_1 r_2}) \oplus Tx \tag{1}$$

Here $H(\cdot)$ is a collision-resistance hash function.

The generation process of the keyword set ciphertext E_W is:

$$\begin{gathered} A = g^{r_1}, B_j = y_j^{r_2}, C_i = h_i^{r_1} f_i^{r_2}, \\ 1 \le \mathrm{i} \le \ell, 0 \le j \le \mathrm{n} \end{gathered} \tag{2}$$

Where $h_i = H_1(w_i), f_i = H_2(w_i)$. Then the complete transaction ciphertext is presented below:

$$\begin{aligned} C_{Tx} &= (E_{Tx}, E_W), \\ E_w &= (A, B_0, \ldots B_n, C_1, \ldots C_l) \end{aligned} \tag{3}$$

Then, the data structure before and after the *Tx* is encrypted, is as follows:

$$\begin{aligned} T_x &= (Tx \,||\, W) = (Tx \,||\, w_1 \,||\, \ldots \,||\, w_l) \\ C_{Tx} &= (E_{Tx} \,||\, E_W) \\ &= (E_{Tx} \,||\, A \,||\, B_0 \,||\, \ldots \,||\, B_n \,||\, C_1 \,||\, \ldots \,||\, C_l) \end{aligned} \tag{4}$$

Subsequently, the proxy node broadcasts C_{Tx} and signatures to consensus nodes.

Ideally, the proxy node receives messages from all specified endorsement nodes, which create consistent execution results against *Tp*. However, the proxy node may not receive response from a minority of nodes (failure, offline, etc.), or the received results are wrong by comparison. The public keys pk_i^{Enc} of these nodes will not be used as input parameters for the algorithm 3.

In some cases, none of the execution results against *Tp* reaches a dominant ratio that is declared in the user contract; or the number of endorsement nodes does not reach the specified threshold. If so, the proxy node returns execution failure message to the sender, clears the execution context of *Tp*, and the local state database does not change.

Remark: Mostly, transaction data is generated by one initiator, while multiple participants share the result of transaction execution. Once transactions are created for recipients multiple times by employing traditional encryption technology, obviously the communication and storage overhead will increase linearly. This is unacceptable for blockchain systems where data capacity is severely limited. Thus in Cipherchain, the multi-receiver public key encryption scheme of ElGamal [ElGamal (1985)] type can process *Tx* in the context of multi-users. The cryptographic operations do not affect the overall performance of the blockchain system.

5.2 Consensus phase

In the consensus phase, since there is no need to execute the transaction logic, the core is to run a consensus algorithm to ensure the consistency of the blockchain ledger, which makes it possible for transactions to be presented in the form of ciphertext.

Algorithm 4. $\boldsymbol{Consensus(C_{Tx})}$ **:** Executed by consensus nodes. Take as input the transaction ciphertext (E_{Tx}, E_W) and signature set of Sig_{Tx}, output the block structure B_T. Proxy nodes in the blockchain network broadcast C_{Tx} to the consensus nodes.

The evaluation that the consensus nodes need to perform are relatively simple, namely:

1) Check if the C_{Tx} is within the limited capacity.

   Each transaction is created with a capacity limit, which ensures that a block with the same capacity cap can accommodate enough transactions. On other hand, a block containing too few transactions will significantly affect system performance. In Cipherchain, each C_{Tx} contains an additional keyword set ciphertext. Therefore, it is necessary to consider various factors such as consensus mechanism and network bandwidth to set these parameters.

2) Check if the timestamp of C_{Tx} is within a specific period.

   The proxy node attaches a timestamp to a transaction before sending it, indicating when the transaction was sent. There is also a timestamp in the transaction structure, but it indicates the generation time of Tp, not perceived by consensus nodes. The consensus nodes detect whether the timestamp attached satisfies within a specific period. If so, proceed to the next step.

3) Check if the identities of the endorsement nodes the transaction associated with are valid.

   The proxy node sends the identity information of all endorsement nodes that have performed endorsement operation. Then consensus nodes will check whether these digital identities are valid and whether they have the authority to endorse.

4) Check if the transaction hashes signed by all endorsement nodes are the same.

   Consensus nodes decrypt the signatures with the public keys pk_i^{ID} of endorsement nodes. If hashes of the decrypted transaction are all the same, this indicates that the results of the transaction execution are consistent.

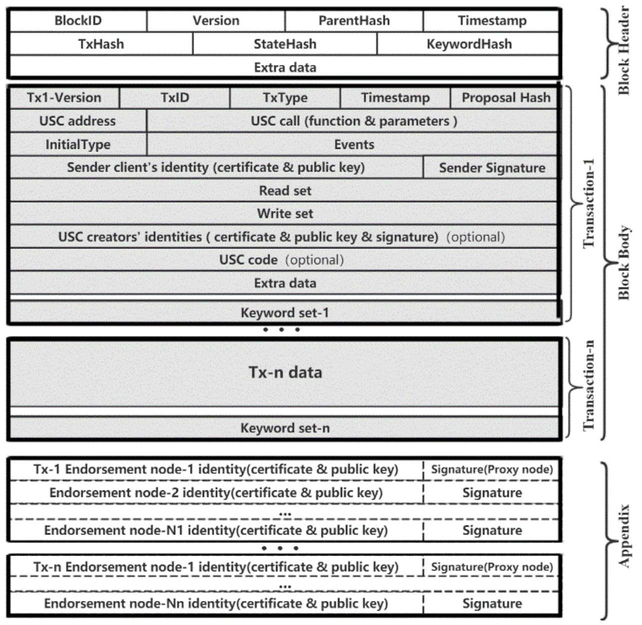
5) Check if there are duplicate transactions.

   Even if the first four steps are verified successfully, there is no guarantee that the transaction is valid. Consensus nodes may collect multiple duplicate transactions from a certain proxy node. Therefore, they need to identify duplicate transactions to prevent "double spending". The easiest way is to keep the transaction hashes processed within several specific rounds of consensus and to compare with each newly received transaction.

There will still be some special cases where the transaction is repeatedly appended to Cipherchain. The proxy nodes may mistakenly consider that a transaction has not been successfully updated to the Cipherchain for some reason. Therefore, they may resend a transaction with a new timestamp. This usually happens when consensus nodes have cleared local transaction history. To address these threats, the state information of all endorsement nodes in the blockchain network can be recorded by creating a global state database in the consensus nodes, and this measure does not reveal valuable information with respect to transactions and users. Then consensus nodes will then run a consensus algorithm against newly generated blocks.

Remark: Since the transactions are submitted to the consensus nodes in the form of ciphertext, consensus nodes do not participate in the verification and execution of the transaction content themselves. This is in line with the premise that the consensus nodes do not need to establish a trust model against transaction. In Cipherchain, we adopt a modular design approach; hence, the consensus module is pluggable. We can update the consensus module since it is independent of other processing modules of transaction flow. The system can adopt a variety of consensus mechanisms, such as PoW, POS, POA, PBFT, etc. Besides, this also greatly reduces the computation overhead, so that consensus nodes can focus on the operation of the consensus algorithm. In addition, Cipherchain as a permissioned blockchain can usually be run and maintained by one or more known service providers. Compared with public blockchain schemes, it can obtain better communication bandwidth and quality of service. Therefore, Cipherchain can achieve faster distributed consensus, which contributes to higher transaction throughput.

The eventual block structure, including *Tx* and Sig_{Tx}, generated after consensus phase is shown in Fig. 3.



BlockID
Version
ParentHash
Timestamp
TxHash
StateHash
KeywordHash
Extra data
Block Header
Tx1-Version
TxID
TxType
Timestamp
Proposal Hash
USC address
USC call (function & parameters)
InitialType
Events
Sender client's identity (certificate & public key)
Sender Signature
Read set
Write set
USC creators' identities (certificate & public key & signature) (optional)
USC code (optional)
Extra data
Keyword set-1
Transaction-1
Tx-n data
Keyword set-n
Transaction-n
Block Body
Tx-1 Endorsement node-1 identity(certificate & public key)
Signature(Proxy node)
Endorsement node-2 identity(certificate & public key)
Signature
...
Endorsement node-N1 identity(certificate & public key)
Signature
Tx-n Endorsement node-1 identity(certificate & public key)
Signature(Proxy node)
...
Endorsement node-Nn identity(certificate & public key)
Signature
Appendix

Figure 3: Data structure of block and transactions in Cipherchain

The block consists of three parts: block header, block body, and appendix. Only the block headers and appendix in the form of plaintext can be accessed. The shaded part consists of *Tx* and keyword set W, which are used as input to algorithm 3 to generate C_{Tx}. Some fields of the block structure are briefly explained below:

***Keyword Hash*:** All E_W in a block are hashed in a Merkle Tree [Merkle (1980)], with only the root called *KeywordHash* included in the block header.

USC address and ***USC call*:** If it is a user contract transaction, set the address to 0, call certain functions of the system contract, and *USC code* must be filled in; otherwise, this field specifies the user contract address associated with the transaction and calls the user contract function.

***InitialType*:** It is related to USC address. If a user contract transaction is to be created, the field indicates how the transaction is initiated. Moreover, the following *USC creators' identities* field must be filled in.

***Appendix*:** This part contains the signature set of endorsement nodes associated with each transaction. *Appendix* serves as an important way to verify the validity of each transaction for the consensus nodes in Section 5.2.

Obviously, the transaction structure of the shaded part is generated and encrypted by the proxy node in the execution phase. Each proxy node broadcasts encrypted transactions along with corresponding *Appendix* to the consensus nodes.

5.3 Update phase

In the update phase, the update operation on the global distributed ledger are to be performed; meanwhile the state databases maintained by certain endorsement nodes and all consensus nodes will be updated as well.

Algorithm 5. $\boldsymbol{Update(B_T)}$ **:** Executed by the storage nodes. Take as input a valid block B_T generated after the consensus phase, and output a series of update operations on the blockchain ledger and related state databases.

The endorsement node and the storage node are logically divided by function. They may belong to the same server physically, or each may run on different servers, hence the update operation may have some differences.

We first discuss the case where the endorsement node and the storage node are the same physical server. Here we refer to this type of physical node as peer. The consensus nodes send a B_T to all peers, which then append it to local ledgers after some necessary validation operations. Some peers may have submitted transactions in advance and then check if these transactions exists in the local ledger. If it exceeds a specific period (such as several rounds of consensus), and there is still no target transaction in the ledger, users will be returned commit failure message. In this case, the state databases do not need to be updated. While the update is successful, storage nodes must return commit success message to consensus nodes, then associated state databases of endorsement nodes and all consensus nodes' database are required to update.

Another situation is that the storage node and the endorsement node are physically different servers. In this case, the storage nodes only serve to maintain the blockchain. After consensus phase, consensus nodes send a B_T to storage nodes. At the same time, they will also broadcast this block to the endorsement nodes that have proposed transactions contained in this B_T in advance.

As mentioned earlier in the endorsement phase, some endorsement nodes may perform incorrect results or fail to endorse transactions. As an example, a simple endorsement policy can be described as operation below:

$Op: Endorsement(2, N1, N2, N3)$

That is, at least two signatures from the endorsement node *N1*, *N2* or *N3* are required. Here we assume that *N3* failed to participate in the endorsement process. The endorsement nodes need to ensure that their respective state databases are synchronized before each single *Tp* is executed. Therefore, *N3* needs to perform a state update as well. After the transaction is eventually in the ledger, the proxy node needs to inform *N3* to update the state. The endorsement nodes can share state information with each other through a keep-alive instruction. As a result, the newly joined nodes (for example, after updating user contract) can also be quickly updated to the latest state.

Remark: In Fabric, peers call VSCC (validation system chaincode) to perform endorsement policy evaluation on all transactions in a received block. If the endorsement fails to meet the requirements, the transaction will be marked as invalid. The VSCC specifically determines whether the endorsement of a transaction is from the intended source and whether the transaction has obtained the required number of signatures. While in Cipherchain, each user contract parameterizes the endorsement policies in the system contract, specifies the ID, quantity, and transaction logic of endorsement nodes. When a user sends an ordinary transaction, the endorsement nodes can determine whether the transaction logic and the endorsement policy are valid based on the local state database and the user contract. At the same time, the consensus nodes also verify the identities and permissions of these endorsement nodes based on the global state database. The function similar to VSCC is completed before the update phase. This design makes it impossible for the storage nodes to obtain any information about the transaction. In update phase, the storage nodes do not need to perform complicated computation related to transactions, which further modularizes the blockchain function and the storage nodes can exist independently. Another situation is when there are peers existing in Cipherchain. Obviously, each peer will just know related transactions, but not the information of other transaction.

5.4 Search phase

After the execution-consensus-update operation of transaction flow is performed, an immutable transaction history record is formed. In the search phase, since transaction information exists in ciphertext, special operations are usually required. Endorsement nodes are primarily intended to invoke a user contract to process a newly received Tp, or in some cases to search a historical transaction for users. Users intend to obtain the higher-level service of the blockchain system. According to the different schemes of transaction search, it can be divided into two modes: ordinary search and trapdoor query. The former only needs to find the transaction in the ledger according to the transaction address, while the latter needs to generate a trapdoor. Here we first explain the specific implementation of the trapdoor query.

Algorithm 6. $\boldsymbol{Trapdoor\left(sk_j^{Enc}, Idx\right)}$**:** Executed by the transaction searchers. Take as input the searcher's private key sk_j^{Enc} and the keyword set index Idx, and output the trapdoor T_j. The searcher selects a random value $r \in Z_p^*$, which is computed separately:

$$T_{j,1} = g^r, T_{j,2} = \left(h_{I_1} \dots h_{I_m}\right)^r, T_{j,3} = \left(f_{I_1} \dots f_{I_m}\right)^{r/sk_j^{Enc}} \tag{5}$$

where $Idx = \left\{I_1, \dots, I_m, w_{I_1}, \dots, w_{I_m}\right\}$ for $m \le l$. Here I_i indicates the position of the keyword w_{I_l} in the keyword set, outputs the transaction trapdoor:

$$T_j = \left(T_{j,1}, T_{j,2}, T_{j,3}, I_1, \dots, I_m\right) \tag{6}$$

The trapdoor is generated locally by the transaction participants and "passed" to the storage nodes. Here, "pass" is used instead of "send" because there is a case where the endorsement node and the storage node belong to the same server.

Algorithm 7. $\boldsymbol{Test\left(pk_j^{Enc}, E_w, T_j\right)}$**:** Executed by the storage nodes. Takes as input the keyword set ciphertext E_w, the public key pk_j^{Enc}, and the trapdoor T_j generated by a searcher. The output result "TRUE" indicates that the search is successful, and the transaction ciphertext information $E_{Test} = \left(E_{Tx} \parallel A \parallel B_j\right)$ is returned back to the searcher; output "FALSE" indicating that search failed, and failure message is returned. Specifically, the storage node performs the following query operations on C_{Tx}:

$$e\left(T_{j,1}, \prod_{i=1}^{m} C_{I_i}\right) = e\left(A, T_{j,2}\right) \cdot e\left(B_j, T_{j,3}\right) \tag{7}$$

If there is no E_W in the local leger satisfying this equation, the *Tx* is not in the ledger; otherwise, the searcher performs the next decryption operation.

Remark: Since each user contract must be endorsed by all associated endorsement nodes, all transaction participants can search the associated user contract transactions in a trapdoor query manner. However, ordinary transactions may not have involved all associated endorsement nodes participating in the endorsement work, some endorsement nodes may return search failure message. Thus in order to search more efficiently and accurately, it is generally possible to refer to the proxy node for related operations.

Algorithm 8. $\boldsymbol{TxDec\left(sk_j^{Enc}, E_{Test}\right)}$**:** Performed by transaction searcher for local ciphertext decryption. Take as input E_{Test} and the private key sk_j^{Enc} of the searcher, and output the result of performing an *XOR* operation on the E_{Tx}. Specifically, compute:

$$E_j' = h\left(e\left(A, B_j\right)^{1/sk_j^{Enc}}\right) \tag{8}$$

$$Tx^* = E_j' \oplus E_{Tx} \tag{9}$$

If the searcher is a user as a member of the transaction participants, or an endorsement node who has executed the corresponding *Tx* correctly, then the desired transaction plaintext $Tx = Tx^*$ is obtained. This is due to the equation:

$$E_j^{'} = h\left(e\left(A, B_j\right)^{1/sk_j^{Enc}} \right) = h\left(e\left(g, g\right)^{r_1 r_2} \right) \tag{10}$$

For the endorsement nodes that have not been involved in the correct execution of this *Tx*, obviously they cannot decrypt the ciphertext.

Earlier we mentioned the difference in physical implementation between storage nodes and endorsement nodes. This would also result in different search methods for participants.

First, we explain the case where the storage node and the endorsement node belong to the same physical server (i.e., peer). Since each peer maintains a blockchain ledger locally, it just processes the *Tp* according to the contract address. Then they decrypt it with sk_j^{Enc}.

Peers also need to respond to users' request to search for a target transaction. Each user can obtain the desired transaction in two ways. One is that the user adopts the above trapdoor query method, which requires both the user and the peer to spend a certain amount of computation overhead. Such a mission can be performed on any peer's server, even if the peer is not involved in the execution of the target transaction. This method does not cause any information leakage for users. The other is that the user specifies the transaction address, and then the peer returns specified transaction ciphertext. Obviously, if the user gets an unrelated transaction ciphertext, he cannot decrypt it.

When the storage node exists independently, endorsement nodes and users can only obtain the target transaction ciphertext through the trapdoor query method. The endorsement node generates a trapdoor based on the keyword set and sends it to a certain storage node. The storage node runs the *Test* algorithm, and returns the result. Similar to the endorsement node, a user can send a transaction request to the endorsement node as will.

6 Performance evaluation

We now evaluate the performance of Cipherchain scheme in terms of computation, storage and communication costs.

6.1 Computation cost

The computation cost here refers mainly to the cryptographic operations with respect to transactions. Since introducing mPECK technology, the transaction flow is inevitably affected by these complex cryptographic operations. Fortunately, since employing our improved modular design scheme that was not available in previous public blockchains, the impact of localized time-consuming operations on the overall performance of the system is limited. Computation overhead is mainly divided into two situations: the formation of transaction flow in local hosts or servers and trapdoor query in cloud servers.

The operations and the executing times of the formation of transaction flow used in the evaluation are defined in Tab. 2. The evaluations were performed on a personal computer (Lenovo with an I5-8300H 2.3 GHz processor, 8G bytes memory, 5400 rpm mechanical hard disk and Windows 10 operating system) using the JPBC library [De Caro and Iovino

(2011)]. In addition, our core code for cryptographic operations and performance evaluation was published on the website https://github.com/alanbaby/Cipherchain.

Table 2: The computation cost of cryptographic operations

Function	Description	User Number	Keyword Number	*Tx* (MB)	Time (*ms*)
Params	$(G_1, G_2, e, H_1, H_2, g)$		—		501
mPECK	Generation of keyword set ciphertext E_W	1	4	—	228
		2		—	234
		4		—	258
		8		—	287
		16		—	349
TxEnc	Generation of transaction ciphertext E_{Tx}	—		2	25
		—		4	35
		—		6	85
		—		8	102
T	Generation of a trapdoor T for Transaction retrieval	8	4	1	61
				2	96
				3	135
				4	178
T1				4	9
T2					80
T3					78
TxDec	Decryption of transaction ciphertext E_{Tx}	—		2	31
				4	41
				6	50
				8	59

The main computation cost of cloud servers is on the Test algorithm. The cryptography scheme requires three pairing operations per trapdoor. Since executing Test algorithm acting on each keyword set is independent of each other, and these three pairing operations per trapdoor can be executed independently, the scheme relies on the high parallel processing capability of cloud servers. In our performance evaluation experiment, the execution time of each *Test* thread related to (7) is about 28 milliseconds. Then we use thread pool technology to run on the PC processors, and finally we measure that it takes about 6 milliseconds each time to execute a *Test* thread.

In addition, it should be pointed out that it may return failure message if the trapdoor query method is used to obtain the target transaction. This means that cloud servers need to spend much time retrieving all keyword ciphertext, and even this behavior can become an attack to deteriorate the performance of cloud servers. Therefore, cloud servers need to take some measures to minimize the risk of retrieval failure, such as identity and privilege verification, the introduction of blacklist mechanism, and so on. Alternatively, the searcher can provide the search scope of ciphertext keywords and even provide a definite search target.

6.2 Communication and storage cost

Here we let $|G_1|$ and $|M|$ denote the bit sizes of point in group G_1 and transaction plaintext, respectively. Tab. 3 shows the communication and storage of Cipherchain using mPECK.

Table 3: Costs of communication and storage

Element	Size
pk_i^{Enc}	$\|G_1\|$
E_{Tx}	$\|M\|$
E_W	$(n+l+2)\|G_1\|$
T_j	$3\|G_1\|$

As shown in Tab. 3, the most significant feature is that the size of the transaction ciphertext does not increase when the transaction plaintext is encrypted. This is obviously the shortest ciphertext size compared with previous PECK schemes [Bösch, Hartel, Jonker et al. (2015)]. We know that the size of transactions in the blockchain structure has a significant impact on system performance. If the E_{Tx} generated by a certain encryption scheme is too large, it is bound to be impossible to realize the ciphertext blockchain scheme. Therefore, the mPECK, which can be implemented in multi-user scenarios, is very suitable for our proposed Cipherchain out of consideration of communication and storage overhead.

7 Application

7.1 Cipherchain and cloud computing

Currently, Bitcoin's distributed ledger has already consumed more than 200 GB. Since the ledgers exist in a chain structure, the data will only continue to increase. The amount of data in each distributed ledger will become larger and larger, so that ordinary users with limited storage capacity will not be able to store them.

At present, the application of the public blockchain is mainly in the field of cryptocurrency. However, future blockchain scheme may be faced with logic that is more complicated or larger amount of data in each transaction or block. Hence, blockchain with commercial applications requires address the issue of data explosion to support higher transaction throughput, which is also what storage nodes of Cipherchain need to face.

In our proposed scheme, we employed mPECK, a searchable encryption technology that addresses the privacy issue of cloud computing. The storage nodes can be physically as the cloud servers with huge computing power and storage capacity. The users or the endorsement nodes just request the cloud server to search the target transaction by employing the trapdoor query method. In this process, cloud service providers are unable to obtain any valuable information about transactions. Compared with the most blockchain schemes, Cipherchain involves relatively complex cryptography operations. However, the execution-consensus-update mode has limited impact on the overall system performance. The combination of "blockchain+cloud computing" is shown in Fig. 1 above, which may be a significant scheme to address many issues in the field of blockchain. The role of major cloud service providers in the blockchain network is similar to that of miners in Bitcoin, providing massive computing and storage services without additional cost of competition in computing power. This may trigger a new business model of cloud service.

Since the specific operation process of proposed "blockchain+cloud computing" is essentially consistent with the scheme described in Section 5, we will not elaborate on it here.

7.2 Cipherchain and public blockchain

In Cipherchain, the consensus module is highly independent. The consensus nodes do not participate in the verification and execution process of the transaction flow and their mission is simplified to run the consensus mechanism. The modularization of system brings significant advantages in performance and updateability. In the consensus phase, the transaction is presented in the form of ciphertext. Except for the users themselves and associated endorsement nodes, no entity can obtain the transaction content. For the above features, Cipherchain, as a type of permissioned blockchain, can even "outsource" consensus function to the existing public blockchains.

In the case of Bitcoin, the miners run the PoW mechanism. We know that most existing blockchains adopt PoW mechanism, which currently accounts for more than 90% of the total market capitalization of existing digital currencies [Gervais, Karame, Wüst et al. (2016)]. At the time of writing this article, the hashrate representing Bitcoin's total network computing power is 40 EH/S, while the Ethereum's has also reached more than 140 TH/S. The enormous computing power of the public blockchain maintains the normal operation of the decentralized system, and its transactions or blocks after consensus can hardly be tampered. Although Cipherchain can use the non-incentive consensus algorithms such as PBFT or Raft, we assume that the consensus mechanism adopted by the public blockchain (here specifically, PoW) can serve this permissioned blockchain. Such design can take the advantages of both the permissioned blockchain's and the public blockchain's respective features. The operations (verification, execution, and endorsement) related to the transaction flow itself during the execution phase are performed by the endorsement nodes. Moreover, the work in the consensus and update phases are handed over to the miners in the public blockchain network.

In this proposal, each miner needs to establish a global state database to maintain state information for all endorsement nodes. Naturally, it requires necessary interaction between the miners and the CA. The miners receive the well-formed transaction ciphertext, since the correctness of the transaction execution result is guaranteed by the endorsement nodes, which bear the corresponding responsibility. The main operations that miners need to perform are: (1) verifying the signature of the sender (a proxy node) on the transaction ciphertext; and (2) verifying the account states of all participating endorsement nodes. The verification operations that miners need to perform are similar to the consensus nodes in Section 5.2.

After the transaction ciphertext is considered valid, the miners' consensus process is similar to the Bitcoin, implementing the PoW consensus mechanism. Finally, the block produced by a miner is accepted by the whole public blockchain network, and the block containing multiple transaction ciphertexts is appended to the ledger. Therefore, each miner also plays the role of a storage node. When transaction participants search for specific transactions on miners' local ledgers, they usually need to pay some fees for additional computation overhead. Since public blockchains usually require incentive mechanism to inspire miners, we will not go into details here.

8 Conclusion

We present Cipherchain, the first ciphertext blockchain scheme, where data can be processed and maintained in the form of ciphertext to realize the privacy protection of transaction. We describe in detail the several algorithms related to the formation of transaction flows and retrieval methods for ciphertext transactions. Since introducing the execution-consensus-update paradigm of transaction flow, transactions are simulated and encrypted based on the mPECK scheme by specified endorsement nodes, while consensus and update operations with respect to ciphertext transaction are globally executed. Such a modular design also makes the overall performance of Cipherchain not greatly affected by cryptographic operations and other necessary computation work, which makes the system performance run efficiently. Then necessary performance evaluation is performed to demonstrate the feasibility of the cryptographic operations acting on Cipherchain. We also make it possible to implement the future application of technology combination of "blockchain +cloud computing" and "Cipherchain+public blockchain".

Our work does not only describe a novel ciphertext blockchain scheme designed for the sake of both privacy and efficiency, but also indicates that both the security model and the design concept "openness and transparency" based on previous Bitcoin-like architecture can be retained in permissioned blockchains, while without the compromise of performance or privacy.

Acknowledgement: This work is supported by the NSFC (Grant Nos. 61671087, 61962009, 61003287), the Fok Ying Tong Education Foundation (Grant No. 131067), the Major Scientific and Technological Special Project of Guizhou Province (Grant No. 20183001), the Foundation of State Key Laboratory of Public Big Data (Grant No. 2018BDKFJJ018), CCF-Tencent Open Fund WeBank Special Funding (CCF-WebankRAGR20180104), the High-quality and Cutting-edge Disciplines Construction Project for Universities in Beijing (Internet Information, Communication University of China), the Fundamental Research Funds for the Central Universities, and the Fundamental Research Funds for the Central Universities No. 2019XD-A02.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K. et al. (2018): Hyperledger fabric: a distributed operating system for permissioned blockchains. *Proceedings of the Thirteenth EuroSys Conference*, pp. 30.

Baudron, O.; Pointcheval, D.; Stern, J. (2000): Extended notions of security for multicast public key cryptosystems. *International Colloquium on Automata, Languages, and Programming*, pp. 499-511.

Boneh, D.; Di Crescenzo, G.; Ostrovsky, R.; Persiano, G. (2004): Public key encryption with keyword search. *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 506-522.

Boneh, D.; Waters, B. (2007): Conjunctive, subset, and range queries on encrypted data. *Theory of Cryptography Conference*, pp. 535-554.

Bonneau, J.; Narayanan, A.; Miller, A.; Clark, J.; Kroll, J. A. et al. (2014): Mixcoin: anonymity for bitcoin with accountable mixes. *International Conference on Financial Cryptography and Data Security*, pp. 486-504.

Bösch, C.; Hartel, P.; Jonker, W.; Peter, A. (2015): A survey of provably secure searchable encryption. *ACM Computing Surveys (CSUR)*, vol. 47, no. 2 pp. 18.

Buterin, V. (2014): A next-generation smart contract and decentralized application platform. *White Paper*, vol. 3, pp. 37.

Chaum, D. (1983). Blind signatures for untraceable payments. *Advances in Cryptology*, pp. 199-203.

Curtmola, R.; Garay, J.; Kamara, S.; Ostrovsky, R. (2011): Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, vol. 19, no. 5, pp. 895-934.

De Caro, A.; Iovino, V. (2011): jPBC: java pairing based cryptography. *IEEE Symposium on Computers and Communications*, pp. 850-855.

Duffield, E.; Diaz, D. (2015): Dash: a privacycentric cryptocurrency. *Self-published*.

ElGamal, T. (1985): A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469-472.

Garay, J.; Kiayias, A.; Leonardos, N. (2015): The bitcoin backbone protocol: analysis and applications. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 281-310.

Gervais, A.; Karame, G. O.; Wüst, K.; Glykantzis, V.; Ritzdorf, H. et al. (2016): On the security and performance of proof of work blockchains. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3-16.

Golle, P.; Staddon, J.; Waters, B. (2004): Secure conjunctive keyword search over encrypted data. *International Conference on Applied Cryptography and Network Security*, pp. 31-45.

Green, M.; Miers, I. (2017): Bolt: anonymous payment channels for decentralized currencies. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 473-489.

Hearn, M. (2016). Corda: a distributed ledger. *Corda Technical White Paper*.

Heilman, E.; Alshenibr, L.; Baldimtsi, F.; Scafuro, A.; Goldberg, S. (2017): Tumblebit: an untrusted bitcoin-compatible anonymous payment hub. *Network and Distributed System Security Symposium*.

Hwang, Y. H.; Lee, P. J. (2007): Public key encryption with conjunctive keyword search and its extension to a multi-user system. *International Conference on Pairing-Based Cryptography*, pp. 2-22.

Karame, G. O.; Androulaki, E. (2016): *Bitcoin and Blockchain Security*. Artech House.

Koshy, P.; Koshy, D.; McDaniel, P. (2014): An analysis of anonymity in bitcoin using p2p network traffic. *International Conference on Financial Cryptography and Data Security*, pp. 469-485.

Kurosawa, K. (2002): Multi-recipient public-key encryption with shortened ciphertext. *International Workshop on Public Key Cryptography*, pp. 48-63.

Lamport, L. (1978): Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, vol. 21, no. 7, pp. 558-565.

Lamport, L.; Shostak, R.; Pease, M. (1982): The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382-401.

Matthew, S. (2017): Merkle patricia trie specification. *Ethereum, October*.

Meiklejohn, S.; Pomarole, M.; Jordan, G.; Levchenko, K.; McCoy, D. et al (2013): A fistful of bitcoins: characterizing payments among men with no names. *Proceedings of the 2013 Conference on Internet Measurement Conference*, pp. 127-140.

Merkle, R. C. (1980): Protocols for public key cryptosystems. *1980 IEEE Symposium on Security and Privacy*, pp. 122-122.

Miers, I.; Garman, C.; Green, M.; Rubin, A. D. (2013): Zerocoin: anonymous distributed e-cash from bitcoin. *IEEE Symposium on Security and Privacy*, pp. 397-411.

Miller, A.; LaViola Jr, J. J. (2014): Anonymous byzantine consensus from moderately-hard puzzles: a model for bitcoin. *Available online: http://nakamotoinstitute.org/research/anonymous-byzantine-consensus*.

Nakamoto, S. (2008): Bitcoin: a peer-to-peer electronic cash system.

Park, D. J.; Kim, K.; Lee, P. J. (2004): Public key encryption with conjunctive field keyword search. *International Workshop on Information Security Applications*, pp. 73-86.

Reid, F.; Harrigan, M. (2013): An analysis of anonymity in the bitcoin system. *Security and Privacy in Social Networks*, pp. 197-223.

Rivest, R. L. (1997): Electronic lottery tickets as micropayments. *International Conference on Financial Cryptography*, pp. 307-314.

Sasson, E. B.; Chiesa, A.; Garman, C.; Green, M.; Miers, I. et al. (2014): Zerocash: decentralized anonymous payments from bitcoin. *IEEE Symposium on Security and Privacy*, pp. 459-474.

Shapiro, M.; Preguiça, N.; Baquero, C.; Zawirski, M. (2011): Conflict-free replicated data types. *Symposium on Self-Stabilizing Systems*, pp. 386-400.

Song, D. X.; Wagner, D.; Perrig, A. (2000): Practical techniques for searches on encrypted data. *Proceeding 2000 IEEE Symposium on Security and Privacy*, pp. 44-55.

Vukolić, M. (2017): Rethinking permissioned blockchains. *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, pp. 3-7.

Wood, G. (2014): Ethereum: a secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1-32.

Yang, B.; Garcia-Molina, H. (2003): PPay: micropayments for peer-to-peer systems. *Proceedings of the 10th ACM conference on Computer and Communications Security*, pp. 300-310.

Zhang, H.; Jin, C.; Cui, H. (2018): A method to predict the performance and storage of executing contract for ethereum consortium-blockchain. *International Conference on Blockchain*, pp. 63-74.