IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2019.Doi Number

Blockchain-based fair payment protocol for deduplication cloud storage system

SHANGPING WANG[1], YUYING WANG[2], and YALING ZHANG[3]

[1]School of Science, Xi'an University of Technology, Xi'an, Shaanxi, China(e-mail: spwang@mail.xaut.edu.cn)
[2]School of Computer Science and Engineering, Xi'an University of Technology, Xi'an, Shaanxi, China(e-mail: yuyingwang1110@gmail.com)
[3]School of Computer Science and Engineering, Xi'an University of Technology, Xi'an, Shaanxi, China(e-mail: ylzhang@xaut.edu.cn)

Corresponding author: Yuying Wang (e-mail: yuyingwang1110@gmail.com).

This work is supported by the National Natural Science Foundation of China under Grants No. 61572019.

ABSTRACT Today more and more enterprises and individuals are outsourcing their data to cloud storage system. Data deduplication is one of the important technologies to reduce the storage cost of cloud storage system. In a cloud storage system with deduplication technology, the client can outsource the data files to the cloud storage server and pay for them. Fair payment is one of key issues in the cloud deduplication storage system. At present, a variety of secure deduplication encryption schemes have been designed to protect the privacy of client data. However, most existing fair payment solutions use traditional electronic cash systems to generate payment tokens, which requires a trusted authority to prevent double-spending. Trusted authorities will become bottlenecks in the payment system. Faced with this problem, in this paper, we propose a new decentralized fair payment protocol for cloud deduplication storage system by utilizing ethereum blockchain technology. The new protocol takes advantage of the decentralization of blockchain technology, allowing direct transactions without the participation of trusted third parties. In the new protocol, if a malicious situation occurs, the system can guarantee fair payment by pre-storing penalty money in the smart contract. Safety analysis and experimental analysis show that our new protocol is feasible.

INDEX TERMS cloud storage, deduplication, fairness, blockchain, ethereum, smart contract

I. INTRODUCTION

In recent years, due to the rapid development of cloud computing and big data technology, more and more enterprises and individuals choose to outsource data to cloud service providers. Many cloud storage systems use deduplication to reduce costs by taking advantage of the redundancy of storing data and avoiding storing the same data multiple times in real life. However, in order to provide secure deduplication, clients must trust the server not only to store their documents, but also to encrypt them. Usually, traditional encryption techniques make deduplication impossible. Suppose Alice and Bob both have a file M, and they encrypt the file under the key KA and KB respectively. Finally, the ciphertext FA and FB are stored on the remote cloud storage server S. In this case, it is difficult for S to detect that the two ciphertexts are the same. In addition, even if it could be detected, it would be difficult for S to store FA and FB in a short copy, thus allowing Alice and Bob to decrypt the plaintext M. The convergent encryption proposed by Douceur J R et al. [1] and its variants deal with data security and privacy issues for secure deduplication. However, most of the existing fair payment schemes in cloud outsourcing storage adopt traditional payment mechanisms and rely on trusted third parties, such as Banks. For example, Google's cloud platform offers a range of computing and storage services, but bank accounts are required to register. However, traditional payment schemes have disadvantages. First, the bank is trusted by all users and servers and handles all processes in a fair way. Second, the payment mechanism needs to adapt to multiple Banks used by different participants and need to be updated at any time, which will become the bottleneck of the payment system. Finally, users' privacy and bank accounts may be violated. Therefore, the fair payment in the cloud service environment has been studied extensively.

In this paper, based on the cloud storage encryption scheme, we improve the cloud storage encryption scheme by introducing ethereum blockchain technology, and make fair

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2939492, IEEE Access

payment by using ethereum smart contract technology. At the same time, we achieve the role of supervision, track the behavior of data, and realize the decentralized fair payment. Because all access records are recorded in the blockchain network, and the blockchain is transparent. Our protocol uses deduplication technology to reduce the cost of cloud storage and to reduce server-side load.

Our contributions:

The contributions of this paper are as follows:

(1) A fair payment protocol based on Ethereum blockchain for cloud storage system is proposed. Decentralized payment is realized through the blockchain technology, and the fairness of payment is guaranteed by the smart contract with pre-existing penalty. There is no trusted third party in our system. The payment process is decentralized by transferring tokens in the blockchain network through ethereum smart contract technology.

(2) On the one hand, deduplication technology is adopted to provide effective and secure methods to reduce storage, communication and computing overhead of cloud storage servers and clients; on the other hand, according to "whether files uploaded by the client are duplicated with existing files on the cloud storage server" in deduplication technology, the protocol provides two different payables, namely, "file duplication", payable is b_1; "file unduplicated", payable is b_2,where $b_1 < b_2$.

(3) The combination of the payment scheme proposed in this paper and deduplication technology weakens the client-side deduplication attack to some extent. Usually if an attacker who knows the file tag *Tag* can convince the storage service that it owns the file, so the server allows the attacker to download the entire file. However, our payment protocol requires that the server does not perform file storage operations until the client pays. That is, even if the attacker has a file tag, he still needs to make a payment, and then the server returns file link pointer *L*. Therefore, this will limit the attack behavior of deduplication attack to a certain extent.

(4) Under the Ubuntu linux system, smart contracts were created and deployed through the Ethereum official test network Rinkeby, and the corresponding performance and cost were analyzed.

The rest of the paper is organized as follows. In the section II, related work is presented. The section III introduces some preliminaries. The section IV shows the system model of our scheme. The specific construction of our scheme is described in detail in section V. And the performance and security analysis are discussed in section VI. Finally, the conclusions and future research directions are given.

II. RELATED WORK

In order to achieve secure deduplication, some scholars propose the following schemes. In 2002, Douceur J R et al. [1] proposed the convergence of encryption, so that duplicate files are merged into one file space, even though these files are encrypted by different users' keys. That is, convergent encryption provides the first clever solution for secure deduplication. In 2013, Li J et al. [2] proposed the DeKey scheme to solve the problem of effective and reliable management of large convergence keys in convergent encryption. In this scenario, users do not need to manage any keys themselves, but instead securely distribute aggregate key sharing across multiple servers. In the same year, Bellare M et al. [3] formalized a new cryptographic primitive called message-locked Encryption (MLE), in which the key that performs encryption and decryption itself comes from the message. MLE provides a way to implement secure deduplication. Abadi M et al. [5] strengthened the security concept proposed by Bellare M et al. by considering the plaintext distribution that might depend on the common parameters of the scheme. Abadi M et al. designed a completely random scheme, which supports an equality testing algorithm defined on ciphertext. A deterministic ciphertext component is constructed to support more efficient equality testing. In both schemes, the overhead of ciphertext length is additional and independent of message length.

In 2013, Keelveedhi S et al. [6] addressed the vulnerability of message-locked Encryption (MLE) to brute force attacks, which can restore files to a known set, by proposing a framework that provides secure deduplication storage to resist brute force attacks, and implemented in a system called DupLESS. In DupLESS, the client encrypts under message-based keys that are obtained from the key server via an unrelated PRF protocol. It enables clients to store encrypted data with existing services, allowing services to deduplicate data on their behalf, while maintaining strong confidentiality guarantees. DupLESS demonstrates that encryption for duplicates can achieve performance and space savings similar to those achieved by using a plaintext data storage service. These secure deduplication solutions provide effective and secure ways to reduce storage, communication, and computing overhead for cloud storage servers and clients.

After deduplication encryption technology has been effectively solved, the issue of fair payment between clients and cloud storage servers has also attracted wide attention. Carbunar B et al. [7] first considered the payment problem in outsourcing computing in 2010. Based on this, they proposed a fair payment scheme based on the segmentation selection protocol and secret sharing protocol. However, this solution is very inefficient for practical applications. Later, the author proposed an improved new payment scheme [8], but the efficiency of this scheme was not improved. It can also be regarded as a specific example of conditional electronic payment [9-10]. In 2012, Chen et al. [11] first considered the third trust issue, that is, worker W would not send the calculation results to outsourcing company O. They adopted the same model, introducing lazy and partial dishonest employees. Then, they proposed a new fair payment scheme, which only uses the traditional e-cash scheme to generate payment tokens. Their solution is more effective than the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2939492, IEEE Access

previous one. In 2014, Chen et al. [12] further proposed a conditional electronic payment system based on restricted partial blind signature scheme.

Since the first introduction of blockchain by the Nakamoto team [13] in 2008, the application of blockchain-based technology has penetrated into various industries, especially those business fields where there are transaction intermediaries, which means that many service businesses will be decentralized. The decentralized distributed structure of blockchain can save a lot of intermediary costs in reality. Because blockchain technology can be used as a tool for large-scale collaboration between people without mutual trust, it can be used in many traditional centralized fields to handle transactions that were originally handled by intermediaries.

In 2014, Andrychowicz M et al. [14] showed how to use the bitcoin system to obtain fairness in any two-party security computing protocol. The significance is as follows: if one party terminates the protocol after receiving the output, the other party will receive economic compensation (bitcoin). One possible application of such an agreement is fair contract signing: each party is forced to complete the agreement or pay a fine to the other. In 2016, Andrychowicz M et al. [15] used the bitcoin system to provide an attractive way to construct a "timed commitments" in which the committer has to reveal their secrets within a certain time frame or to pay a fine. This, in turn, can be used to achieve fairness in some multi-party protocols. Secondly, they introduce the concept of the multi-party protocols that work "directly on Bitcoin". In 2017, Ateniese G et al. [16] introduced auditable storage based on reversible Bloom filter expansion, and demonstrated how to combine it with zero-knowledge proof based on bitcoin. However, the combination involves a trusted third party, called a bitcoin arbitrator. In the same year, Campanelli M et al. [17] defined the concept of zero-knowledge contingent service payment and realized the service payment based on blockchain. Two advanced protocols are constructed and implemented based on the retrievable service proof. However, the proposed protocol is only conceptual, lacking design details, and its efficiency needs to be improved due to the use of an indistinguishable protocol [23] as a building block.

Based on game theory and Ethereum smart contract, Dong C et al. [18] proposed a protocol to verify the correctness of computation in cloud computing. However, assuming that users are honest; the two clouds cannot collude. On the other hand, in order to improve the transaction throughput and latency in blockchain, the current work mainly focuses on offline payment channels, which can be combined with the payment channel network to achieve multiple payments without accessing the blockchain. In 2018, Zhang Y et al. [19] introduced TKSE, a trusted keyword search scheme based on encrypted data, without any third party. In TKSE, an encrypted data index based on digital signatures allows users to search for outsourced encrypted data and check that the search results returned by the cloud meet the pre-specified search requirements. In particular, it is the first time that server-side verifiability has been implemented to protect honest cloud servers from malicious data owners in the data storage phase. In addition, using the blockchain technique and the hash function, even if the user or the cloud itself is malicious, the payment fairness of the search fee can be realized without introducing a third party. In the same year, Zhao Yanqi et al. [29] used blockchain trading technology to realize the decentralized fair payment of the "publish-subscribe" system.

In summary, the use of blockchain technology's decentralized, non-tamperable features into the payment solution can solve the malicious problems of both sides. However, blockchain technology has just emerged, and the fair payment based on blockchain for cloud storage has not yet begun. It is of great value and significance to study the decentralized fair payment based on blockchain under cloud services.

III. PRELIMINARIES

A. Secure deduplication

Cloud computing provides a low-cost, scalable, location-independent infrastructure for data management and storage. The rapid adoption of cloud services has led to an increase in the amount of data stored on remote servers, requiring technologies that save disk space and network bandwidth.

Deduplication [3-4, 22] is an important technology to reduce the storage cost of cloud storage and management system, that is, the server only stores one copy of each file, regardless of how many clients request to store the file. Many storage systems use deduplication to reduce costs. Take advantage of the redundancy of stored data to avoid storing the same data multiple times in real life. For example, in a cloud storage system, suppose *n* clients share the same copy of file F. If some actual storage costs are omitted, then deduplication will change the storage cost of the file from $O(\mathrm{n}\cdot|F|)$ to $O(\mathrm{n}+|F|)$, $|\cdot|$ is the bit length of the file.

In a typical cloud storage system with deduplication, the client first sends only a hash of the file to the server, which checks to see if the hash value already exists in its database. If the hash is not in the database, the server requests the entire file. Otherwise, since the file already exists on the server (possibly uploaded by someone else), it tells the client not to send the file itself, saving bandwidth and storage (this is called client-side deduplication). Either way, the server marks the client as the owner of the file, and since then, there is no difference between the client and the original party that uploaded the file. Therefore, the client can request recovery of the file regardless of whether the client is required to upload the file. Either way, if the client needs this duplicate data, it will be charged. Although the server does not require the client to upload this duplicate file, the client still needs the server to return the file storage location. That is, the client

completes a stored procedure, but the server has the same file and does not need to upload it.

It has been reported that business applications can achieve deduplication rates from 1:10 to 1:500, resulting in savings of more than 90% of disk and bandwidth. Deduplication can be applied at the file level or at the block level, and file level deduplication is used in this paper.

During deduplication, the client tries to identify deduplication opportunities that already exist on the client to save bandwidth in uploading existing copies of files to the server. However, in order to provide secure deduplication, clients must trust the server not only to store their documents, but also to encrypt them. Usually, the traditional encryption technology makes deduplication impossible. Suppose Alice and Bob both have a file M, and they encrypt the file under the key KA and KB respectively. Finally, the ciphertext FA and FB are stored on the remote cloud storage server S. In this case, it is difficult for S to detect that the two ciphertexts are the same. In addition, even if it could be detected, it would be difficult for S to store FA and FB in a short copy, thus allowing Alice and Bob to decrypt the plaintext M. Convergent encryption [1] provides the first clever solution to secure deduplication and its variants, designed to address data security and privacy issues. At present, a variety of secure deduplication encryption schemes have been designed to protect the privacy of customer data. This paper uses convergent encryption to ensure data security.

B. convergent encryption

Convergent encryption (CE) [3] aims to provide data confidentiality for deduplication. Here, the client derives a convergent key *K* from each original data copy *M*, and uses *K* to encrypt the data copy to get ciphertext *C*. In addition, the client also derives a tag *Tag* for the data copy, which is used to detect the copy. Here, we assume that tag correctness attribute [41] is true. More precisely, if two copies of data are the same, they have the same tag. To detect a replica, the user first sends a tag to the server to check that the same replica has been stored. Note that the convergence key and the tag are independently derived, and the tag cannot be used to derive the convergent key and destroy data confidentiality. The encrypted copy of the data and its corresponding tag are stored on the server-side. Formally, according to the definition of [3], the following is the definition of the convergent encryption scheme used in the system.

A convergent encryption scheme (CE) is composed of a four-tuple (KG, Enc, Dec, TG).

$KG(M) \to K$. *KG* is an important generation algorithm, input data *M*, and output convergent key *K*.

$Enc(K, M) \to C$. *Enc* is a symmetric encryption algorithm that outputs ciphertext *C* with *K* and *M* as inputs.

$Dec(K, C) \to M$. *Dec* is a decryption algorithm that outputs a copy of the original data *M* with *K* and *C* as inputs.

$TG(M) \to Tag(M)$. *TG* is a tag generation algorithm for the original data copy *M*. (Normally, the input to the *TG* is the ciphertext of *M*, so $TG(C) \to Tag(M)$). In this paper, the file ciphertext *C* is hashed using the hash function SHA-256 to generate a file tag *Tag* to achieve tag unification.

C. signature scheme

In the digital signature scheme, the sender first publishes its public key *pk*, and then uses its private key *sk* to sign the message. When a signed message is received by the receiver, the sender's public key can be used to verify the message.

Digital signature scheme is a set of probability polynomial time algorithm (*Gen*, *Sign*, *Verify*), so that:

$Gen(k) \to (pk, sk)$.The key generation algorithm *Gen* takes the security parameter *k* as input and outputs a pair of keys *(pk,sk)*, which are called public key and private key respectively.

$Sign(sk, m) \to \sigma$ .The signature algorithm *Sign* accepts the private key *sk* and the message *m* from the message space as input. It outputs a signature σ.

$V\text{erify}(pk, m, \sigma) \to b$ .The deterministic verification algorithm *Verify* enters the public key *pk*, message *m*, and signature σ. It outputs a bit *b*; *b* = 1 means valid, *b*= 0 means invalid.

It is required that for each *k*, every *(pk,sk)* output of *Gen(k)*, and each message *m* in the plaintext space, the signature of the message *m* satisfies $V\text{erify}(pk, m, \sigma) = 1$.

D. Proof of Ownership

Cloud storage systems are gaining popularity. One promising technique to reduce the cost of deduplication is to store only one copy of the duplicates. Client deduplication attempts to identify existing deduplication opportunities on the client and to save bandwidth by uploading existing copies of files to the server.

Harnik et al. recently found that client-side deduplication introduces new security issues [43]. For example, the server tells the client that it does not need to send the file, which indicates that some other clients have exactly the same file, which may be sensitive information. Specifically, an attacker who knows the file tag *Tag* can convince the storage service that it owns the file, so the server allows the attacker to download the entire file.

To overcome such attacks, [4] introduces the concept of proof of ownership (POW), which allows the client to effectively prove to the server that the client holds a file, rather than just some brief information about the file.

The concept of POW can be achieved by using a Merkle tree-based retrievable protocol proof. That is, we first use the Erasure Code to encode the file so that we can recover the entire file from, for example, 90% of the encoded bits. Then, we build a Merkle tree on the encoded files, and lets the server asks a random selection of super-logarithmic leaves.

According to the erasure code properties, if the enemy is missing any part of the file, then at least 10% of the leaves are not aware of it. Moreover, if the file has a high min-entropy [4] from the point of view of the adversary, it cannot

even guess the value of the 10% leaf, and there is no obvious chance of success. Therefore, it is very likely to be caught. Detailed POW scheme description references [4].

1. Basic knowledge

(1) $E:\{0,1\}^M \to \{0,1\}^{M'}$ is an erasure code [4] that can erasure up to a α portion of the bit (for a certain constant $\alpha > 0$). Namely, the file $F \in \{0,1\}^n$ is erasured with the security parameters n. From any $(1-\alpha)M'$ position of $E(F)$, in principle, the original file $F \in \{0,1\}^n$ can be completely recovered.

(2) The Merkle tree provides a clean promise for large buffers [4], so that later blocks of the buffer can be opened and validated without providing the entire buffer. To construct the Merkle tree, we split the input buffer into blocks, then group the blocks in pairs, and hash each pair using a collision resistant hash function. The hash values are then grouped in pairs again, and each pair is further hashed, repeating the process until only one hash value remains. This generates a binary tree with the leaves corresponding to the blocks of the input buffer and the roots corresponding to the final remaining hash values. (When the number of blocks in the input buffer is 2^h, the resulting tree is a complete binary tree of height h.)

Use $MT_{H,b}(X)$ to represent the binary Merkle tree of buffer X, using the b-bit leaf node and the hash function H. In addition, let H be a collision resistant hash function with an output length of n' bits (e.g., SHA256, $n' = 256$). For each node n in $MT_{H,b}(X)$, we use v_n to represent the value associated with that node. That is, the value of the leaf node is the corresponding block of the buffer X, and the value of the intermediate node $n \in MT_{H,b}(X)$ is the hash $v_n = h(v_l, v_r)$, v_l, v_r are the values of the left child node and the right child node of n, respectively. (If a child of a node is missing from the tree, its value is treated as null.)

For a leaf node $l \in MT_{H,b}(X)$, the sibling path of l consists of the value v_l and the sibling path values of all the nodes in the path from l to the root.

Given the subscript of leaf node $l \in MT_{H,b}(X)$ and the sibling path of l, we can calculate the values of all the leaves on the l-root path in a bottom-up manner, by starting from two leaf nodes and then repeatedly computing the value of the parent node as a hash of the values of the two child nodes.

We say that a sibling path $P = (v_l, v_{n0}, v_{n1}, ..., v_{ni})$ of $MT_{H,b}(X)$ is valid, if i is indeed the height of the tree, and th e calculated root value in the sibling path is the same as the r oot value of $MT_{H,b}(X)$. Note that in order to verify that a giv en sibling path is valid, it is sufficient to know the number of leaves and the root value of $MT_{H,b}(X)$.

2. Basic structure

According to [4], the following POW scheme is a strong proof of ownership protocol with robustness $(1-\alpha)^u$.

(1) Let the parameter $b = 256$ be the leaf size of the Merkle tree; ε, the desired robust boundary; and α, the erasure recovery capability of the erasure code. We use the collision resistant hash function $H(\cdot)$, and the erasure code $E(\cdot)$ with a recovery capability of α.

(2) Once the M-bit file F is inputted, the verifier calculates the code $X = E(F)$ and the Merkle tree $MT_{H,b}(X)$, and holds only the root of the tree and the number of leaves as the verification information.

(3) During the proof protocol, the verifier randomly selects a u leaf index, $l_1,...,l_u$, u is the smallest integer that makes $(1-\alpha)^u < \varepsilon$. The verifier asks the prover for the sibling path of all the leaves, and calculates the Merkle root by using the returned sibling path, and determines whether all the sibling paths are valid for the Merkle tree $MT_{H,b}(X)$.

(4) If all sibling paths are valid, the prover proves to the verifier that it owns the file.

In this paper, to prevent unauthorized access, a secure proof of ownership scheme, POW, is used so that when the server discovers a copy, the client provides proof that it does have the same file.

E. blockchain technology and ethereum

Blockchain technology was introduced to the world by "bitcoin". Bitcoin is a P2P encrypted digital currency. Since the establishment of Nakamoto Satoshi[13] in 2008, its value and popularity have increased. In the case of bitcoin, blockchain supports a payment system and a complete digital currency, which is secure and decentralized. That is, it is a user-driven peer-to-peer network with no central authority. As bitcoin began to attract attention, developers took advantage of the blockchain technology as an infrastructure to create their own platform (in addition to the primary use of bitcoin to facilitate the transfer of digital money). On the one hand, some platforms use the bitcoin network as infrastructure to notarize or certify the existence of digital documents, crowdfunding, dispute mediation, and spam control. On the other hand, several platforms have emerged in the form of tokens, a blockchain-based cryptocurrency that aims to enhance bitcoin's capabilities by implementing its own features and functions. So far, there are almost 2,000 tokens, but the most attractive are ether [20], litecoin[35] and dogecoin[36].

In this paper, we will use the ethereum platform. In 2013, ethereum [20] [37] was proposed by Vitalik Buterin to create a distributed computing platform based on blockchain, with the ability to build and run decentralized applications and smart contracts. Ethereum's development was achieved through online crowdfunding in mid-2014, and the platform was launched in 2015. Ethereum has since gained considerable attention and is a pioneer of blockchain 2.0[37], the next generation cryptographic space. As a cryptocurrency based on the blockchain, it provides the same functions as bitcoin: simple mobile payment, reliability, complete control over one's own money, high availability, fast internal payment, zero or low cost, protected identity and privacy. However, ethereum offers an online transfer of digital

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2939492, IEEE Access

currency that enables its users to build and deploy smart contracts. So ethereum is a programmable blockchain.

F. ethereum virtual machine

The core of ethereum is the Ethereum Virtual Machine (EVM) [20], which can execute code with arbitrary algorithm complexity. Ethereum is "turing-complete", and developers can use existing programming languages to create applications that run on ethereum virtual machines, such as JavaScript and Python. To maintain consistency across the blockchain, each network node runs an ethereum virtual machine. The decentralized consistency allows ethereum to be highly fault tolerant, with zero downtime, and the ability to store data on the blockchain to remain constant and censor resistant. Computing in ethereum virtual machines is paid for with ether (ETH), the currency used by ethereum.

G. ethereum account

The basic unit of ethereum is the account. Ethereum uses two types of accounts: external accounts (EOA) and contract accounts. The external account EOA is controlled by a corresponding private key, has an Ether balance, can send transactions (forward Ether to another account or trigger a contract code), and has no associated code. The external account EOA is similar to a bitcoin address and consists of hexadecimal digits, such as 0x990069a8450174f7a988ace7e3211309b5a23296, so the external account EOA is anonymous. A contract account has its own Ether balance and associated code, and all actions are performed by the external account through the transaction. Execution of the contract code means receiving a transaction from the external account EOA. The contract code can also be triggered by messages from other contract accounts. Compared to Bitcoin scripts, contracts execute Turing's complete calculations and are written in high-level languages such as Solidity [38], Serpent, and more. The behavior of a contract is entirely dependent on his code and the transactions initiated to it, creating the possibility for a decentralized system.

H. smart contract

Smart contract [20-21] is essentially a program written in a certain computing programming language, which can be run in the container provided by the blockchain system, and at the same time, the program can be automatically run under the activation of some external and internal conditions. The combination of such features and blockchain technology can not only avoid artificial malicious tampering with rules, but also take advantage of the efficiency and cost of smart contracts. Since the code of the smart contract is stored in the blockchain, the operation of the smart contract is also in the container provided by the blockchain system. Combined with the cryptographic principle used by the blockchain technology, the smart contract is naturally tamper resistant and anti-counterfeiting features. The results produced by the smart contract are also stored in the block, so that the execution from the source, the execution process and the result are all executed in the blockchain, which ensures the authenticity and uniqueness of the release, execution and record of the smart contract.

I. transaction information

The smart contract [20-21] deployment is essentially a transaction initiated on ethereum. Ethereum transactions are signed data packets that allow the transfer of ether from one account to another. In addition to transmitting Ether, transactions can trigger the execution of code in smart contracts. Transactions include the initiation account address, the transaction destination account address, gasPrice, gasLimit, the Ether value transferred, additional data fields, etc. (the specific meanings of the transaction information parameters are shown in Table 1 below). The originating transaction account can place the data field into the additional data field of the transaction, while in the smart contract, the binary bytecode of the smart contract code is placed into the additional data field. In this scheme, we mainly make fair payment between client and cloud storage server through smart contract. Each call to a smart contract is an Ethereum transaction and can trigger the execution of the relevant code in the contract.

TABLE 1. The specific parameters of the transaction information

Parameters	Parameters Meaning
blockHash	Block hash in blockchain
blockNumber	Block height in blockchain
contract address	The contract account address (only when the contract is created, the transaction returns the contract address, the rest is null)
from	Source of the transaction (Ethereum account for deploying smart contracts)
gasPrice	The gas value required for the transaction
gasLimit	Maximum amount of gas allowed to be consumed
hash	Transaction hash (you can get the information of the entire transaction through this value)
input	Binary bytecode for smart contracts
to	Transaction destination address (the transaction destination information generated after deploying the smart contract is null)
value	Transaction cost

IV. System Model and Security Requirements

Our scheme is improved on the cloud deduplication system [22]. The original scheme [22] introduces the concept of deduplication into common cloud storage schemes, and uses traditional payment methods with trusted third parties for transactions. And our scheme based on cloud storage deduplication, cancel the third party, introducing the ethereum smart contract for both sides pay agreement. More importantly, clients and cloud storage service providers use ethereum smart contract to transfer and pay tokens, and every contract call is recorded on the blockchain. Therefore, the information transfer between the client and the cloud storage service provider is tamper resistant and non-repudiation. The symbolic annotations used in the scheme are shown in Table 2.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2939492, IEEE Access

TABLE 2. The symbolic meaning in the scheme

Symbols	Symbolic meaning
S-CSP	The cloud storage server
ID_t	The client
b_1	The payable when "uploading files duplicated"
b_2	The payable when "uploading files not duplicated"
ServerContract	Smart contracts deployed on the server side
ClientContract	Smart contracts deployed on the client side
ClientAccount	The ethereum account of the client
ServerAccount	The ethereum account of the cloud storage server
pk_{S-CSP}	The public key of the cloud storage server
sk_{S-CSP}	The private key of the cloud storage server
μTAB	The table that stores the records uploaded by the clients
$TAB(tag, link)$	The table that stores the file tag *Tag*
TxId	A transaction number containing payment transaction information
σ	The signature of the cloud storage server to the client
L	The location where files are stored on the cloud storage server
F	The file uploaded by the client
Tag	The file tag generated using tag generation algorithm
state	The status of the file uploaded by the client
h	The hash function to generate a file tag to achieve tag unification
K	The convergent key
C	The ciphertext of file F

A. System Model

The scheme has three roles, namely client, cloud storage server and ethereum blockchain, where miners in ethereum blockchain are not considered.

Client: request to upload the encrypted file to the cloud storage server, and make the active payment when receiving the payment from the cloud storage server.

Cloud storage server: our solution is to build on the deduplication, that is, the cloud storage server needs to determine whether the file is duplicated or not. According to whether the uploaded files of the client are duplicated or not, the scheme provides two different payables: "uploading files duplicated", payable is b_1; "uploading files not duplicated", payable is b_2, $b_1 < b_2$. The cloud storage server then sends the corresponding amount payable. After the cloud storage server confirms the payment from the client, the cloud storage server performs the operation of uploading encrypted files and performs different file uploading algorithms according to whether the files are repeated or not. The specific file upload process is shown in figure 2. After the encrypted file is uploaded, the cloud storage server returns the payment receipt and the file link pointer to the client.

Ethereum blockchain: The client and cloud storage server deploy smart contract *ServerContract*, *ClientContract* respectively, and smart contracts open storage data and interface for acquiring data on Ethereum.

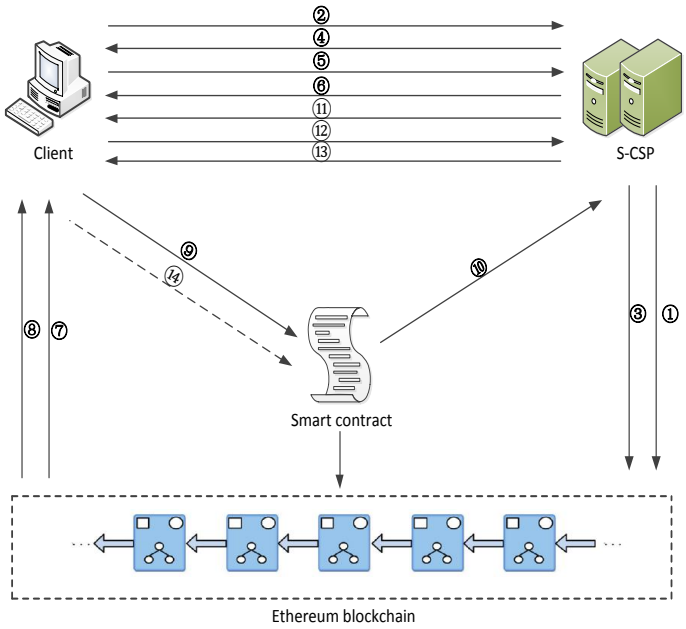

FIGURE 1. System Model

The description of the steps in the Figure 1 is as follows:

① The cloud storage server S-CSP deploys the server contract *ServerContract* to blockchain;

② The client registers in the cloud deduplication system for payment;

③ The cloud storage server S-CSP responds to the client's registration request, authorizes the client, and writes the client's address into the blockchain;

④ The cloud storage server S-CSP sends the registered transaction number, S-CSP contract address, contract ABI, and client contract code to the client through a secure channel.

⑤ The client requests to upload files to the cloud storage server S-CSP;

⑥ The cloud storage server S-CSP returns the payable to the client according to the request of the client through a secure channel.

⑦ Before making payment, the client checks whether the registration is successful or not according to the registered transaction number;

⑧ The client checks whether the ethereum account has enough funds to pay;

⑨ The client initiates the payment;

⑩ S-CSP obtains *TxId* of payment transaction information through ethereum smart contract;

⑪ After the cloud storage server confirms receipt of payment, it returns the transaction receipt for the client;

⑫ The cloud storage server S-CSP performs file uploading;

⑬ The cloud storage server S-CSP returns the signature σ and the link pointer L to the file;

⑭ When the cloud storage server S-CSP has a malicious situation, the client can initiate a penalty transaction fine to get a penalty.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2939492, IEEE Access

After understanding the overall architecture of the blockchain-based cloud deduplication system fair payment scheme, we now elaborate on the cloud deduplication of file uploads. The specific process is shown in Figure 2.

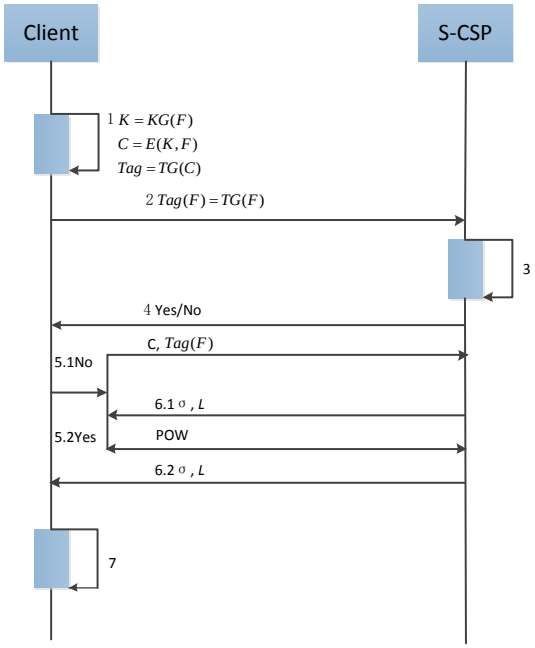

FIGURE 2. Secure cloud deduplication process

① The client uses convergent encryption scheme to encrypt file *F*, and then uses tag generation algorithm to generate the file tag *Tag*;

② The client sends the file tag *Tag* to the cloud storage server S-CSP;

③ After the cloud storage server S-CSP receives the file tag, it checks whether the tag of the same file exists in the table μTAB, and stores the record in the table;

④ The cloud storage server returns the check result of the file tag to the client;

⑤ If the client receives the return result "no file duplicate", the client will upload the ciphertext and file tag to the cloud storage server; If the client receives the return result "file duplicate", the client will run the POW scheme to prove that it owns the file stored in the cloud storage server;

⑥ The cloud storage server returns the corresponding signature σ and file link pointer *L* to the client after confirming payment;

⑦ The client stores the convergent key *K* and file link pointer *L*, and deletes file *F* at the same time.

The fair payment scheme of cloud deduplication system based on blockchain consists of the following algorithms:

System setup:

Initializing convergent encryption scheme (*KG*,*Enc*,*Dec*,*Tag*) to encrypt client data; Also, initialize the consensus solution POW as a black box so that the client can prove to the cloud storage server S-CSP that it has some specific data.

The cloud storage server S-CSP initializes public-private key pairs (pk_{S-CSP}, sk_{S-CSP}) and publishes public key to all clients in the network. At the same time, two types of storage systems are initialized: rapid storage system is used to store tag table $TAB(tag, link)$, *tag* means the file tag, and *link* stores the corresponding file address (that is, the location of the encrypted file in the file storage system); and the corresponding user information table $\mu TAB(tag, num, user(ID, time, state))$, *tag* item is as described above, *num* refers to the number of users sharing the same tag, *user* refers to the user information, where *ID* refers to the user's identity, *time* records the time when the user uploads the file, and *state* marks the status of the user's file upload (If S-CSP does not respond, state is 0, state is 1 when confirming the response, and state is -1 when deleting the response). File storage systems are used to store encrypted copies of data.

In Ethereum, the client ID_t and cloud storage server S-CSP create Ethereum accounts and deploy corresponding smart contracts to the blockchain respectively, and publicize the smart contract address and smart contract *ABI* (Application Binary Interface, which contains several functions in JSON format) for later work. The client registers with the cloud deduplication payment system to make a payment.

File outsourcing:

The client calculates the file tag *Tag* and sends it to the cloud storage server S-CSP; S-CSP checks if the same tag is in the table μTAB, stores the record, and sets *state* to 0. In turn, S-CSP replies to the client with "file duplicate" or "no file duplicate" and returns different payables to the client depending on whether the same file exists.

After the cloud storage server S-CSP confirms payment from the client, S-CSP performs file upload. If the response received by the client is "no file duplicate", the encrypted file and tag are uploaded to S-CSP, and S-CSP returns the signature σ of the client and the link pointer *L* of the file, and sets *state* to 1. In the case of "file duplicate", the client proves to the cloud storage server S-CSP that it actually has the same file by running the POW scheme (see section 2.4) without actually sending the file. If POW passes, the cloud storage server S-CSP returns the signature σ of the client and the link pointer *L* of the file to the client, and sets *state* as 1. If POW fails, S-CSP aborts the upload operation.

Payment phase:

When the client sends an upload file request to the cloud storage server S-CSP, S-CSP returns different payables to the client based on whether the same file exists. The client transfers tokens by calling the method transfer(address _to, uint256 _value) in the smart contract, that is, payment. The cloud storage server S-CSP obtains the payment transaction number *TxId*; after confirming the payment, the transaction receipt is sent to the client.

If the cloud storage server S-CSP appears malicious, the client can appeal to the S-CSP and ask it to return the link pointer *L* of the file. If the appeal fails, the client can initiate

fine transactions that have been preset in the smart contract to get the penalty.

File download:

The client ID_t first sends a request to the cloud storage server S-CSP containing an identity and a pointer to a file link. Upon receiving the request, the S-CSP will check if ID_t is eligible to download the file. If it fails, S-CSP will send an abort signal to ID_t indicating that the download failed. Otherwise, S-CSP returns the corresponding data. After receiving data from S-CSP, the client ID_t uses the convergent key K to decrypt the data and recover the original file.

B Security Requirements

Next, we will consider the safety performance of the system.

Confidentiality: In the blockchain environment, the payment process from the client to the cloud storage server is not affected by illegal modification. The client's payment process is confidential.

Authentication: In the fair payment system for cloud deduplication based on blockchain, only authorized clients can use the system to make payments.

Scalability: In a fair payment system for cloud deduplicati -on based on blockchain, the number of clients should be scal -able.

Integrity: Integrity means that if an honest client and an honest cloud storage server execute an agreement, the honest cloud storage server can get the payment from the client and the honest client can receive the required data.

Fairness: When the malicious client executes the protocol, if the client does not pay, the cloud storage server will not perform file uploading and the client cannot get any required data; when the malicious cloud storage server executes the protocol, if it fails to return the result data to the client in time, the client will appeal and require it to return the required result data. If the appeal fails, the client will initiate fine transaction to obtain the penalty from the cloud storage server.

B Potential Attacks

The following attacks may exist in the cloud deduplication fair payment system.

Denial of service attack: An attacker could issue a large number of upload file requests to the network layer to crash the system.

Unfair upload attack: like the sybil attack, an attacker forges a large number of client request file uploads, but the client does not pay and does not require the server to return results. In turn, the server causes a lot of useless work and consumes resources.

Collusion attacks: Many clients may gather to request file uploads without paying for them in order to consume server resources. This is an extension of the unfair upload attack.

Re-Entry attack: The attacker has many malicious entries into the network. When a malicious action on the client is exposed, it can register as a new client.

V. SCHEME CONSTRUCTION

The fair payment protocol based on blockchain proposed by us can be directly combined with most cloud storage systems to realize decentralized fair payment in the cloud storage payment system. In the scheme, we added a group of authorized users to the smart contract to prevent anyone from invoking the smart contract to make payment. An unauthorized user's request for a payment call will be rejected by the smart contract. Meanwhile, our scheme guarantees the fairness of payment process through smart contract.

In this scheme, on the basis of cloud deduplication payment system, Ethereum blockchain technology is introduced to improve the cloud deduplication payment system scheme. And the Ethereum smart contract technology is used for payment. Through the blockchain network, payment is transparent and publicly viewable.

A. CONCRETE CONSTRUCTION

System setup:

The parameters needed to initialize the scheme in the system setup phase.

(1) Initiate convergent encryption scheme (KG,Enc,Dec,Tag) to encrypt client data and perform deduplication on the cloud server. The convergent encryption scheme adopted by cloud deduplication payment system is as follows: firstly, a hash function h is selected. For file F, set the convergent key K equal to the hash of file F, that is, $K = h(M)$; ciphertext C is equal to the use of convergent key K to encrypt file F, that is, $C = Enc(K, M)$; Tag is equal to hashing ciphertext C, that is, $Tag = h(C)$.

(2)In addition, the client initializes POW algorithm, specifically, selects collision resistant hash function H and erasure code $E:\{0,1\}^M \to \{0,1\}^{M'}$, and specifies the leaf size $b = 256$ of Merkle tree. It is used to prove to the cloud storage server S-CSP that the client has files already stored on the cloud storage server. Here, we use the POW algorithm as a black box.

(3)The cloud storage server S-CSP initializes public-private key pairs (pk_{S-CSP}, sk_{S-CSP}) , and issues the public key to all clients in the network, and secrets the private key.

(4)Initialize the fast storage system to store tags table $TAB(tag, link)$ for effective duplicate checking; It is used to store user information table $\mu TAB(tag, num, user(ID, time, state))$ for effective repetition rate check. Initializes a file storage system for storing copies of encrypted data, that is, the file ciphertext C.

The *tag* in *TAB* table represents the file tag, and *link* stores the corresponding file address. In the table μTAB , the tag is as described above, *num* refers to the number of users sharing the same tag, *user* refers to the user information,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2939492, IEEE Access

where *ID* refers to the user's identity, *time* records the time when the user uploads the file, and *state* marks the status of the user's file upload to indicate whether the file upload was successful or not.

If the client wants to store the file and make a query to the cloud storage server S-CSP, the S-CSP stores the record in the table μTAB and sets *state* to 0.If the S-CSP confirms a response to the record, that is, the file was stored successfully, then the status for this user is 1.If S-CSP responds to the deletion of the record, that is, the file storage fails, the status about the user is marked as -1.

(5) The file storage system is initialized to *NULL*. Note that if the cloud storage server S-CSP has to provide different *link* for different clients, it must also add a user identity entry to the *TAB* table.

(6) In Ethereum, client ID_t and cloud storage server S-CSP create ethereum account *ClientAccount* and *ServerAccount* respectively for later work.

(7) In the system setup stage, when building the smart contract, in order to make the payment system more secure and perfect, we introduce the cloud storage server ServerContract. The S-CSP strictly limits the access of users in the payment system and takes the form of authorized user set authorizeClients[newClientAddress]. The cloud storage server S-CSP can add, modify and delete users through the relevant function interface of the contract. (See algorithms 3 and 4 below for details)

(8) In order to ensure the payment fairness of the system, we preset fine transaction in server S-CSP contract ServerContract. If the server is malicious and the client fails to appeal, the client can initiate the fine transaction and get the penalty. (See algorithm 5 in the following section for details)

File upload:

Assume that the client ID_t upload file is *F*. Then, in the file upload phase:

(1) Once file *F* is entered, the client ID_t calculates and sends file tag $Tag(M) = TG(C)$ to the cloud storage server S-CSP. The S-CSP checks if the same file tag is in the table μTAB. The S-CSP stores the client ID_t record and sets *state* to 0.

(2) In general, we consider public deduplication (rather than private deduplication between individual user data) and assume that users always upload different data to the cloud. However, if the "file tag duplicated", the cloud storage server S-CSP provides the client with a response; if "no file tag duplicated", the cloud storage server S-CSP performs fine-grained deduplication [40].

Once a file tag $Tag(M)$ is received, the cloud storage server S-CSP verifies that the same file tag exists. If the same file tag exists, S-CSP responds to the client with "file duplicated"; otherwise, "no file duplicated". At the same time, S-CSP will return different payables to the client based on whether the files are duplicated, that is, "Uploading file is duplicated", payable is b_1; "Uploading file is not duplicated", payable is b_2.

After the cloud storage server S-CSP confirms payment from the client, S-CSP performs file upload. If the response received by the client is "no file duplication", the client uploads the unique encrypted file and file tag to the cloud storage server. Also, the S-CSP returns a signature σ to the identity *ID* of the client ID_t, and a file link pointer *L* pointing to the corresponding file address stored in the *link* field.

If the response the client receives is "file duplicate," the client runs the POW algorithm (see section 2.4) to prove that it actually has the same file *F* stored on S-CSP.

If POW passes, the S-CSP only returns a signature σ to the client and a link pointer to the file *L* to the client, with no further information to upload. Based on this, S-CSP will change the state of the client to 1; If POW fails, S-CSP aborts the upload operation.

Payment phase:

During the system setup phase, both the client ID_t and the cloud storage server S-CSP set up their own accounts in ethereum and deployed corresponding smart contracts. When the client sends an upload file request to the cloud storage server S-CSP, S-CSP returns different payables to the client based on whether the same file exists. The client transfers the token by calling the method transfer (address _to, uint256 _value) in the smart contract, that is, payment, where the transaction details are in the receipt. The transaction receipt includes the client address ClientAddress, the server address ServerAddress, payable amount, the transaction number *TxId*: the transaction hash receipt.getTransactionHash (), the block hash receipt.getBlockHash (), and the cost of gas (receipt.getGasUsed()). The client can use the transaction number *TxId* to trace each payment. The cloud storage server S-CSP obtains the payment transaction number *TxId*; upon receipt of payment, the receipt is sent to the client. The detailed payment process is described as follows:

(1) Before the client makes payment, the system first checks whether the client ID_t's ethereum address ClientAddress has sufficient funds, and if so, deploys the contract to the ethereum blockchain; otherwise, an exception is thrown (see algorithm 1 for details). Among them, the payable duePayment are from the payable returned by the cloud storage server to the client.

(2) The client initiates the payment, and after checking that there is enough funds, creates the smart contract ClientContract and deploys it to the Ethereum blockchain, and transfers the token by calling the method transfer(address _to, uint256 _value) in the smart contract to make the payment (see the algorithm 8 in the next section for details).

(3) The cloud storage server S-CSP obtains the payment transaction number *TxId*; after confirming the payment, S-CSP sends the transaction receipt to the client. The transaction number *TxId* in the transaction receipt: transaction hash receipt.getTransactionHash (); block hash

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2939492, IEEE Access

receipt.getBlockHash(); cost of gas(receipt.getGasUsed()), obtained by algorithm 2.

Algorithm 1 checkIfUserHasEnoughFunds

Input: user
Output: bool

```
if user's amount >= duePayment then
    return true;
else
    throw;
end if
```

Algorithm 2 executeTransferRequest

Input: request
Output: receipt

```
if request is null then
    throw;
else
    get contract ' s address;
    load contract;
    receipt ← contract 's transfer(to,value);
    return receipt.getTransactionHash(),
    receipt.getBlockHash(),receipt.getGasUsed();
end if
```

More often, in our system, we consider two special cases: the first one, the client is malicious because he did not pay after receiving the payment request from the cloud storage server; the second, the cloud storage server is not completely trusted. After receiving the payment from the client, the file link information and the receipt are not returned.

Case 1: our payment scheme requires the client to make payment first, and then the cloud storage server will execute the client's request for file upload after confirming the payment, and finally send the file link information to the client. Therefore, if the client does not pay, it cannot receive any information about the stored file, and uploading is meaningless.

Case 2: since our payment process is in the form of smart contract, the client can trace the payment information. If the client finds a malicious situation in the cloud storage server, it can appeal to the cloud storage server.

In the long-term development of the system, the cloud storage server should promptly handle client complaints and return the file link information and receipt. If the client still does not receive the file link information returned by the server, the client obtains the pre-deposit of the server by publishing a fine transaction.

File download:

The client ID_t first sends a request to the cloud storage server S-CSP containing an identity and a pointer to a file link. Once receiving the request, the S-CSP checks whether the client ID_t is eligible to download file F. Specifically, the S-CSP compares the identity information and file information in the request with the stored information already in the file storage system. If it fails, the S-CSP sends an abort signal to the client ID_t indicating that the download failed. Otherwise, the S-CSP returns the corresponding data. Once the client ID_t receives the data from S-CSP, the client uses the convergent key K to decrypt the data and restore the original file.

Remark 1: In our protocol, we assume that the blockchain system contains enough honest miners, of which 51% attack is unavailable. The blockchain is a secure environment with sufficient bandwidth to prevent denial of service attack. For malicious situations, smart contracts presuppose a penalty transaction to get a penalty.

Remark 2: For unfair upload attacks and collusion attacks, the cloud storage server will mark the malicious client to the database. If it reaches a certain number of times, it will be removed from the collection of authorized users of the system.

Remark 3: One way to prevent client re-entry attacks is to link the client's IP address as its unique identity. In our protocol, the blockchain is a secure environment where the number of clients is scalable. Only authorized clients can use the system to make payments.

B. Contract construction

This section mainly introduces the smart contract related interface and algorithm logic used in this paper. The Ethereum smart contract is written by solidity [38]. There are always some special variables and functions in the global namespace, which are mainly used to provide information about the blockchain. In this paper, we mainly use the following special variables:

msg.sender: The sender of the message or transaction (the current call). When the smart contract is deployed, it is the address of the contract creator, and when the smart contract is invoked, it is the address of the smart contract caller.

msg.value: The number of *wei* in the message sent. $1\ ether = 10^{18} wei$. For subsequent usage, we use *$msg.value* to indicate the number of *wei* attached to the message, and *$value* to indicate the number of fixed *wei.*

Server contract

The contract is deployed by the cloud storage server S-CSP and we call it the server contract ServerContract.

Server contract initialization: This process defines some of the contract's variables when the contract is created.

(1) Address type cloud storage server S-CSP variable, which defines the address of the S-CSP.

(2) The authorized user variable authorizeClients of the mapping type, which defines a mapping set from the authorized user address to the bool value. The cloud storage server S-CSP can add, modify, and delete collection elements through the relevant functional interfaces of the contract.

In this paper, the smart contract ServerContract is created and deployed by the cloud storage server S-CSP which mainly provides the following function interfaces:

1. addClient (newClientAddress): The function can only be executed by the contract creator (S-CSP). Each time the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2939492, IEEE Access

client sends the S-CSP a registration request and its certificate of identity (which can be done through a secure channel), the external ownership account EOA of the client is authorized through this function after the client's identity is verified. (See Algorithm 3 for details)

2. removeClient (oldClientAddress): The function can only be executed by the contract creator S-CSP. When the S-CSP needs to remove the client, it removes the client from the authorization set by passing the client's external ownership account EOA to the function. (See Algorithm 4 for details)

3. fine (client, fixedValue, startTime, daysAfter): The function is created and deployed by the cloud storage server S-CSP, but can only be executed by the client. When the client finds that the S-CSP appears malicious, if the appeal fails, the client will initiate a fine transaction after the time daysAfter, and get the penalty of the S-CSP (See Algorithm 5 for details). Among them, the penalty fixedValue is set to a fixed value.

4. withdraw (): The function can only be executed by the contract creator S-CSP, so that S-CSP can withdraw the contract account balance at any time. (See Algorithm 6 for details)

Algorithm 3 addClient

Input: newClientAddress
Output: bool

```
if msg.sender is not S-CSP then
    throw;
end if
if newClientAddress has exist then
    return false;
else
    authorizeClients[newClientAddress] ← true;
    return true;
end if
```

Algorithm 4 removeClient

Input: oldClientAddress
Output: bool

```
if msg.sender is not S-CSP then
    throw;
end if
if oldClientAddress hasn't exist then
    return false;
else
    authorizeClients[oldClientAddress] ← false;
    return true;
end if
```

Algorithm 5 fine

Input: client, fixedValue, startTime, daysAfter
Output: bool

```
if msg.sender is not client then
    throw;
end if
if currentTime >= startTime + daysAfter then
    require S-CSP's balances >= fixedValue;
    S-CSP's balances − fixedValue;
    client's balances + fixedValue;
    return true;
end if
```

Algorithm 6 withdraw

Input: null
Output: null

```
if msg.sender is not S-CSP then
    throw;
end if
if contract 's balance > 0 ether then
    send contract's balance to msg.sender;
end if
```

Client contract

Contracts are deployed by clients ID_t , and we call them client contracts ClientContract.

Client contract initialization: this process defines some of the contract's variables when the contract is created.

(1) The client ID_t variable of address type, which defines the client's address.

(2) A mapping type of user balance variable balances that defines a collection of mappings from user addresses to uint256 values. It is used to describe changes in the amount of wallet between the cloud storage server and the client. Client contracts mainly provide the following functional interfaces:

5. deposit(value): The function is used to store the ether to the client contract. The smart contract balance is used for the payment function of the client. (See Algorithm 7 for details)

6. transfer(_to , _value): The function is used to make payments to the cloud storage server S-CSP. (See Algorithm 8 for details)

In some cases, the contract creator needs to terminate the smart contract to obtain the ether in the contract, so he needs to call the self-destruct method *self-destruct (Address)*. After the contract is self-destructed, if anyone sends ether to this contract address, the ether can no longer be redeemed and will disappear. Therefore, the smart contract in this paper cannot easily implement the self-destruct contract method to avoid economic losses.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2939492, IEEE Access

Algorithm 7 deposit

Input: deposite value
Output: null

```
if $msg.value not equal deposite value then
    throw;
end if
send $value to client contract address
```

Algorithm 8 transfer

Input: to,value
Output: bool

```
if msg.sender is not client then
    throw;
else
    require client's balances >= $value;
    client's balances − $value;
    S-CSP's balances + $value;
    return true;
end if
```

VI. ANALYSIS AND EVALUATION

A. SECURITY ANALYSIS

This paper combines Ethereum blockchain, cloud deduplication system, payment mechanism and smart contract technology to realize the advantages of data storage and payment in traditional cloud storage system. The smart contract technology of ethereum has transformed the traditional payment scheme of cloud deduplication system, no longer relying on the third party, but realizing the payment interaction between the client node and the cloud storage server node through the smart contract technology. In this section, we discuss the benefits, security, and privacy of this scheme.

Conclusion1*:* Convergent encryption is of semantic security.

According to the security analysis of convergent encryption in literature [22], it is concluded that when encrypted data copies are unpredictable, they are semantically secure. That is, if the user does not have the file, they cannot obtain ownership of the data from S-CSP by running the proof of ownership protocol. Therefore, data is safe for adversaries who do not have it.

Conclusion 2*:* Our scheme realizes fairness of payment.

Proof: In a traditional scheme, we need to rely on a trusted third party to pay accordingly. However, the cloud storage server may return incorrect result or return no results to save resources. At this time, the client needs to obtain the payment voucher from the third party, and then make a complaint, etc., resulting in the waste of resources and time. In this paper, we propose a solution to ensure the fairness of the payment process through smart contracts. Smart contracts can honestly perform payment operations based on predefined logic and return corresponding results.

First, the payment is made by the client. When the server confirms that the payment is received from the client, the server sends the file link information to the client, which is to resist the malicious situation of the client; then, due to the transparency and traceability of the payment scheme, the scheme reduce server-side dishonesty. Therefore, the fairness of both parties to the payment is realized.

Conclusion 3*:* The scheme realizes payment integrity.

Proof: In normal case, when the client ID_t , and the cloud storage server S-CSP execute the protocol, the cloud storage server S-CSP will obtain the corresponding payment ether/token, and the client will receive the file link information and receipt.

Conclusion 4*:* The scheme realizes auditability.

Proof: When a malicious situation occurs on the client, the client sends a file to the cloud storage server without paying. Due to our payment system settings, if the client does not pay, the server will not send file link information, that is, the client will not get any results. Therefore, we mainly discuss the malicious situation of cloud storage server. Here, three time nodes are set, the final time of payment is t_1 , the final time of client receiving the result is t_2 , and the final time of complaint is t_3 , where $t_1 < t_2 < t_3$. If the client still does not receive the result from the cloud storage server at time t_2 , it enters the appeal stage and requires the cloud storage server to return the result. If the cloud storage server returns the result and the client verifies correctly, the transaction ends; otherwise, at the final time t_3 , the client initiates a fine transaction and gets the penalty of cloud storage server.

Conclusion 5*:* The scheme realizes authentication.

Proof: In this system, only authorized clients can use the payment system to make payment to the cloud storage server S-CSP. On the one hand, when the client initiates a file upload operation to the cloud storage server S-CSP, the system first checks whether the client account is in the authorized user set. If the client account is not in the authorized users of the system, the system requires the client to register with the cloud storage server S-CSP to obtain the authorization of the S-CSP. After the client account is successfully authorized by S-CSP, the cloud storage server S-CSP will send the registered transaction number, the S-CSP contract address, the contract ABI, and the client contract code to the client through a secure channel, so as to facilitate the client to conduct the subsequent payment operation; If the client account is in the collection of authorized users of the system, the client can perform subsequent payment and file upload operations. On the other hand, for attacks launched by malicious clients that cause the server to do a lot of useless work and consume resources, the cloud storage server S-CSP will mark the malicious clients to the database. If it reaches a certain number of times, it will be removed from the system authorized users.

B. VULNERABILITY ANALYSIS

Once deployed, smart contracts are difficult to modify,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2939492, IEEE Access

so if there are security holes in smart contracts, it is difficult to prevent attacks by hackers. In this case, it's important to ensure that you don't write code that has any security threats. Smart contracts belong to emerging things, so there are still many defects and security holes.

The Decentralized Autonomous Organization (DAO) was one of the major hacking incidents during ethereum's early development. The contract lost 3.6 million ethers and resulted in a hard fork in ethereum's network. Other vulnerabilities in smart contracts include Transaction-Ordering Dependence(TOD), Timestamp Dependency, Error Handling Exception, etc., which can cause significant losses. Therefore, using secure analysis tools to analyze code is critical.

SECURIFY[44, 45] is a security scanner of ethereum smart contracts, created by ICE center, ETH Zurich and ChainSecurity AG, a top provider for smart contract audits. The contract bytecode is first converted into their own custom language, and then compared with a validation module to verify whether its semantics are satisfied. Finally, the security report is generated. Figure3 shows the security analysis report for the smart contract. Problems with smart contracts are classified, and info displays detailed reports. The red box said Violation: the contract is guaranteed to violate the vulnerability, orange said Warning: the contract may, but us not guaranteed to violate the vulnerability. We use SECURIFY security scanner to analyze security before deploying the smart contracts. Figure3 (a) and Figure3 (b) are security analysis reports for the client contract and the server contract, respectively. The security scanner shows that the contract we used without any Violation.

FIGURE 3. (a)safety analysis report for the client contract

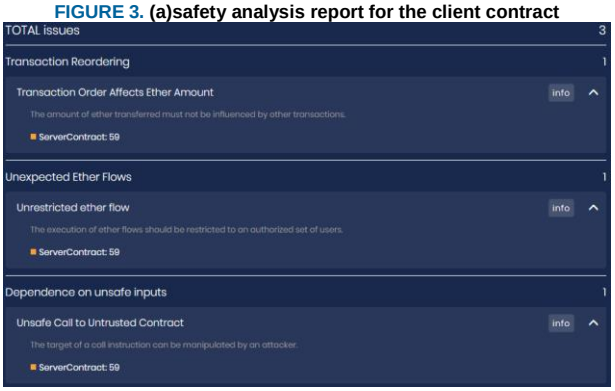

FIGURE 3. (b)safety analysis report for the server contract

C. PERFORMANCE ANALYSIS

In this section, we compare the performance of our protocol with similar protocols that already exist. Table 3 shows the comparison between the four schemes. First of all, the four schemes are all about payment under cloud services. Secondly, our scheme is based on the blockchain system, which does not require a trusted third-party currency system (bank). The schemes in [11, 22] both require a currency system for payment transactions. However, there are bottlenecks in third-party currency systems. If the transaction load in the system is too heavy, which will lead to the failure of the payment transaction, then the third-party currency system is impossible to complete the task. However, the payment transaction process of our scheme is on the blockchain network. As the blockchain network is a decentralized and peer-to-peer network, so there is no bottlenecks for payment transactions. Moreover, each payment transaction is untampered and traceable in our scheme. Scheme 42 uses the blockchain system to replace the traditional currency system to complete the payment transaction, but a trusted third party is still required to supervise both parties to achieve fair payment. Our scheme achieves fair payment without the need for any trusted third party. Firstly the scheme in [19] realizes the fair payment without the need of a trusted third party, but it realizes the fairness by the way of the client paying the deposit. However, our scheme does not require the client to pay a deposit, which is more practical in practice. Secondly, our scheme not only achieves fair payment without a trusted third party, but also achieves safe and effective cloud storage by combining deduplication technology. Finally, our scheme requires the cloud storage server S-CSP to return the corresponding receipt to the client, and both parties can trace each payment process according to the transaction number, so as to realize the transparency and verifiability of the payment process.

D. EXPERIMENTAL EVALUATION

In order to analyze the feasibility and performance of this scheme, we implement a prototype. The specific configuration of the experimental platform and environment is: Intel core i5-3230@2.60GHz processor, 4GB RAM, and the system are Windows10 and Linux Ubuntu 16.04LTS. The programming languages are Java and Solidity. External helper is web3j. Web3j is a lightweight Java development library for integrating Ethereum functionality, which is implemented in the Java version of the Ethereum JSONRPC interface protocol. Web3j provides a package of smart contracts for solidity that enables packaged objects generated by web3j to interact directly with all methods of smart contracts.

The implementation of this paper is based on the two operating systems, the Ethereum blockchain is deployed in the Linux ubuntu16.04 LTS established in the virtual machine. The smart contract was developed by the Solidity programming language and deployed on the private chain

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2939492, IEEE Access

TABLE 3. Comparison of the four schemes

Scheme	Confidentiality	Verifiability	Deduplication technology	Fairness	Based on blockchain	Trust third party	Receipt
Miao[22]	yes	yes	yes	no	no	yes	no
Chen[11]	yes	yes	no	yes	no	yes	no
Huang[42]	yes	yes	no	yes	yes	yes	no
Zhang[19]	yes	yes	no	yes	yes	no	no
Our protocol	yes	yes	yes	yes	yes	no	yes

created by the Ethereum Geth client under the Linux ubuntu 16.04 LTS system.

Under the Windows 10 system, use the development environment of the Remix IDE to develop and test. This development environment can be connected to the Ethereum Geth client via IP to deploy the smart contract on the Geth client. After the compilation is successful, use the web3j to generate the JavaBean from the smart contract to the Maven project in eclipse. By relying on some jar packages of web3j, the interaction between the client and the S-CSP for the smart contract is realized, which makes the fair payment algorithm of this paper better by using the smart contract.

Taking the literature [22] as an example, the scheme is applied to fair payment algorithm and experiment. Because of the high value of the Ethereum, it is necessary to test in the Ethereum private chain or the open test chain before the smart contract is deployed on the Ethereum main chain. The gas costs some operations on smart contracts are deployed for testing on Rinkeby, the test network of the Ethereum network. However, considering the experimental operability, the efficiency comparison of the solution is deployed for testing on the local private chain of the Ethereum network.

Compared with the traditional payment encryption scheme, the execution of the algorithm in this chapter has additional consumption mainly reflected in the gas consumption of the method call in the smart contract.

There are additional drains on the creation and execution of smart contracts, and Table 4 lists the gas costs and costs of some operations on smart contracts.

Considering the wide application and circulation of tokens at present, this experiment uses the ERC20 standard to produce tokens and Ether for testing respectively. Next we analyze the cost of creating and executing the function of the smart contract. First, in April 2019, 1 *ether*≈160 *USD* and set 1 *gasPrice*≈1 *Gwei*， $1Gwei = 10^{9} wei = 10^{-9} ether$.

TABLE 4. The smart contract cost (*gasprice* = 1 *Gwei*，1 *ether* = 160 *USD*)

function	GasUsed	Actual Cost(ether)	USD
ServerContract create	662390	0.00066239	0.1059824
addClient	44305	0.000044305	0.0070888
removeClient	14319	0.000014319	0.00796112
fine	49757	0.000049757	0.00796112
ClientContract create	763098	0.000763098	0.12209568
transfer	51417	0.000051417	0.00822672

Table 4 shows the cost of some of the operations of the smart contract, with little change in the cost of multiple executions. The cloud storage server S-CSP's smart contract creation operation is created only once, consuming 662,390 gas, and costs about $0.11. Each client's smart contract creation operation is created only once, consuming 763,098gas and costing about $0.12. When the cloud storage server S-CSP adds an authorized client, the cost of performing the addClient operation is about $0.007. When the cloud storage server S-CSP removes the authorized client, it costs about $0.008 to perform the removeClient operation. When the client receives the payment request from the cloud storage server S-CSP, the transfer operation will cost about $0.008. When the client finds that the cloud storage server S-CSP is maliciously affected and fails to appeal, it will cost about $0.008 to perform fine operation after the time daysAfter. These costs are based on prototypes deployed on the blockchain and can be reduced using optimized code. If the input size of these functions is minimal, the cost can be further reduced.

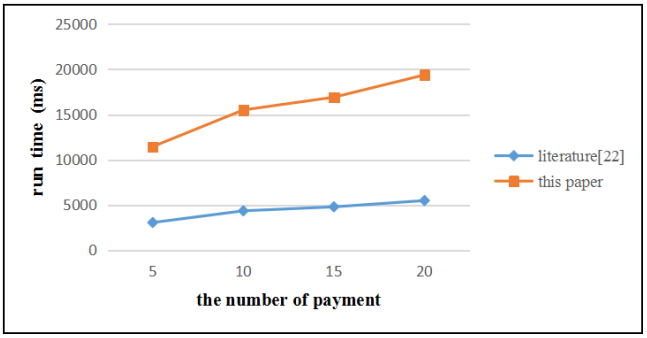

FIGURE 4. Run time of algorithm under different number of payment

The abscissa in Figure 4 is the number of payments at the same time, the number is 5, 10, 15, 20; the ordinate is expressed as the running time of the fair payment algorithm. The broken line of the blue diamond shape indicates the change trend of the execution time of the algorithm in the literature [22] with the increase of the payment; and the broken line of the orange square indicates the change rule of the execution time of the algorithm as the payment grows. Similarly, the execution time of the original algorithm increases as the payment increases. The running time of the algorithm is almost consistent with the trend of the running time of the original algorithm. Since the protocol of this chapter is based on blockchain, it is slightly higher in efficiency than the original scheme.

VII. CONCLUSION

In this paper, we propose a decentralized fair payment scheme based on blockchain technology. Our proposed protocol can be applied as follows: when Mary plans to buy

something in an online mall, she needs to register as a user of the online mall and then add her favorite item to the shopping cart and pay for it. In the meantime, Mary will receive a receipt for this order. The merchant then sends Mary the items for this order. Then Mary received the purchase. At this point, a normal payment process is completed. However, if the merchant does not send the goods to Mary after Mary pays, then at time A, Mary can appeal to the merchant and ask the merchant to send the goods. If Mary receives the purchase, the payment process is completed; Otherwise, at time B, Mary initiates a fine transaction to get the merchant's pre-existing penalty on the blockchain network. Our scheme improves the traditional payment system based on trusted third party by using the smart contract technology of ethereum. In order to prevent the trusted third party from reaching the bottleneck due to the excessive visits of users, the decentralized payment scheme is realized through the interaction between the client node and the cloud storage server node. On the one hand, based on the smart contract under ethereum blockchain, the system solves the problem of fairness of payment under malicious circumstances and the opaque payment process in traditional payment, and the payment process is traceable. On the other hand, the system weakens the attacks that occur on the client to some extent by combining our fair payment scheme with deduplication. Experiments have shown that the cost of making a payment to a cloud storage server by a client is minimal.

The shortcoming of the scheme in this paper is that the decentralized structure is not complete. The scheme is based on cloud storage platform. Future work can replace cloud storage platform with a decentralized storage platform, such as InterPlanetary File System IPFS [39], Storj [33], etc., including the study of decentralized fair payment schemes.

REFERENCES

[1] Douceur J R, Adya A, Bolosky W J, et al. Reclaiming space from duplicate files in a serverless distributed file system[C]//Proceedings 22nd international conference on distributed computing systems. IEEE, 2002: 617-624.
[2] Li J, Chen X, Li M, et al. Secure deduplication with efficient and reliable convergent key management[J]. IEEE transactions on parallel and distributed systems, 2013, 25(6): 1615-1625.
[3] Bellare M, Keelveedhi S, Ristenpart T. Message-locked encryption and secure deduplication[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2013: 296-312.
[4] Halevi S, Harnik D, Pinkas B, et al. Proofs of ownership in remote storage systems[C]//Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011:491-500.
[5] Abadi M, Boneh D, Mironov I, et al. Message-locked encryption for lock-dependent messages[C]//Annual Cryptology Conference. Springer, Berlin, Heidelberg, 2013: 374-391.
[6] Keelveedhi S, Bellare M, Ristenpart T. DupLESS: server-aided encryption for deduplicated storage[C]//Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13). 2013: 179-194.
[7] Carbunar B, Tripunitara M. Fair payments for outsourced computations[C]//Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conferenceon. IEEE, 2010: 1-9.
[8] Carbunar B, Tripunitara M V. Payments for Outsourced Computations[J]. IEEE Transactions on Parallel & Distributed Systems, 2011, 23(2):313-320.
[9] Shi L, Carbunar B, Sion R. Conditional e-cash[C]//International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2007: 15-28.
[10] Chen X, Li J, Ma J, et al. New and efficient conditional e-payment systems with transferability[J]. Future Generation Computer Systems, 2014, 37: 252-258.
[11] Chen X, Jin L, Susilo W. Efficient Fair Conditional Payments for Outsourcing Computations[J]. IEEE Transactions on Information Forensics & Security, 2012, 7(6):1687-1694.
[12] Chen X, Li J, Ma J, et al. New and efficient conditional e-payment systems with transferability[J]. Future Generation Computer Systems, 2014, 37: 252-258.
[13] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin. pdf." (2008).
[14] Andrychowicz M, Dziembowski S, Malinowski D, et al. Fair two-party computations via bitcoin deposits[C]//International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2014: 105-121.
[15] Andrychowicz M, Dziembowski S, Malinowski D, et al. Secure multiparty computations on bitcoin[C]//Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, 2014: 443-458.
[16] Ateniese G, Goodrich M T, Lekakis V, et al. Accountable storage[C]//International Conference on Applied Cryptography and Network Security. Springer, Cham, 2017: 623-644.
[17] Campanelli M, Gennaro R, Goldfeder S, et al. Zero-knowledge contingent payments revisited: Attacks and payments for services[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017: 229-243.
[18] Dong C, Wang Y, Aldweesh A, et al. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017: 211-227.
[19] Zhang Y, Deng R H, Shu J, et al. TKSE: Trustworthy Keyword Search over Encrypted Data with Two-side Verifiability via Blockchain[J]. IEEE Access, 2018, 6: 31077-31087.
[20] Wood, G. (2014). Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 1–32. https://doi.org/10.1017/CBO9781107415324.004
[21] "Ethereum blockchain app platform." [Online]. Available: https://www.ethereum.org/
[22] Miao M , Jiang T , You I . Payment-based incentive mechanism for secure cloud deduplication[J]. International Journal of Information Management, 2015, 35(3):379-386.
[23] Yuan J, Yu S. Secure and constant cost public cloud storage auditing with deduplication[C]//2013 IEEE Conference on Communications and Network Security (CNS). IEEE, 2013: 145-153.
[24] Bentov I, Kumaresan R. How to use bitcoin to design fair protocols[C]//International Cryptology Conference. Springer, Berlin, Heidelberg, 2014: 421-439.
[25] Gennaro R, Gentry C, Parno B. Non-interactive verifiable computing: Outsourcing computation to untrusted workers[C]//Annual Cryptology Conference. Springer, Berlin, Heidelberg, 2010: 465-482.
[26] Song W, Wang B, Wang Q, et al. Publicly verifiable computation of polynomials over outsourced data with multiple sources[J]. IEEETransactions on Information Forensics and Security, 2017, 12(10): 2334-2347.
[27] G.Developers. (2017) Google cloud platform.[Online]. Available: https://cloud.google.com/free/docs/frequently-asked-questions
[28] Buterin, Vitalik. "A next-generation smart contract and decentralized application platform." white paper (2014).
[29] Zhao Y, Li Y, Mu Q, et al. Secure Pub-Sub: Blockchain-Based Fair Payment With Reputation for Reliable Cyber Physical Systems[J]. IEEE Access, 2018, 6: 12295-12303.
[30] He Y, Li H, Cheng X, et al. A Blockchain Based Truthful Incentive Mechanism for Distributed P2P Applications[J]. IEEE Access, 2018, 6: 27324-27335.
[31] Dorsala M R, Sastry V N. Fair Protocols for Verifiable Computations Using Bitcoin and Ethereum[C]//2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE, 2018: 786-793.

[32] Delgado-Segura, Sergi, et al. "A fair protocol for data trading based on Bitcoin transactions." Future Generation Computer Systems (2017).
[33] C. Gray, "Storj Vs. Dropbox: Why Decentralized Storage Is The Future,"2014. [Online].Available:https://bitcoinmagazine.com/articles/storj-vs-dropbox-decentralized-storage-future-1408177107/
[34] Keelveedhi S, Bellare M, Ristenpart T. DupLESS: server-aided encryption for deduplicated storage[C]//Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13). 2013: 179-194.
[35] Litecoin.[Online].Available:https://litecoin.org/. Accessedon:July 20, 2018.
[36] Dogecoin. [Online]. Available: http://dogecoin.com/. Accessed on:July 10, 2018.
[37] Ulieru M. Blockchain 2.0 and Beyond: Adhocracies[M]// Banking Beyond Banks and Money. Springer International Publishing, 2016.
[38] Dannen C. Introducing Ethereum and Solidity : foundations of cryptocurrency and blockchain programming for beginners[M]// Introducing Ethereum and Solidity. Apress, 2017.
[39] Benet J. Ipfs-content addressed, versioned, p2p file system[J]. arXiv preprint arXiv:1407.3561, 2014.
[40] Li J, Chen X, Li M, et al. Secure deduplication with efficient and reliable convergent key management[J]. IEEE transactions on parallel and distributed systems, 2014, 25(6): 1615-1625.
[41] Memopal, Online backup. http://www.memopal.com/
[42] Huang H, Chen X, Wu Q, et al. Bitcoin-based fair payments for outsourcing computations of fog devices[J]. Future Generation Computer Systems, 2018, 78: 850-858.
[43] Harnik D, Pinkas B, Shulman-Peleg A. Side channels in cloud services: Deduplication in cloud storage[J]. IEEE Security & Privacy, 2010, 8(6): 40-47.
[44] Mense A, Flatscher M. Security Vulnerabilities in Ethereum Smart Contracts[C]//Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services. ACM, 2018: 375-380.
[45] https://securify.ch/