IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 11, 2019, accepted July 3, 2019, date of publication July 23, 2019, date of current version August 27, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2929205

A Secure Cloud Storage Framework With Access Control Based on Blockchain

SHANGPING WANG[1], **XU WANG**[2], **AND YALING ZHANG**[2]

[1]School of Science, Xi'an University of Technology, Xi'an 710048, China
[2]School of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048, China

Corresponding author: Xu Wang (xwang1628@gmail.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61572019, and in part by the Key Research and Development Program of Shaanxi under Grant 2019GY-028.

ABSTRACT Now more and more data are being outsourced to cloud services. In order to ensure data security and privacy, data are usually stored on the cloud server in the form of ciphertext. When a user requests access to the encrypted data, an access key distributed by a third party is needed. However, if the third party is dishonest, the security of the system will be threatened. Faced with this problem, in this paper, we propose a new secure cloud storage framework with access control by using the Ethereum blockchain technology. Our new scheme is a combination of Ethereum blockchain and ciphertext-policy attribute-based encryption (CP-ABE). The proposed cloud storage framework is decentralized, that is, there is no trusted third party in the system. Our scheme has three main features. First, as the Ethereum blockchain technology is used, the data owner can store ciphertext of data through smart contracts in a blockchain network. Second, the data owner can set valid access periods for data usage so that the ciphertext can only be decrypted during valid access periods. Finally, as the creation and invocation of each smart contract can be stored in the blockchain, thus, the function of the trace is achieved. The analysis of the security and experiment shows that our scheme is feasible.

INDEX TERMS Cloud storage, access control, Ethereum, blockchain, smart contract.

I. INTRODUCTION

Recently, with the rapid development of cloud computing and big data technology, more and more businesses and individuals choose to outsource their data to the cloud service. Most of the data stored in the cloud is highly sensitive, for examples, personal medical records and internal data of a company. In order to ensure the security of data and the privacy of users, data is usually stored on the cloud server in the form of ciphertext. In order to achieve access control of the data, encryption technology can be regarded as a security guarantee. But how to achieve access control for encrypted data is a big challenge. In 2007, ciphertext-policy attribute-based encryption (CP-ABE) was firstly introduced by Bethencourt [1]. In CP-ABE mode, a ciphertext is associated with an access policy, and a user's private key is associated with an attribute set. The user can decrypt the given ciphertext if and only if his attribute set satisfies the access policy established by the data owner. The data user obtains the corresponding key from the attribute authority center according to the attribute set he owns. The data owner can control the access of the data according to access policy.

In the CP-ABE scheme, one or more fully trusted attribute authorities or center authority are required. If the center authority is corrupted, it will endanger the entire system. Therefore, in the field of access control technology, decentralized system is urgent to get rid of the potential threat of trusted center authority.

Although in recent years, some people have studied blockchain-based access control schemes, most of them propose a framework or idea for such schemes. There is no specific solution to realize the integration of the decentralization idea of blockchain technology and access control technology. There is still a lot to be done in this area. Therefore, the decentralized access control research based on blockchain has important value and significance.

In this paper, we introduce the Ethereum blockchain technology to a ciphertext-policy attribute-based encryption algorithm, and use Ethereum smart contract technology to store the publicly available information into the blockchain network, at the same time to achieve the role of supervision and track the behavior of the data access. All access records

The associate editor coordinating the review of this manuscript and approving it for publication was Petros Nicopolitidis.

are recorded in the blockchain network. In our framework, decentralization of access control scheme is achieved without any trusted center authority by using blockchain technology.

Our contributions

The contributions of this paper are as follows:

(1) A secure cloud storage framework with access control based on blockchain is proposed, which is a combination of Ethereum blockchain and ciphertext-policy attribute based encryption (CP-ABE) algorithm, the aim is to realize fine-grained access control for cloud storage. No trusted attribute authority is required in our scheme. The key information is stored in the blockchain network by Ethereum smart contract technology. Thus, decentralization is achieved in our cloud storage framework.

(2) When an attribute set is assigned to the data user, the data owner can append an effective access period for the data user, and store an access period time of information on the Ethereum blockchain. Only when the valid access period time and the attributes of data user satisfy the access control policy, the data user can perform data decryption algorithm correctly.

(3) In the environment of Ubuntu linux system, smart contracts were created and deployed on the local Ethereum private network. The corresponding performance and cost were analyzed, the experiment show that our scheme is feasible.

The rest of the paper is organized as follows. In the section II, related work is presented. The section III introduces some basic knowledge of Ethereum blockchain. 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

The existing attribute-based encryption access control scheme is mainly based on single-center authority. When the center authority is untrusted or maliciously attacked, it may lead to key leakage. In response to this problem, some scholars have proposed a multi-authority attribute-based encryption access control scheme to decentralize the power of the center authority. In 2007, Chase [2] proposed a multi-authority attribute-based encryption scheme so that multiple authorities can assign attributes to users in the system, easing the threat of a single center authority's failure. In 2010, Lin *et al.* [3] proposed a threshold multi-authority fuzzy identity encryption scheme without center authority, which improved the security level of multi-authority systems. In 2011, Lewko and Waters [4] proposed a decentralizing attribute-based encryption scheme, which is essentially a multi-authority scheme without any center authority. In 2012, Yang K and other scholars [5] designed a multi-authority access control framework and constructed a specific multi-authority access control scheme. This scheme not only mitigates the threat of single point of failure brought by a single authority, but also supports the attribute update of data users in a multi-authority scheme. In 2016, Wei *et al.* [6] proposed a safe and efficient multi-authority access control scheme, which adopted linear secret sharing, in which multiple authorities effectively reduced the pressure on individual authority.

In 2008 Nakamoto [7] first introduced blockchain technology, till now many applications based on blockchain technology has penetrated into various industrial areas, especially in those areas where a third trusted party is needed. The decentralized and distributed structure of the blockchain can be trusted in global. Since the blockchain technology is a useful tool for large-scale collaboration between peoples without mutual trust. It can therefore be used in many traditional centralization areas to deal with transactions that were originally handled by intermediaries.

In 2015, Zyskind *et al.* [8] proposed a point-to-point decentralized computing model that allows different parties to store and run data together while keeping the data completely private. This model achieves automatic control of personal data by eliminating the need for trusted third parties. In 2017, Jemel and Serhrouchni [9] introduced a decentralized access control mechanism based blockchain. The blockchain nodes verify the legitimacy of user and add a time dimension to the shared file which is encrypted by ciphertext-policy attribute-based encryption. Xia *et al.* [10] presented a data sharing model between cloud service providers based blockchain. The model leverages the advantages of smart contracts and access control mechanisms to effectively track data access behavior and revoke access authorization for violation of access rules, addressing the problem of medical data sharing in an untrusted environment. In 2018, Xu *et al.* [11] proposed a decentralized capacity-based access control mechanism (BlendCAC), which can effectively protect the security of equipment, services and information in the large IoT (Internet of things) system. Liu *et al.* [12] design a framework using smart contracts and blockchain technology for tracking, managing and enforcing such data sharing agreements. Lin *et al.* [13] present a blockchain- based system for secure mutual authentication to enforce fine-grained access control policies, which provide privacy and security guarantees.

Generally speaking, scholars have recognized that the combination of blockchain technology with access control scheme is an effective way to solve the trust problems still existing today. For example, the literature [16] uses blockchain technology to store user's access control lists, the literature [19] uses blockchain technology for biomedical and health care applications, [20] uses three smart contracts for access control for the Internet of Things. However, blockchain technology has just emerging. Most of the researches on the decentralization of access control technology are just in the stage of framework, such as the literatures [14], [15], [17], [18], and there are few specific schemes.

TABLE 1. The notation table.

Notation	Description
DO	data owner
DU	data user
PK	the public key
MSK	the master key
k	the security parameters
U	the universal set of attributes
ck	the symmetric key
CT	the ciphertext
S	the attribute set of the data user
SK	the private key of the data user
SK'	the encrypted private key
Γ	the access tree(access policy)
$E_{ck}(M)$	the encrypted file

Therefore, in this paper, we propose a new secure decentralized cloud storage scheme with access control by using Ethereum blockchain technology. In this scheme, by introducing the blockchain technology, the problem of potential single point failure of the center authority in the original scheme is solved to some extent. At the same time, the introduction of a blockchain is equivalent to adding a logging system to the access control scheme to record all access operation records. The introduction of blockchain makes all access operations records non-tamperable and undeniable, which is more convincing as supervision.

III. PRELIMINARIES

In this section, we briefly review the relevant knowledge in order to better understand our scheme. Table 1 presents some of the notations used throughout the paper.

A. BILINEAR MAPPING

Let G and G_T be two groups of prime order q. g is the generator of G. A bilinear mapping $e : G \times G \rightarrow G_T$ satisfies the following properties:

1. Bilinearity: For any $u, v \in G$, and $a, b \in \mathbb{Z}_p$, it has $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: There exists $u, v \in G$, such that $e(u, v) \neq 1$.
3. Computability: For all $u, v \in G$, there is an efficient computation for $e(u, v)$.

B. ACCESS TREE

The access structure defines an authorized access set to describe the access policy. Let Γ denote an access tree. The tree contains leaf nodes and non-leaf nodes. The leaf nodes are associated with attributes, and the non-leaf nod- es are associated with threshold values. Let num_x indicates the number of child nodes of the node x. Let k_x denotes the threshold of the node x, $1 \leq k_x \leq num_x$. When $k_x = num_x$, the node denoted “and” gate in the logic. When $k_x = 1$, the node denoted “or” gate in the logic.

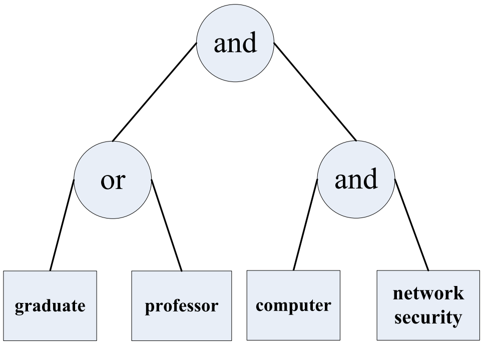

FIGURE 1. An access tree.

$parent(x)$: It means the parent node for node x except the root node.

$att(x)$: It represents the attributes associated with leaf node x.

$index(x)$: It indicates the number of child nodes for each non-leaf node x.

Figure 1 below is an access control tree based on an access policy, where a leaf node represents an attribute and a non-leaf node represents a threshold. Suppose that there are four attributes in this access tree. Assume that the attribute set is as follows: {graduate, professor, computer science, network security}. As it can be seen in the figure, only two types of people who satisfy the access tree. The first attributes set S1 is {computer science, network security, professor}. It represents the professors in the Institute of Network Security at the School of Computer Science. The second attributes set S2 is {computer science, network security, graduate}. It denotes the graduate students in the School of Network Security at the School of Computer Science.

C. BLOCKCHAIN TECHNOLOGY AND ETHEREUM

Blockchain technology [29] was introduced to the world by “Bitcoin”. Bitcoin is a P2P encrypted digital currency. Since “Nakamoto” [7] developed Bitcoin in 2008, its popularity have been increasing. In the Bitcoin system, the blockchain supports payment systems and complete digital currency, which is secure and decentralized. In other words, it is a user-driven and peer-to-peer network with no center authority.

As Bitcoin begins to draw attention, developers use the advantages of blockchain technology to create their own platform as an infrastructure (except for the primary use of convenience in digital currency transfer in Bitcoin). On the one hand, some platforms use the Bitcoin network as an infrastructure for notarization, crowdfunding, dispute resolution, and spam control. On the other hand, some platforms have emerged and are in the form of tokens which is a blockchain-based cryptocurrency designed to improve Bitcoin’s capabilities by implementing its own features and functions. Up to now, there are almost 2,000 kinds of tokens, but the most attractive are Litecoin [23] and Dogecoin [24]. In this paper, we will use the Ethereum platform.

In 2013, Ethereum [22], [25]–[27] was proposed by Vitalik Buterin. Ethereum is a blockchain-based distributed computing platform with the ability to build and run decentralized applications with smart contracts.

Ethereum's development was successful through online crowdfunding in mid-2014, and the platform went live in 2015. Since then, Ethereum has received great attention and is the pioneer of Blockchain 2.0 [27], which is said to be the next generation of blockchain. D. ETHEREUM VIRTUAL MACHINE

The core of Ethereum is the Ethereum Virtual Machine (EVM) [22], [29], which can execute code with arbitrary algorithm complexity. Ethereum is "Turing complete". Developers can use existing programming languages to create applications that run on Ethereum virtual machines, such as Javascript, Python, and more. In order to maintain the consistency of the entire blockchain, each network node runs an Ethereum virtual machine. The decentralized consistency makes Ethereum extremely fault tolerant, guarantees zero downtime, and data stored on the blockchain are immutable and anti-censorship. The calculations in the Ethereum virtual machine are paid for by ETH, which is the token used by Ethereum.

D. ETHEREUM ACCOUNTS

A basic component of Ethereum [29] is account. Ethereum uses two types of accounts, namely External Owned Accounts (EOA) and Contract Accounts. The External Owned Account (EOA) is controlled by a corresponding private key. An EOA has an ether balance, and EOA can send the transaction (forwarding some ether to another account or triggering a contract code) and there is no relevant code. An EOA is similar to a bitcoin address and consists of hexadecimal digits, such as 0x6695a16ef848d5fc520c2ea8a4-f09406f2cc9b1b. Consequently, An EOA is anonymous and can be shared publicly. A contract account has its own ether balance and associated code, and all actions are performed through the transactions created by EOAs. Execution of the contract code means receiving a transaction from an EOA. The contract code can also be triggered by messages from other contract accounts. Compared to Bitcoin scripts, contracts perform Turing-complete calculations and are written in high-level languages such as Solidity [28], Serpent, and so on. The behavior of a contract is entirely dependent on its code and the transactions initiated to it, creating the possibility for a decentralized system.

E. SMART CONTRACT

A smart contract [22], [26] is essentially a program written in a computing programming language that runs in a container provided by the blockchain system. At the same time, this program can also be automatically run under the activation of some external and internal conditions. The combination of those features of smart contract with blockchain technology can not only avoid artificial malicious tampering to rules, but also bring the high efficiency and low cost advantages of Smart Contracts into full play. 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 principles used by blockchain technology, smart contracts are naturally tamper-proof and anti-counter- feiting. 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 smart contract.

TABLE 2. 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

F. TRANSACTION INFORMATION

The deployment of smart contracts [22], [26] is essentially a transaction initiated on Ethereum. Ethereum transaction is a type of signature packet that allows some ether to be transferred from one account to another. In addition to transferring ether, it can also trigger the execution of code in smart contracts through transactions. The transaction includes the account address of the transaction, the account address of the transaction, gasPrice, gasLimit, the transferred ether value, the additional data field, etc. (the specific parameters of the transaction information are as shown in Table 2 below). The account that initiated the transaction can put the data into additional data field of the transaction. Similarly, in smart contracts, the binary bytecode of the smart contract code is placed in an additional data field. In this scheme, we mainly store and acquire ciphertext information through smart contracts. Each time a smart contract is called, it is an Ethereum transaction and triggers the execution of the relevant code in the contract.

IV. SYSTEM MODEL

In our scheme, we will use Smart Contract to store information about encrypted file. More importantly, data users and data owners use Ethereum smart contracts to store and retrieve ciphertext data to run encryption and decryption algorithms. Every contract call is recorded on the blockchain.

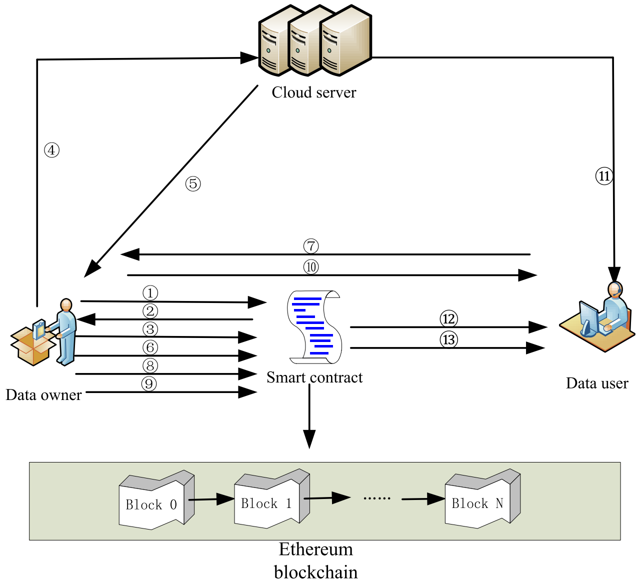

FIGURE 2. **System model.**

Therefore, the information transferred between data users and data owners is non-tamper with and non-repudiation.

There are four entities in our scheme, namely Cloud server, Ethereum blockchain, Data owner and Data user.

Cloud server: Responsible for storing encrypted files uploaded by data owners;

Ethereum blockchain: Deploy smart contracts on Ethereum, the smart contracts is of interfaces to store data and get data;

Data Owner(*DO*): Responsible for creating and deploying smart contracts, uploading encrypted files, defining access control policies, assigning attribute sets and appending valid access periods to data users;

Data User(*DU*): Accessing an encrypted file stored in the cloud server. When its attribute set satisfies the access structure embedded in a given ciphertext, it can decrypt the received ciphertext to obtain the content key to decrypt the encrypted file.

The description of the steps in the Figure 2 is as follows:

① The smart contract named *StorageSC* is deployed by *DO* in Ethereum.

② After the smart contract is deployed successfully, the contract address is returned.

③ *DO* stores the file *ID* hash H(*ID*) in the smart contract.

④ *DO* package the contract address *contractAddress*, file *ID*, and encrypted file $E_{ck}(M)$ and then upload to the cloud server.

⑤ *DO* records file path returned by cloud server.

⑥ *DO* stores the ciphertext of the encrypted document key in the Ethereum.

⑦ *DU* sends a access request to *DO*.

⑧ *DO* adds the effective period to *DU* and stores it in the smart contract.

⑨ *DO* encrypts the secret key of *DU* and stores it in the smart contract.

⑩ *DO* sends the contract address with user information through a secure channel.

⑪ *DU* downloads encrypted file from the cloud server.

⑫ *DU* obtains effective period from the smart contract.

⑬ *DU* obtains his secret key ciphertext from the smart contract.

The secure cloud storage framework with access control is based on blockchain is composed of the following algorithms:

Setup$(1^k, U) \rightarrow (PK, MSK)$: The setup algorithm is executed by *DO* with the security parameters k and the universal set U of attributes as inputs. Through the execution of the algorithm, the public key *PK* and the master key *MSK* are generated. At the same time, the smart contract named *StorageSC* is deployed in Ethereum. As shown in step ①② in Figure 2.

Before *DO* uploads the encrypted file to the cloud server, the file M with the file *ID* is encrypted by the AES symmetric encryption algorithm, and is recorded as $E_{ck}(M)$ (where *ck* is the encryption key). At the same time, the file name *ID* is hashed to H(*ID*) by a hash algorithm *sha*256. Next, the file *ID* hash H(*ID*) is stored in the smart contract, and the contract address *contractAddress*, file *ID*, and encrypted file $E_{ck}(M)$ are packaged and uploaded to the cloud server. *DO* records file path returned by cloud server. As shown in step ③④⑤ in Figure 2.

Encrypt$(PK, ck, \Gamma) \rightarrow CT$. The encryption algorithm takes the public key *PK*, access structure Γ and the symmetric encryption key *ck* as inputs, and outputs the ciphertext *CT*. The ciphertext *CT* is stored by *DO* in smart contract. As shown in step ⑥ in Figure 2.

KeyGen$(MSK, S) \rightarrow SK$: The key generation algorithm is still executed by the data owner *DO.DU* sends a access request to *DO*, then *DO* assigns an attributes set to *DU* and adds the effective access period to *DU*.The algorithm sets the attribute set S of *DU*, the master key *MSK* as inputs, and outputs the private key *SK* of *DU*. After *DO* and *DU* share the common key (the common key is generated by the Diffie-Hellman key exchange protocol), *SK* is symmetrically encrypted by the AES algorithm with the common key as encryption key. The ciphertext SK', which is the encrypted private key, is stored in the smart contract to ensure its privacy. As shown in step ⑦⑧⑨⑩ in Figure 2.

Decrypt$(PK, SK, CT) \rightarrow ck$: The decryption algorithm is executed by *DU*. The *DU* obtains access period from smart contract. Then *DU* performs decryption algorithm if and only if the time is within the valid access period. *DU* obtains *CT* and the private key's ciphertext SK' from smart contract. The SK' is decrypted as the private key *SK* by the symmetric encryption algorithm AES by using the common key as decryption key. The algorithm inputs the public key *PK*, private key *SK* and ciphertext *CT*. If and only if *SK* satisfies the access policy Γ, *DU* can recover the key *ck* of the encrypted document, so that the encrypted document is decrypted, otherwise the decryption will fail. *DU* obtains the encrypted document $E_{ck}(M)$ from the cloud server, decrypts the encrypted document M' by using key *ck*, and outputs the document M before DO encrypts. As shown in step ⑪ ⑫ ⑬ in Figure 2.

V. SCHEME CONSTRUCTION

Our proposed new secure cloud storage framework with access control based on blockchain can be used in conjunction with most CP-ABE algorithm to achieve decentralized fine-grained access control. The interaction between the data owner and the data user is achieved through the Ethereum smart contract technology, so that each data user's access is recorded in the Ethereum blockchain network. In order to better illustrate our framework, The CP-ABE algorithm in [30] is selected as an example in this paper, in which uses access tree as access control policy to achieve fine-grained access control of information.

A. CONCRETE CONSTRUCTION

Suppose that there are m attributes in the system, denoted as $U = \{1, 2, \cdots, m\}$.

Let $e : G_0 \times G_0 \rightarrow G_T$ be a bilinear group, where G_0 is a bilinear group of prime order p with generator g. Let $\mathrm{H} : \{0, 1\}^* \rightarrow G_0$ be a hash function, witch maps any attribute to a random element of G_0.

The details of our scheme are as follows:

1) PHASE 1: SYSTEM INITIALIZATION

Setup$(1^k, U) \rightarrow (PK, MSK)$. The setup algorithm is executed by DO, and outputs the public key PK and the master key MSK of the data owner DO.

The system selects a CP-ABE algorithm, taking the attribute-based encryption algorithm in [30] as an example. The DO chooses a bilinear group G_0 of prime order p with generator g and two random elements $\alpha, \beta \in Z_p$. The public key is published as: $PK = \{G_0, g, g^\alpha, g^\beta, e(g, g)^\beta, \mathrm{H}\}$ and the master key is $MSK = \{\alpha, \beta \in Z_p\}$.

In Ethereum, DO and DU create their own Ethereum accounts separately and ensure that the account balance is sufficient.

2) PHASE 2: FILE ENCRYPTION

Before DO uploads the file to the cloud server, the smart contract named *StorageSC* (See the next section for details) is deployed by DO to Ethereum, and smart contract address named *contractAddress* is obtained. The smart contract address, the smart contract ABI and DO's Ethereum account public key are published (ABI's full name is Application Binary Interface, which contains several functions expressed in JSON format), and the following processing is performed on the uploaded file M:

(1) The DO chooses a unique identifier ID for the file M. After the ID is hashed by the hash function *sha*256, it is recorded as $\mathrm{H}(ID)$. The $\mathrm{H}(ID)$ is stored in the Ethereum by the execution of function *setHashFileId*(*HashFileId*) (see Algorithm 1 in the next section for details) in smart contract.

(2) The DO encrypts the file M with the file ID using a symmetric encryption algorithm named AES, in which the content key ck is randomly obtained in the key space, and the encryption result is recorded as $E_{ck}(M)$, and upload $\{contractAddress, ID, E_{ck}(M)\}$ to the cloud server.

(3) The DO defines an access structure for access the encrypted content key ck, thus the content key ck is encrypted by selected attribute-based encryption algorithm with this access structure. The access structure in this selected attribute-based encryption algorithm is an access tree Γ whose leaf nodes are attributes. The ciphertext CT is output by running the following algorithm.

Encrypt$(PK, ck, \Gamma) \rightarrow CT$. The DO selects a polynomial q_x with degree d_x. For each node x in Γ, these polynomials are chosen in the following way in a top-down manner, starting from the root node R. For each node x in access tree Γ, set the threshold of the node to be (n_x, k_x). The threshold value k_x and the order d_x of q_x have the following relationship, $k_x = d_x + 1$.

Starting from the root node R, the DO selects a random number $s \in Z_p$, and sets $q_R(0) = s$. The DO then randomly selects d_R other coefficients of $q_{_R}$ to obtain polynomial $q_R(x)$. For any node x, set $q_x(0) = q_{parent(x)}(index(x))$, where $index(x)$ denotes the index of node x in its parent node $parent(x)$. Then randomly select d_x other coefficients to define q_x until the leaf node is calculated.

For access tree Γ, set X be its leaf nodes set and the key ck's ciphertext CT is created as

$$CT = \left\{ \begin{array}{l} \Gamma, \tilde{C} = ck \cdot e(g, g)^{\beta s}, C = g^s, \\ \forall x \in X : C_x = g^{\alpha q_x(0)}, C'_x = H(att(x))^{q_x(0)} \end{array} \right\}.$$

Because CT is stored as an object in experiments, it is necessary to serialize ciphertext into binary files after generating CT. Finally, the hexadecimal encoding of the path of the binary file containing the ciphertext is stored in the Ethereum by the execution of the function named *setCipherText*(*HashFileId*, *CipherText*) in the smart contract. (The DO may upload more than one file. The DU can return the corresponding ciphertext of different files by inputting different and valid file hashes *HashFileId* through the execution of function named *getCipherText*(*HashFileId*). For details, see Algorithm 5 and Algorithm 6 in the next section.)

3) PHASE 3: KEY GENERATION

The DU sends a request to the DO, including the Ethereum account public key and its account address *userAddress*, the expiration date of the access, and so on. The DO assigns an attributes set $S \subseteq U$ to the DU and adds the valid access period for the DU through the execution of function named *setInterval*(*userAddress*, *Inerval*) in the smart contract(See Algorithm 7 in the next section for details). The key generation algorithm is executed by the DO, and the key by running the selected attribute-based algorithm is generated as follows:

KeyGen$(MSK, S) \rightarrow SK$. The key generation algorithm firstly choose a random number $r \in Z_p$ as a secret. Then for each attribute $j \in S$, $r_j \in Z_p$ is randomly selected. Finally, the private key SK is generated as

$$SK = \{D = g^{\beta+\alpha r}, \forall j \in S : D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{\alpha r_j}\}.$$

DO uses the Diffie-Hellman key exchange protocol to calculate the common key based on the data user's Ethereum account public key. SK is symmetrically encrypted by the AES algorithm with the common key as encryption key. The ciphertext SK' of encrypted private key SK is mapped to the corresponding Ethereum user DU with address *userAddress*, and is stored in the blockchain by executing a function named *setSecretKey*(*useraddress*, sec *retKey*) in the smart contract (see the next chapter, Algorithm 3)

4) PHASE 4: FILE DECRYPTION

During the file decryption phase, the execution of the decryption algorithm requires the ciphertext CT and ciphertext SK'. Firstly, the DU obtains the contract address *contractAddress* and file ID from $\{contractAddress, ID, E_{ck}(M)\}$ uploaded by the DO in the cloud server. The file hash *HashFileId* is checked to see if it exists in Ethereum through the execution of function named *checkHashFileId*(*HashFileId*) in the smart contract. (This function needs to input the hash value *HashFileId* after hashing it by the hash algorithm *sha*256. See the algorithm in the next section for details.) If it does not exist, it cannot be continued (because the file ID hash is required as an index to obtain ciphertext information).

The DU obtains the access permission through the execution of the smart contract function *getInterval*() (see Algorithm 8 in the next section). The DU can proceed if the time is within the valid access period set by the DO. The ciphertext CT and the private key's ciphertext SK' are obtained through the execution of functions *getCipherText*(*HashFileId*) and *getSecretKey*() respectively (see Algorithm 4 and Algorithm 6 in the next section for details). (The SK' here is encrypted by the common key of both DO and DU, so it needs to be decrypted to SK by the symmetric encryption algorithm AES.) After obtaining the CT and SK of the parameters required by the decryption algorithm, the decryption process of the selected attribute- based algorithm is as follows:

Decrypt$(PK, CT, SK) \rightarrow ck$. Decryption process is a recursive algorithm. It is in a down-top manner, and therefore need to define a recursive algorithm $DecryptNode(CT, SK, x)$.

(1) If x is a leaf node. Let $j = att(x)$. If $j \notin S$, $DecryptNode(CT, SK, x) = null$. If $j \in S$,

$$\begin{aligned} DecryptNode(CT, SK, x) &= \frac{\mathrm{e}(D_j, C_x)}{e(D'_j, C'_x)} \\ &= \frac{e(g^r \cdot H(j)^{r_j}, g^{\alpha q_x(0)})}{e(g^{\alpha r_j}, H(att(x))^{q_x(0)})} \\ &= \frac{e(g, g)^{q_x(0)\cdot\alpha r} e(H(j), g)^{q_x(0)\cdot\alpha r}}{e(g, H(j))^{q_x(0)\cdot\alpha r}} \\ &= e(g, g)^{q_x(0)\cdot\alpha r}. \end{aligned}$$

(2) If x is a non-leaf node, recursive algorithm *Decrypt Node* (CT, SK, x) be defined as follows:

For all nodes z that are children of x, it performs $\mathrm{F}_z = DecryptNode(CT, SK, z)$. Let S_x be an arbitrary k_x- *sized* child nodes set $\{z\}$, and $F_z \neq null$. If no such set exists then $F_x = null$. Otherwise, F_x is calculated as follows:

$$\begin{aligned} F_x &= \prod_{z\in S_x} F_z{}^{\Delta_{j,S'_x}(0)} \\ &= \prod_{z\in S_x} (e(g, g)^{q_z(0)\cdot\alpha r})^{\Delta_{j,S'_x}(0)} \\ &= \prod_{z\in S_x} (e(g, g)^{q_x(index(z))\cdot\alpha r})^{\Delta_{j,S'_x}(0)} \\ &= \prod_{z\in S_x} (e(g, g)^{q_x(j)\cdot\alpha r})^{\Delta_{j,S'_x}(0)} \\ &= e(g, g)^{\sum_{z\in S'_x} \alpha r\cdot q_x(j)\Delta_{j,S'_x}(0)} \\ &= e(g, g)^{\alpha r q_x(0)} \end{aligned}$$

where $j = index(z)$, $S'_x = \{index(z) : z \in S_x\}$ and $\Delta_{j,S'_x}(x) = \prod_{j\in S'_x, j\neq i} \frac{x-j}{i-j}$ is Lagrange interpolation coefficient.

Then the decryption process is as follows: it call function *DecryptNode* for the root node R of Γ. If the DU's attributes set S satisfies Γ, then

$$\begin{aligned} \mathrm{F_R} &= DecryptNode(CT, SK, R) \\ &= e(g, g)^{\alpha r q_R(0)} \\ &= e(g, g)^{\alpha r s}. \end{aligned}$$

Consequently, the content key ck can be pushed out: $ck = \tilde{C} \cdot F_R \big/ e(D, C)$.

After obtaining ck, the DU decrypts the encrypted document $E_{ck}(M)$ obtained from the cloud server through a symmetric encryption algorithm, and outputs the docu- ment M.

B. SMART CONTRACT DESIGN

In this section, we mainly describe the interface and algorithm logic related to smart contracts in our scheme. In Ethereum, smart contracts are compiled in solidity [28] and deployed on the Geth Ethereum client. In our scheme, smart contract named *StorageSC* is created and deployed.

Smart Contract construction: This Contract defines some variables of the contract and two structures when the contract is created.

1) The DO is the creator of the contract, and its Ethereum account address is recorded as owner. All Ethereum account addresses that call smart contracts are recorded as *msg.sender*.

2) The file ID hash is used as an index of file information, and is recorded as *HashFileId*. DU's Ethereum account address is used as an index of user information, recorded as *userAddress*.

3) The Smart Contract defines two structures: file information named *File* and user information named *User*. *File*

is used to store file related information, such as *HashFileId*. *User* is used to store user related information, such as *userAddress*.

There are eight function interfaces in this Smart Contract named *StorageSC* listed as follows:

Algorithm 1 setHashFildId(*HashFileId*)

```
Input: HashFileId
Output: bool
if msg.sender is not dataOwner then
    return false;
end if
HashFileIds[HashFileId] $\Leftarrow$ HashFileId;
return true;
```

Algorithm 2 checkHashFildId(*HashFileId*)

```
Input: HashFileId
Output: bool
if HashFileId is Null then
    return false;
end if
if HashFileId does not exist then
    return false;
else
    return true;
end if
```

Algorithm 3 setSecretKey *(user Address, secretKey)*

```
Input: secretKey
Output: bool
if msg.sender is not dataOwner then
    return false;
end. if
if user Address is Null then
    return false;
else
    mapping secretKey $\Rightarrow$ (userAddress);
    and add it to User(userAddress, secretKey);
end if
return true;
```

1. ***setHashFileId*(*HashFileId*)** : This function can only be executed by contract's creator *DO* and is used to store a hash $H(ID)$ of unique identifier *ID* of the file *M* uploaded by the *DO*. When the smart contract is invoked, the caller's Ethereum address will be retrieved and recorded as *msg.sender*. The *DO* then uploads the smart contract address *contractAddress*, file *ID*, and encrypted document collection $\{contractAddress, ID, E_{ck}(M)\}$ to the cloud server.

2. ***checkHashFileId*(*HashFileId*)** : This function is called by the *DU* to check whether the unique hash $H(ID)$ of the file *ID* uploaded by the *DO* exists in the Ethereum, and the output is Boolean. The *DU* can download the encrypted file in the cloud server to get the smart contract address *contractAddress*. The file *ID* hash $H(ID)$ stored in the blockchain is retrieved by the *contractAddress* to check if it exists in the Ethereum. If it does, proceed; otherwise $\perp$.

3. ***setSecretKey*(*userAddress*, *secretKey*)** : This function can only be executed by the contract's creator *DO* to store the ciphertext SK' of the private key of the *DO* assigned to *DU*. (The private key here is encrypted by the common key which is negotiated by *DO* and *DU* with Diffie-Hellman protocol). The SK' is stored in a structure named *User*. There are two attributes in the structure: a private key of a string type, and a valid access period of the string type. When storing the SK', the data owner inputs the ciphertext of the Ethereum account address *userAddress* of the *DU* and the SK' as a one-to-one mapping relationship (Each *DU* has their own private key and only needs their own account address *userAddress* to get the private key).

Algorithm 4 getSecretKey()

```
Input: null
Output: secret Key
if msg.sender is not dataUser then
    return false;
else
    mapping ([msg.sender $\Rightarrow$ secretKey);
    return secretKey;
end if
```

Algorithm 5 setCipherText(*HashFileId,CipherText*)

```
Input: HashFileId,CipherText
Output: bool
if msg.sender is not dataOwner then
    return false;
end if
if HashFileId is not exist in HashFileIds[HashFileId] then
    return false;
end if
[HashFileId].CipherText $\Leftarrow$ CipherText;
return true;
```

Algorithm 6 getCipherText(*HashFileId*)

```
Input: HashFileId
Output: CipherText
if HashFileId does not exist then
    throw;
else
    return CipherText;
end if
```

4. ***getSecretKey*()** : This function is called by the *DU* to obtain its own SK' (the ciphertext of the private key SK'). When the smart contract is invoked, the caller's Ethereum

address will be retrieved and recorded as *msg.sender*. Therefore, the call of the function outputs the SK' by a one-to-one mapping relationship of the account address of the calling contract without inputting any parameters.

5. ***setCipherText*(*HashFileId*, *CipherText*)** : This function can only be executed by the *DO* to store the path of the *CT* file obtained by the previous section of the encryption algorithm. This function needs to input the hash value *HashFileId* of the file *ID* and the path to the *CT* with the access policy. Since the *CT* exists in the form of an object, the ciphertext object is serialized into a binary file format, and its binary file path is converted into a hexadecimal format encoding storage. The ciphertext path only needs to be stored once, not limited by the number of accesses by *DU*.

Algorithm 7 setInterval (*user Address, Interval)*

```
Input: Interval
Output: bool
if msg.sender is not dataOwner then
    return false;
end if
if user Address is Null then
    return false;
else
    mapping interval ⇒ (userAddress);
    and add it to User (user Address, Interval);
end if
return true;
```

Algorithm 8 getInterval()

```
Input: null
Output: Interval
if msg.sender is not dataUser then
    throw;
else
    mapping ([msg.sender] ⇒ interval);
    return Interval;
end if
```

6. ***getCipherText*(*HashFileId*)** : This function is called by the *DU* to get the path of the *CT* stored in the Ethereum. The corresponding hexadecimal ciphertext file path is obtained by *HashFileId*, and then it needs to restore its hexadecimal to the true ciphertext file path.

7. ***setInterval*(*userAddress*, *Inerval*)** : This function can only be executed by *DO*, and *DO* adds a valid access period for the *DU* making the request. The call to this function inputs the public key address *userAddress* of the *DU* and the valid access period *Interval* for that *DU*. The hexadecimal encoding of the *Interval* is mapped to the *userAddress* in the contract, and the *userAddress* and *Interval* are added to the structure stored in the structure named *User*. If the addition was successful, returns true; otherwise returns false.

8. ***getInterval*()** : This function is called by the data consumer. The valid access period *Interval* (hexadecimal coded form) set by the data owner for himself can be queried through his own Ethereum account address. Valid access period *Interval* can be obtained by hexadecimal decoding into character type. And you can verify by code whether the current time is within the valid access period. If the current time is within the valid access period, the file is allowed to access, otherwise access is not allowed.

In our scheme, the smart contract is mainly stored by two mapping to the structure method, so that the data owner can store multiple sets of data (that is, multiple encrypted files can be uploaded). At the same time, the data user obtains the corresponding key data through the file *ID*, and obtains the corresponding effective access period through its own public key address. In some cases, the contract creator needs to terminate the smart contract to get ether in smart contract, and you need to call the self-destruct method *selfdestruct*(). When the contract is self-destructed, if someone sends the Ethereum to the contract address, then the Ethereum can no longer be redeemed and will disappear. Therefore, the smart contract in this paper cannot easily implement the self-destruction contract method to avoid economic losses. The smart contract detailed code in this article see https://github.com/xwangsharing/Storage.

VI. ANALYSIS AND EVALUATION

A. CASE EVALUATION

In our scheme, we introduces Ethereum's smart contract technology to transform the traditional attributed based encryption scheme, the key interaction between the data owner node and the data user node is realized by Ethereum network. Our new scheme no longer relies on the attribute authority to distribute the key.

1) ACCOUNT PROCESSING

All users, in this scheme, need to generate an Ethereum account EOA, which is used to create smart contracts and execute functions in contracts. Any of these users can upload encrypted files as data owners and create smart contracts. Other users need to call the method in the contract to access, obtain the key information and execute the algorithm to decrypt.

2) DATA OWNER COMPLETE CONTROL OVER DATA

After the *DO* uploads the encrypted file to the cloud server, the setup algorithm is executed to obtain the public key and the master key. After the *DO* formulates the access policy, the ciphertext is obtained through the execution of the encryption algorithm. Through the execution of the smart contract, the relevant information of the file, such as ciphertext, is stored in the blockchain network, so that the data cannot be tampered with and all of operations are transparent. If a *DU* wants to access the data, the first thing need to do is to get the decryption key, which requires the *DU* to request the *DO* to assign the attribute set. According to the *DU*'s attribute set, the *DO* generates a private key for *DU* and encrypts the private key, which is stored in the Ethereum through the smart contract. The *DU* can obtain the private key's ciphertext SK' simply by calling the contract. The private key *SK* is then decrypted by the common key which is negotiated by *DO* and *DU* with Diffie-Hellman protocol. Thereby the decryption algorithm can be executed.

In addition, The data owner adds a valid access period while assigning a attributes set to each data user. For example, as a student, you can visit the e-school library, and the school will add an expiration date to the student's attribute each semester. The effective access period for each data user's access is stored in the blockchain network through smart contract, making it impossible to tamper with and transparent. The setting of the effective access period can reduce the cost of the data user's attribute revocation to some extent.

The decryption can be successful if and only if the data consumer is within the valid access period and its set of attributes meets the access policy established by the data owner. Therefore, in this scheme, the data owner's complete control over their data is implemented.

3) KEY SECURITY

This paper transforms the traditional ciphertext-policy attribute based encryption scheme by introducing Ethereum's smart contract technology. More user access can overwhelm the pressure of a single center authority that distributes keys in the original solution. The key of the data user in this paper is assigned by the data owner rather than the center authority.

The key of the data user of the scheme is assigned by the data owner. Considering the security risks of the transmission channel, the data owner and the data user negotiate the encryption key using the Diffie-Hellman key exchange protocol before transmitting the private key of the data user. The data user obtains the encrypted private key ciphertext by paying a small amount of ether and decrypts it to obtain its own private key.

The smart contract in our scheme is implemented with the function of restricted attribute. That is, the modifier in the source code of the smart contract is implemented to restrict the calls of the smart contract to specific users. For example, a function in a smart contract that modifier is assigned to the identity of the ''onlyOwner'' can only be called by the creator of the contract. If a non-contract creator attempts to execute these methods, the execution will fail.

Therefore, in this paper, the security of key information is guaranteed.

4) LOG WITH PRIVACY PROTECTION

In this scheme, the deployment and invocation of smart contracts are recorded in the Ethereum blockchain in the form of transactions. The blockchain network is continuously synchronized, so that the transaction information of all nodes in the blockchain network is completely consistent. In other words, the introduction of Ethereum blockchain technology is equivalent to adding a log system to the access control scheme of this paper to record all access operation records. The introduction of blockchain makes all access operation records non-destructive and non-repudiation.

TABLE 3. Functional analysis and comparison.

literature	Center authority	blockchain	Access log	Access period
[30]	√	X	X	X
[4]	X	X	X	X
[20]	X	√	√	X
This paper	X	√	√	√

The execution of all functions in the smart contract is reflected in the smart contract's log file and the Ethereum blockchain network. The data owner stores information such as user information and uploaded files in ciphertext in the blockchain network. Therefore, other users are unable to use mandatory means to break the information on the blockchain. The adversary cannot obtain user privacy information from a large amount of transaction information in the Ethereum blockchain network. In the transaction information of each transaction, the transaction initiator's account address can be obtained. However, other related information of the user cannot be obtained, so that the user information is effectively protected. Therefore, the transaction information in the Ethereum blockchain network can be used as a secure access log for the user.

5) ALGORITHMIC FUNCTION ANALYSIS

The functional comparison with the attribute-based encryption scheme in recent years is shown in Table 3.

The performance of this scheme and the recent schemes are compared by whether there is a center authority, whether it is based on the blockchain, whether it has an access log, and whether the validity period is added. Document [30] is a traditional access control scheme for distributing keys by the center authority. Document [4] is a multi-authority attribute-based encryption access control scheme. Document [20] is a smart contract-based access control scheme, which achieves not only access control but also access log.

In this scheme, fine-grained access control is implemented, and the center authority is removed, so that the access control scheme is more dispersed. Thus, the problem of single point of failure brought by the center authority in the original scheme was solved. Because of the introduction of Ethereum blockchain technology, the solution is indirectly accompanied by incentives and access logs. Therefore, this solution has better applicability and usability.

B. EXPERIMENT ANALYSIS

In this section, we give the experiment analysis of our scheme. The specific configuration of the experimental platform and experimental environment is: Inter(R) Core(TM)2 Duo CPU E8400@3.00GHz processor, 4 GB RAM, and the system are Windows10 and Linux ubuntu16.04 LTS. The programming language is java and solidity. External helper is JPBC and web3j. The full name of the external auxiliary JPBC is Java Pairing-Based Cryptography. JPBC is an implementation of the java version of the bilinear pair encryption algorithm in cryptography. The main encryption algorithm in this scheme is implemented by relying on the jar package. 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, and the main encryption algorithm is implemented in the Windows10 system. The smart contract was developed by the Solidity programming language and deployed on the private chain 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. In the Maven project, the attribute encryption algorithm is written using eclipse by introducing the jar package of JPBC. By relying on some jar packages of web3j, the interaction between the data owner and the data consumer for the smart contract is realized, which makes the access control algorithm of this paper better by combining the attribute encryption algorithm with the smart contract.

Taking the literature [30] as an example, the framework is applied to the attribute-based encryption 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 smart contract is deployed on the local private chain of the Ethereum network to implement the solution in this chapter. Compared with the traditional attribute-based 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.

Since the price of the Ethereum in the Ethereum main chain is erratic, in order to facilitate the analysis of the cost of the experimental data in this section, the price of the Ethereum is set to $1\ ether \approx 200\ USD$,and let $1\ gasPrice \approx 1\ Gwei$, $1 Gwei = 10^9 wei = 10^{-9} ether$.

Table 4 lists the costs of some operations for smart contracts. The creation of a smart contract for each data owner is created only once, consuming 1,272,934 gas and cost about $0.50. After the data owner executes the key generation algorithm, the ciphertext of data user's private key is stored in the Ethereum blockchain, and the setSecretKey operation is performed, which requires a cost of about $0.36; After the data owner uploads the encrypted document, the cost of executing the setHashFileId operation is approximately $0.027; The data owner assigns a valid access period to the data consumer, and the cost of executing the setInterval operation is approximately $0.035; The data owner develops an access policy and stores the encrypted ciphertext in the Ethereum blockchain. The cost of executing the setCipherText operation is approximately $0.044.

TABLE 4. The smart contract cost (gasprice = 2 *Gwei*, 1 *ether* = 200 *USD*).

function	GasUsed	Actual Cost(ether)	USD
Contract create	1272934	0.002545868	0.5091736
setSecretKey	911964	0.001823928	0.3647856
setHashFileId	69501	0.000139002	0.0278004
setCipherText	110852	0.000221704	0.0443408
getSecretKey	41088	0.000082176	0.0164352
checkHashFileId	24133	0.000048266	0.0096532
getCipherText	25798	0.000051596	0.0103192
setInterval	87589	0.000175178	0.0350356
getInterval	28871	0.000057742	0.0115484

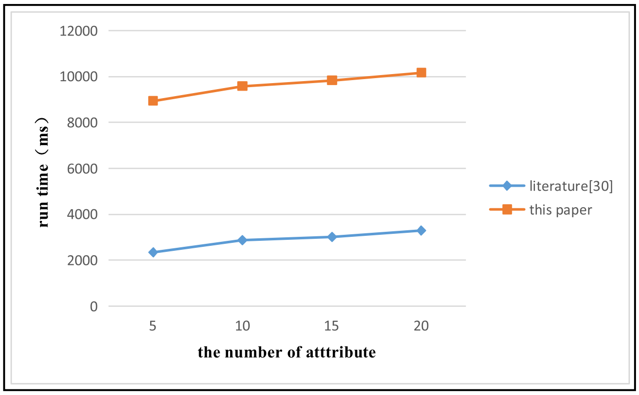

FIGURE 3. Run time of algorithm under different number of attributes.

Generally speaking, whenever the data owner uploads a file as a share, he needs to spend less than $1 to deploy a smart contract. Similarly, every time a data user accesses data, he spend about $0.048. These costs are based on prototypes deployed on the blockchain and can be reduced with optimized code. If the size of the input parameters for these functions is smaller, the cost can be further reduced.

Experiments have shown that the cost paid by data owners to share files is small and will be further reduced as data users increase. The more data users access, the more benefits the data owner receives. Of course, the cost to the data user to access the data owner's file is very little.

The abscissa in Figure 3 is the number of attributes, the number is 5, 10, 15, 20; the ordinate is expressed as the running time of the access control algorithm. The broken line of the blue diamond shape indicates the change trend of the execution time of the algorithm in the literature [30] with the increase of the attribute; and the broken line of the orange square indicates the change rule of the execution time of the algorithm as the attribute grows. Similarly, the execution time of the original algorithm increases as the attribute increases. The running time of the algorithm is almost consistent with the trend of the running time of the original algorithm. Since the framework of this chapter is based on blockchain, it is slightly higher in efficiency than the original scheme. The smart contract detailed code in this paper can be seen in https://github.com/xwangsharing/Storage.

VII. CONCLUSION

In this paper, a secure cloud storage access control framework based on blockchain is proposed. The traditional ciphertext-policy attribute-based encryption algorithm is transformed by introducing Ethereum's smart contract technology. In order to prevent the center authority from being attacked, the distribution key no longer relies on the center authority. Our scheme is decentralized. A distributed access control scheme is implemented through interaction between the data owner node and the data user node. Experiments show that the cost of accessing files is very small.

Further research work is still worth doing. This framework is based on the cloud storage platform, the cloud storage platform is semi-honest. Therefore, the program also lacks research data integrity which ensure that data owner to upload the document has not been tampered with. In the future, cloud storage platforms may be replaced with decentralized storage platforms, such as Inter Planetary File System (IPFS) [31], Storj [32], etc.

REFERENCES

[1] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2008, pp. 321–334.

[2] M. Chase, "Multi-authority attribute based encryption," in *Proc. 4th Theory Cryptogr. Conf. (TCC)*. Amsterdam, The Netherlands: Springer-Verlag, Feb. 2007.

[3] H. Lin, Z. Cao, X. Liang, and J. Shao, "Secure threshold multi authority attribute based encryption without a central authority," *Inf. Sci.*, vol. 180, no. 13, pp. 2618–2632, Jul. 2010.

[4] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 6632, K. G. Paterson, Ed. Berlin, Germany: Springer, 2011.

[5] K. Yang and X. Jia, " Attributed-based access control for multi-authority systems in cloud storage," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, Jun. 2012, pp. 536–545.

[6] J. Wei, W. Liu, and X. Hu, "Secure and efficient attribute-based access control for multiauthority cloud storage," *IEEE Syst. J.*, vol. 12, no. 2, pp. 1731–1742, Jun. 2018.

[7] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: https://bitco.in/pdf/bitcoin.pdf

[8] G. Zyskind, O. Nathan, and A. S. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *Proc. IEEE Secur. Privacy Workshops*, May 2015, pp. 180–184.

[9] M. Jemel and A. Serhrouchni, "Decentralized access control mechanism with temporal dimension based on blockchain," in *Proc. IEEE 14th Int. Conf. E-Bus. Eng. (ICEBE)*, Nov. 2017, pp. 177–182.

[10] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, "MeDShare: Trust-less medical data sharing among cloud service providers via blockchain," *IEEE Access*, vol. 5, pp. 14757–14767, 2017.

[11] R. Xu, Y. Chen, E. Blasch, and G. Chen, "BlendCAC: A smart contract enabled decentralized capability-based access control mechanism for IoT," in *Proc. IEEE Int. Conf. Blockchain*, Jul./Aug. 2018, pp. 1027–1034.

[12] K. Liu, H. Desai, and L. Kagal, "Enforceable data sharing agreements using smart contracts," 2018, *arXiv:1804.10645*. [Online]. Available: https://arxiv.org/abs/1804.10645

[13] C. Lin, D. He, X. Huang, K.-K. R. Choo, and A. V. Vasilakos "BSeIn: A blockchain-based secure mutual authentication with fine-grained access control system for industry 4.0," *J. Netw. Comput. Appl.*, vol. 116, pp. 42–52, Aug. 2018.

[14] Q. Xia, E. Sifah, A. Smahi, S. Amofa, and X. Zhang, "BBDS: Blockchain-based data sharing for electronic medical records in cloud environments," *Information*, vol. 8, no. 2, p. 44, Apr. 2017.

[15] H. G. Do and W. K. Ng, "Blockchain-based system for secure data storage with private keyword," search," in *Proc. IEEE World Congr. Services*, Jun. 2017, pp. 90–93.

[16] P. J. Lu, L.-Y. Yeh, and J.-L. Huang, "An privacy-preserving cross-organizational authentication /authorization/accounting system using blockchain technology," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[17] G. Magyar, "Blockchain: Solving the privacy and research availability tradeoff for ehr data: A new disruptive technology in health data management," in *Proc. IEEE 30th Neumann Colloquium (NC)*, Nov. 2017, pp. 135–140.

[18] S. Alansari, F. Paci, and V. Sassone "A distributed access control system for cloud federations," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, Jun. 2017, pp. 2131–2136.

[19] T.-T. Kuo, H.-E. Kim, and L. Ohno-Machado, "Blockchain distributed ledger technologies for biomedical and health care applications," *J. Amer. Med. Inform. Assoc.*, vol. 24, no. 6, pp. 1211–1220, Nov. 2017.

[20] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1594–1605, Apr. 2019.

[21] *Diffie-Hellman Key Exchange*. Accessed: Oct. 3, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

[22] W. Gavin, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project Yellow Paper, 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[23] *Litecoin*. Accessed:Jul. 20, 2018. [Online]. Available: https://litecoin.org/

[24] *Dogecoin*. Accessed:Jul. 10, 2018. [Online]. Available: http://dogecoin.com/

[25] *Ethereum Homestead Documentation*. Accessed: Dec. 2, 2018. [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/ethereum-homestead/latest/ethereum-homestead.pdf

[26] *Ethereum Blockchain App Platform*. Accessed: Feb. 1, 2018. [Online]. Available: https://www.ethereum.org/

[27] M. Ulieru, "Blockchain 2.0 and beyond: Adhocracies," in *Banking Beyond Banks and Money* (New Economic Windows), P. Tasca, T. Aste, L. Pelizzon, and N. Perony, Eds. Cham, Switzerland: Springer, 2016.

[28] C. Dannen, *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. New York, NY, USA: Apress, 2017.

[29] J. P. Cruz, Y. Kaji, and N. Yanai, "RBAC-SC: Role-based access control using smart contract," *IEEE Access*, vol. 6, pp. 12240–12251, 2018.

[30] P. Zhang, Z. Chen, K. Liang, S. Wang, and T. Wang, "A cloud-based access control scheme with user revocation and attribute update," in *Information Security and Privacy* (Lecture Notes in Computer Science), vol. 9722, J. Liu and R. Steinfeld, Eds. Cham, Switzerland: Springer, 2016.

[31] J. Benet, "IPFS—content addressed, versioned, P2P file system," 2014, *arXiv:1407.3561*. [Online]. Available: https://arxiv.org/abs/1407.3561

[32] C. Gray. (2014). *Storj Vs. Dropbox: Why Decentralized Storage is the Future*. [Online]. Available: https://bitcoinmagazine.com/articles/storj-vs-dropbox-decentralized-storage-future-1408177107

SHANGPING WANG received the B.S. degree in mathematics from the Xi'an University of Technology, Xi'an, China, in 1982, the M.S. degree in applied mathematics from Xi'an Jiaotong University, Xi'an, in 1989, and the Ph.D. degree in cryptology from Xidian University, Xi'an, in 2003. He is currently a Professor with the Xi'an University of Technology. His current research interests include cryptography and information security.

XU WANG received the B.S. degree from the School of Telecommunication Engineering, HaoJing College, Shaanxi University of Science and Technology, Xi'an, China, in 2016. He is currently pursuing the M.S. degree with the School of Computer Science and Engineering, Xi'an University of Technology, Xi'an. His research interests include information security and blockchain technology.

YALING ZHANG received the B.S. degree in computer science from Northwest University, Xi'an, China, in 1988, and the B.S. degree in computer science and the Ph.D. degree in mechanism electron engineering from the Xi'an University of Technology, Xi'an, in 2001 and 2008, respectively, where she is currently a Professor. Her current research interests include information security and privacy protection.