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.2018.DOI

A Blockchain-Based Framework for Data Sharing with Fine-grained Access Control in Decentralized Storage Systems

SHANGPING WANG[1], YINGLONG ZHANG[2]✉, 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 Science, Xi'an University of Technology, Xi'an, Shaanxi, China(e-mail: ylzhang3550@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: Yinglong Zhang (e-mail: ylzhang3550@gmail.com).

This work is supported by the National Natural Science Foundation of China under Grants No. 61572019, No. 61173192, the Key Project of Natural Science Foundation of Shaanxi Province of China under Grant No. 2016JZ001.

ABSTRACT In traditional cloud storage systems, attribute-based encryption (ABE) is regarded as an important technology for solving the problem of data privacy and fine-grained access control. However, in all ABE schemes, the private key generator (PKG) has the ability to decrypt all data stored in the cloud server, which may bring serious problems such as key abuse and privacy data leakage. Meanwhile, the traditional cloud storage model runs in a centralized storage manner, so single point of failure may lead to the collapse of system. With the development of blockchain technology, decentralized storage mode have entered the public view. The decentralized storage approach can solve the problem of single point of failure in traditional cloud storage systems and enjoy a number of advantages over centralized storage, such as low price and high throughput. In this paper, we study the data storage and sharing scheme for decentralized storage systems, and propose a framework that combines the decentralized storage system IPFS, the Ethereum blockchain and attribute-based encryption (ABE) technology. In this framework, the data owner has the ability to distribute secret key for data users, and encrypt shared data by specifying access policy, and the scheme achieves fine-grained access control over data. At the same time, based on smart contract on the Ethereum blockchain, the keyword search function on the ciphertext of the decentralized storage systems is implemented, which solves the problem that the cloud server may not return all of the results searched or return wrong results in the traditional cloud storage systems. Finally, we simulated the scheme in the linux system and the Ethereum official test network Rinkeby, and the experimental results show that our scheme is feasible.

INDEX TERMS ABE, Ethereum blockchain, smart contract, IPFS, access control, keyword searchable

I. INTRODUCTION

WITH the rapid development of internet technology, cloud storage has become an important business model in our daily life. It has provided different kinds of data storage services for individuals and enterprises, making it possible for users to access Internet resources and share data at anytime and anywhere, it has brought great convenience to our lives. Such cloud storage systems has been very successful and has gained increasing acceptance, however, as such kind of systems only depends on a large company with a strong storage capacity to store and transmit data, in which the large company is regarded as a trusted third party, it inevitably inherits the single point of failure drawback of relying on third party services. Even if cloud storage systems are backed up for data availability, cloud storage services providers may still suffer from certain factors of force Majeure (such as political censorship) lead to users will not be able to access their own data. In addition, with the development of storage technology, the cost of storage devices has become smaller and smaller. The cost of centralized cloud storage services comes mainly from employee wages, legal costs, and data center rentals, etc. These fixed

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.2018.2851611, IEEE Access

costs are unchanged or gradually increased. The price of the centralized cloud storage services will be higher.

From the above point of view, the future needs a decentralized storage approach to provide people with data storage and sharing services. It does not need to trust that third parties will honestly transmit and store data for us, nor does it need to worry about data being inaccessible. Fortunately, with the advent of Bitcoin, its underlying technology blockchain can bring an elegant implementation to such decentralized storage systems. The emergence of the blockchain has enabled us to connect the P2P cryptocurrency with storage space, bandwidth, CPU power, etc. Imagine that we can lease free extra harddisk space over the internet and get a return on cryptocurrency [1]. Users don't have to worry that they won't be able to access their own data, because the availability of data can be guaranteed by smart contracts deployed on the blockchain, and they just have to pay a regular fee for the data they have stored.

In traditional cloud storage systems, if users want to secretly share data stored in a third-party cloud server, a technology is needed to achieve access control over data that can only be accessed and decrypted by a specific user. Driven by this demand, the attribute-based encryption mechanism [2] (ABE) was proposed and rapidly developed. Through this mechanism, the data owner can specifies the data access policy based on the user's identity and attributes to achieve fine-grained access control over data. Almost all ABE encryption schemes require a trusted private key generator (PKG) to setup the system and distribute the corresponding secret key for users [3]. There are a lot of problems with such a system. Firstly, it is difficult to find a trusted PKG in reality. Secondly, such a system has the problem of the key abuse, the ownership of users data is not controlled in their own hands. The PKG has the ability to decrypt all data in the server, and PKG may leak users data for reasons such as certain interests or political censorship,etc. Once the data owner loses its own secret key, he can't even decrypt his own data, and PKG can still decrypt the users data. In practical applications, there are situations where we need data owner to have the ability to control their own data and distribute secret key for users. For example, a data manager in a hospital should be able to distribute secret key for his doctors and related staff so that they can access different levels of data according to their position.

To better protect the privacy and availability of data, we should transfer data storage and sharing from the centralized cloud storage systems to the decentralized storage systems, which have a lower prices than traditional cloud storage, the advantage of large data throughput, but also the need to stop worrying about single point of failure. In the existing decentralized storage systems, it is very simple to implement encryption and storage of data, but how to address the secretly sharing of data is an urgent problem to be solved. In this paper, we propose a framework that can achieve fine-grained access control over data in decentralized storage systems, and search the data based on the interesting keywords.

The contributions of this paper are as follow:

(1) We propose a framework that combines the decentralized storage system IPFS, the Ethereum blockchain, and attribute-based encryption(ABE) technology to achieve fine-grained access control over data in decentralized storage systems. The data owner is the only one who controls his own data, trusted PKG is not needed in our system, and the data owner has the ability to distribute secret key for data users, which is more flexible than the traditional ABE schemes. At the same time, the Ethereum blockchain is used to manage the users secret key, the problem of key management in the traditional ABE schemes is solved. When a user forgets his own secret key, he only needs to read the corresponding transactions data from the Ethereum blockchain, and decrypts it to get its own secret key information.

(2) Through the building of encrypted keyword indexs for the shared file, the encrypted keyword indexs information is stored on the Ethereum blockchain, furthermore, the smart contract is deployed on the Ethereum blockchain to implement the keyword search in the decentralized storage systems. Once the smart contract is deployed, it will operate in good faith and in accordance with the logic of the smart contract, and only if the users retrieves the correct search results, the service fee will be paid. The problem that search service providers may not return search results honestly in traditional cloud storage is solved in our scheme.

(3) Under the Ubuntu linux system, a simulation of the system scheme is carried out through the Ethereum official test network Rinkeby, and the corresponding performance and cost were analyzed.

The rest of the paper is organized as follows. Section II consist of related work, Section III review the preliminaries used throughout this paper. In Section IV, the system model is described in detail. A case study is presented in Section V, and the performance and security analysis is discussed in Section VI. Finally, the conclusion and future direction is presented.

II. RELATED WORK

A. BLOCKCHAIN TECHNOLOGY

In recent years, decentralized cryptocurrency (such as Bitcoin [4], Ethereum [5], Zcash [6], etc.) have become hot, and the blockchain technology as the underlying technology of cryptocurrency has been paid to more and more attention. Nowadays, the blockchain has been playing a major role in the financial field [7]. In addition, it has been found to be useful in many other non-financial fields. Such as: decentralized supply chain [8], identity-based PKI [9], decentralized proof of document existence [10], decentralized IOT [11], decentralized storage [12] [13] [14], etc.

In order to allow data owner to own and control their own data, a personal data management system based on blockchain technology [15] has been proposed, the system can get better protection of the privacy of users data. In order to solve the problem of data privacy in IOT systems, a blockchain architecture system for IOT [16] was proposed, in which attribute-based encryption (ABE) technology is used

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.2018.2851611, IEEE Access

to implement data access control. For the purpose of solving the security and privacy issues that hinder the development of big data, a blockchain-based access control framework for enhancing the security of big data platforms [17] was proposed.

Decentralized storage systems (such as IPFS [13], Storj [12], Sia [18], etc.) do not rely on a central service provider, allowing users to store files to storage nodes that rent out free storage space. These systems use blockchain as their core structure. As a content-addressed decentralized storage platform, IPFS uses Filecoin [14] as an incentive layer to incent nodes to provide storage and retrieval services. We noticed that IPFS does not provide a strong privacy cryptographic algorithm interface for user-uploaded files. In contrast, the Storj platform provides an end-to-end encryption approach, and stores cryptographic hash fingerprint of file on the blockchain while providing a method of verifying file integrity. The Sia platform combines blockchain technology with a peer-to-peer storage network, splits the uploaded file into multiple file segments, and encrypts each segment. The file ciphertext is sent to the nodes that provide storage service through smart contracts. The users pays Siacoin for the storage service, and the storage nodes periodically submits a file proof of storage to prevent the storage node from deleting the stored file.

B. ATTRIBUTE-BASED ENCRYPTION TECHNOLOGY

In traditional cloud storage, attribute-based encryption technology [2] can achieve fine-grained access control over data. In this technique, attributes are used instead of identities. Data owner can assign the users groups that can access the data by setting an access policy. Only the users whose attributes set meet the access policy can access the data. Since attribute-based encryption technology was proposed, many research works have been done in many ways driving by actual needs, and have achieved many significant research results. For example, in a practical application, if the users attributes change, the corresponding users secret key must also be changed accordingly, so attribute revocable attribute-based encryption schemes [19] [20] [21] [22] are proposed; as access policies may be revealed important privacy information of users, attribute-based encryption schemes with hidden access policy [23] [24] were proposed; in many commercial application scenarios, multiple attribute authorities are required for attribute distribution and management, so multi-authority attribute-based encryption schemes [25] [26] [27] has been developed. In addition, with the widespread application of mobile devices with limited computing and storage resources, outsourced decryption in attribute-based encryption schemes [28] [29] are proposed.

At present, attribute-based encryption technology has been well developed and applied in traditional cloud storage systems, but there is no research project to achieve fine-grained access control over data in decentralized storage systems.

TABLE 1. Notations table

Notation	Description
DO	data owner
DU	data users
PK	system public parameter
MK	system master key
$\mathcal{S}$	user attributes set
$\mathcal{P}$	access policy
$h_{location}$	file location
SK_d	user attributes secret key
SK_s	user search secret key
kw	keyword set
CT_F	encrypted file
CT_K	encrypted symmetric key
CT_{usk}	encrypted user secret key
CT_l	encrypted file location
CT_{md}	encrypted metadata
$index$	encrypted keyword indexs
$token$	search token
Enc_K	AES encryption algorithm, K is symmetric key
Dec_K	AES decryption algorithm, K is symmetric key

C. KEYWORD SEARCHABLE TECHNOLOGY IN DECENTRALIZED STORAGE SYSTEMS

In 2000, Song et al. [30] first proposed a one-to-one symmetric searchable encryption mechanism. In traditional cloud storage systems, searchable encryption technology has obtained a lot of research results. Such as: public key searchable encryption [31], searchable encryption supporting multiple keywords [32] [33] and so on.

After the emergence of the blockchain, many scholars began to study how to transfer the traditional data storage and sharing to the decentralized storage systems. Li et al. [34] proposed a searchable symmetric encryption scheme using the Bitcoin blockchain system. In this scheme, users data and encrypted keyword indexes are split and stored on the blockchain, but a large amount of data storage will cause the expansion of the blockchain, and it cannot achieve fine-grained access control over data. Cai et al. [35] proposed a trustworthy and privacy keyword search scheme in encrypted decentralized storage. The scheme combines an efficient dynamic searchable encryption scheme with blockchain to solve the problem of fairness between the client and the service nodes. Hoang et al. [36] proposed a decentralized storage system using blockchain technology and a privacy keyword search scheme in the system. Jiang et al. [37] proposed a blockchain-based privacy keyword search scheme in decentralized storage, which implements oblivious keyword search in the decentralized storage systems. In these schemes, the relatively mature decentralized storage systems has not been used, and these schemes are still in the theoretical stage. The feasibility and stability have not been validated.

III. PRELIMINARIES

In this section, we review some of the relevant background knowledge and related knowledge that will be used in this paper. Table. 1 presents some of the notations used throughout the paper.

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.2018.2851611, IEEE Access

A. ATTRIBUTE-BASED ENCRYPTION

1) Bilinear mapping

Let $\mathbb{G}_0$ and $\mathbb{G}_T$ be cyclic multiplicative group of big prime order p , g be a generator of $\mathbb{G}_0$, and we denote the identity of $\mathbb{G}_T$ as 1. We call $\hat{e} : \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_T$ a bilinear pairing [38] if $\hat{e} : \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_T$ is a map with the following properties:

a) **Bilinear**: $\hat{e}(g^a, g^b) = \hat{e}(g, g)^{ab}$ for all $a, b \in \mathbb{Z}_p^*$.

b) **Non-degenerate**: There exists $g_1, g_2 \in \mathbb{G}_0$, such that $\hat{e}(g_1, g_2) \neq 1$.

c) **Computable**: There is an efficient algorithm to compute $\hat{e}(g_1, g_2)$ for all $g_1, g_2 \in \mathbb{G}_0$.

2) Access Policy

An access policy [38] $\mathcal{W}$ in ABE is a rule that returns either 0 or 1 given a attributes set $\mathcal{L}$. We say that $\mathcal{L}$ satisfies $\mathcal{W}$ if and only if $\mathcal{W}$ answers 1 on $\mathcal{L}$. Usually, notation $\mathcal{L} \vDash \mathcal{W}$ is used to represent the fact that $\mathcal{L}$ satisfies $\mathcal{W}$, and the case of $\mathcal{L}$ does not satisfy $\mathcal{W}$ is denoted as $\mathcal{L} \nvDash \mathcal{W}$. In our construction, we consider AND-gate policy AND_m^*. Formally, given an attribute list $\mathcal{L} = [L_1, L_2, \cdots, L_n]$ and an access policy $\mathcal{W} = [W_1, W_2, \cdots, W_n] = \wedge_{\mathcal{I}_W} W_i$, where $\mathcal{I}_W$ is a subscript index set and $\mathcal{I}_W = \{i | 1 \leq i \leq n, W_i \neq *\}$, we say $\mathcal{L} \vDash \mathcal{W}$ if $L_i = W_i$ or $W_i = *$ for all $1 \leq i \leq n$ and $\mathcal{L} \nvDash \mathcal{W}$ otherwise. It is noted that the wildcard * in $\mathcal{W}$ plays the role of "don't care" value. For example: if we have access policy $\mathcal{W} = [Hospital.A, Doctor, *, China]$, attribute set $\mathcal{L}_1 = [Hospital.A, Doctor, male, China]$ and $\mathcal{L}_2 = [Hospital.A, Nurse, male, China]$, then $\mathcal{L}_1 \vDash \mathcal{W}$, $\mathcal{L}_2 \nvDash \mathcal{W}$.

B. BLOCKCHAIN TECHNOLOGY AND ETHEREUM

1) Blockchain technology

In 2009, Bitcoin [4] came into the world as a decentralized cryptocurrency, its enabler blockchain technology attracting a lot of attention. In the last few years, academia and industry have conducted extensive research on it and found that it not only can be used as a decentralized payment system, but also can be used in many industries [39] (e.g., financial, medical , public utilities, identity management, asset registration, government agencies, etc.) to play an important role.

The blockchain is a distributed database that records all the transactions that have occurred in the peer-to-peer network. All participants in the network hold the same copy of the database. In this network, there is no central authority and there is no single node can control the entire network. As shown in Fig. 1, the blockchain is essentially a series of linked data blocks. Blocks are added to the blockchain by a consensus among most nodes in the system. Each block contains a block header and a series of transactions, each block header contains the link pointers of the block headers of the previous block, the merkle root of the tree-like transaction information, and a timestamp. In this way, the blocks are linked together in chronological order. The cryptography hash algorithm ensures that the transaction data in each block is immutable, and the linked blocks in the blockchain cannot be tampered.

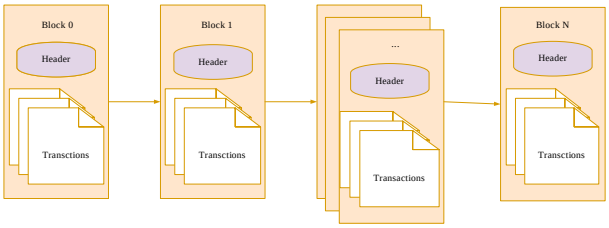

FIGURE 1. Blockchain structure

2) Ethereum

Ethereum [5] [40] [41] is a new decentralized application platform of smart contract, and it is developed from Bitcoin. In the Bitcoin system, the transaction operations are performed by a stack-based non-Turing complete scripting language, which can only support simple logic, thus limiting its application in many fields. Ethereum is usually described as a Bitcoin-like programmable system with the Turing complete scripting language. Compared to the Bitcoin system, its innovation is that it is a programmable blockchain. The platform supports the application of Turing complete, allowing users to create, deploy and run smart contracts on the blockchain. Once the contract is deployed, it can be automatically executed according to the agreed logic of smart contracts. Ideally, there is no downtime, censorship, fraud, and third-party intervention and other issues. We briefly summarize the main elements of the Ethereum platform as follows:

Ethereum account: In the Ethereum platform, there are two main types of accounts: External Owned Accounts (EOA) and Contract Accounts, both of which are identified by a 20-bytes hexadecimal string, such as: 0x-e4874f5a6077b6793de633f52b5ff112aec012de. The EOA is controlled by the external users private key. It has a corresponding balance and Nonce field. It can send a transaction to transfer the ether to another address or trigger the execution of the contract code. The contract account is controlled by the code stored in the account. In addition to the balance and Nonce, it also has a code and storage space associated with it. In this paper, we stored the encrypted keyword indexs into the contract account's storage space.

Ether and Gas: Ether are the token used in the Ethereum platform. The Ethereum blockchain implements a smart contract running environment, known as the Ethereum Virtual Machine (EVM). When the code in the contract is triggered by a transaction or message, each miner node in the network (the miner node receives, broadcasts, verifies, and executes the transactions in the Ethereum network) runs EVM as part of the block validation protocol. They will verify each transaction covered in the block and run the code triggered by transaction in EVM, perform the same calculations and store same values. Each operation in the EVM has a specific consumption, which is counted by the unit gas. Gas can be purchased through the ether, and the sender of the transaction needs to pay for ether for the operations he wants to perform

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.2018.2851611, IEEE Access

(computing, data storage, etc.). The actual transaction cost is calculated as ether=Gas Used * Gas Price.

Transactions and Messages: An Ethereum transaction is a signed data packet that allows you to transfer ether from one account to another. In addition to transferring ether, it is also possible to trigger the execution of the code in the smart contract through the transaction. A transaction consists of account Nonce, address of recipient, Gas price, Gas limit, ether value transferred, sender's signature, optional additional data fields.

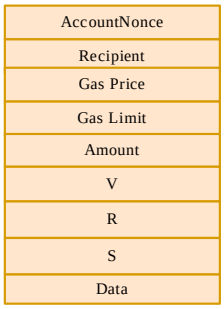

FIGURE 2. Ethereum transaction data structure

In the transaction structure shown in Fig. 2, the Data field can be put into any data that the transaction sender wants to put. In the scheme, we will attach some key data, such as the ciphertexts of users attributes secret key and file location, to the transaction and store it on the Ethereum blockchain, forming an unchangeable record. These key data are assembled in JSON format and then encoded as hexadecimal embedded into the Data field of the transaction. The data can be read and parsed through the JSON API interface eth_getTransactionByHash provided by Ethereum official.

An Ethereum message is similar to Ethereum transaction, except that the message can only be sent from one contract to another. Through the sending of a message, it is also possible to trigger the execution of the relevant code in the receiving contract account.

Mining and proof of work: On the Ethereum blockchain network, mining is a process that includes new blocks to the blockchain. Miners use proof of work (POW) algorithms to add blocks to the blockchain by packaging newly generated transactions. The POW algorithm in the Ethereum is called Ethash (a modified Dagger-Hashimoto algorithm). The difference between this algorithm and the POW algorithm in the Bitcoin system is that it is memory-intensive and largely eliminates the threat of computational power centering. The miner solves a very difficult cryptography puzzle by constantly trying to assemble blocks and new random numbers until the correct random number is found. The miner will broadcast this block to the network, other nodes will verify the correctness of the block and the transactions it contains. After the verification is passed, the block is added to the blockchain. The security of the Ethereum platform depends on the mining and POW algorithm. Mining has enabled the nodes in the entire Ethereum network to reach consensus on tamper resistance. None of the participants can deceive other nodes.

Smart contracts: Smart contracts [5] [41] are a kind of computer protocols that can be self-executed and self-verified once formulated and deployed without the need for human intervention. From a technical point of view, a smart contract can be considered as a computer program that can autonomously performs all or part of the contract-related operations and produces corresponding evidence that can be verified to demonstrate the effectiveness of the contract operation. Before the smart contract was deployed, all related logic processes associated with the contract were already established. In the Ethereum blockchain, a smart contract is a special account with associated code. The deployment process is shown in Fig. 3. After the smart contract is compiled into EVM byte code and deployed on the Ethereum blockchain, the contract address and Application Binary Interface(ABI) need to be recorded, and we can interact with the contract through the contract address and ABI. In the scheme, we use smart contracts to store encrypted keyword indexs and some related data, and guarantee the fairness of search. In the environment that does not require a trusted third party, the users deposits the service fee into the contract, and the smart contract helps users to retrieve. Only when the correct result is retrieved, the service fee will be deducted from the contract. It solves the problem that searcher deliberately not returning the result or returning the wrong result in order to save resources in traditional cloud storage schemes.

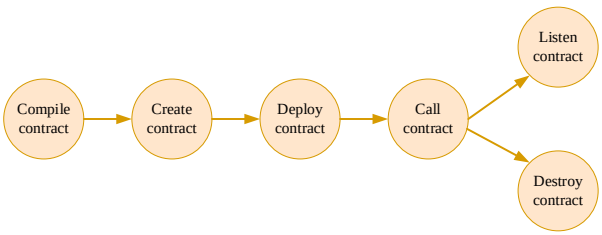

FIGURE 3. Smart contract deploy process on the Ethereum blockchain

C. IPFS

Interplanetary File System(IPFS) [13] is a peer-to-peer distributed file system that seeks to connect all computing devices with the same system of files. IPFS provides a high-throughput content-addressed block storage model, with content-addressed hyperlinks. It combines technologies such as distributed hash tables (DHT), an incentivized block exchange, and a self-certifying namespaces, etc. At the same time, IPFS has no single point of failure, and nodes do not need to trust each other. The advantage of IPFS over existing cloud storage is that there is no central server, and the data is distributed and stored in different places of the world. In some way, the use of IPFS is similar to the way we use the Web today. Uploading a file to the IPFS system will obtain a unique file cryptographic hash string through which file can

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.2018.2851611, IEEE Access

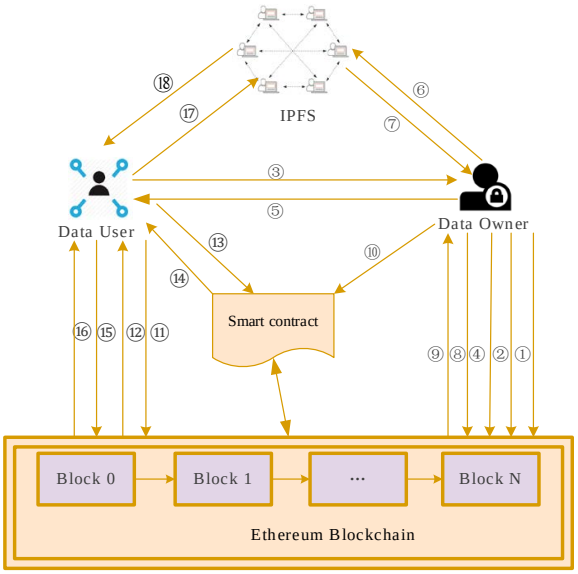

FIGURE 4. System framework

be retrieved. The hash string can be understood as a Uniform Resource Locator (URL) in the Web. We will refer to the hash string as the file location in the following. In practical applications, blockchain are not suitable for storing large files (video, audio, etc.) due to block bloated and transaction fees. In the scheme, therefore, we store the encrypted file in the IPFS. A few metadata are stored on the Ethereum blockchain. Only when the users attributes set meets the access policy defined by the data owner, the users will able to read the data from the Ethereum blockchain and decrypt the file location, download the encrypted file from the IPFS via the file location, and then decrypt it.

IV. SYSTEM MODEL

The system consists of the following two entities:

Data owner(DO): DO is a person or organization that owns a series of files to share.

Data user(DU): DU are $DO's$ data clients that are authorized to view some of the file.

The miners on the Ethereum blockchain and the storage nodes in IPFS are not considered here. The double arrow pointing to the smart contract indicates that the smart contract is deployed on the blockchain. To illustrate this clearly, draw it separately. The system framework is shown in Figure 4:

The corresponding description of each step number in the Fig. 4 is shown as follows:

① DO setups the system. The system master key is encrypted and then it is embedded into an Ethereum transaction.

② DO deploys a smart contract on the Ethereum blockchain.

③ DU sends a registration request to DO.

④ DO generates secret key for DU, and uses the shared key to encrypt the secret key and embed the encrypted secret key into an Ethereum transaction.

⑤ DO sends the transaction id related to secret key, smart contract address, smart contract ABI and smart contract source code to DU through a secure channel.

⑥ DO selects a keyword set from the shared file, and uses AES algorithm to encrypt the file and uploads it to IPFS.

⑦ DO records the file location returned by IPFS.

⑧ DO uses a selected AES key K to encrypt the file location, and uses a selected ABE algorithm to encrypt AES key K. Then selects a AES key K_1 to encrypt these information and embeds it into an Ethereum transaction.

⑨ DO records the id of the Ethereum transaction and AES key K_1.

⑩ DO generates encrypted keyword indexs and stores it to the smart contract.

⑪ DU reads transaction data related to secret key from the Ethereum blockchain.

⑫ DU uses the shared key to decrypt transaction data, then gets secret key.

⑬ DU generates search token and invokes the smart contract.

⑭ The smart contract searches according to the token and returns the relevant results.

⑮ DU reads relevant transaction data based on search results returned by smart contracts;

⑯ DU decrypts the transaction data.

⑰ DU downloads encrypted file from IPFS.

⑱ DU decrypts the encrypted file.

In our work, the system model consists of six algorithms given below:

- $\mathbf{Setup}(1^\lambda) \rightarrow (PK, MK)$: The system setup algorithm is run by DO. It takes as input security parameter λ. It outputs system public parameter PK and system master key MK. Since PK is public, DO can publish it on media(e.g., website or public database). DO encrypts the system master key, and embeds it into an Ethereum transaction. DO deploys a smart contract on the Ethereum blockchain. The contract is used to store encrypted keyword indexs and provide search service for DU. As shown in step ①② of the Fig. 4 of system framework.
  When DU sends a registration request to DO, DO first authenticates the identity of DU. After the identity is satisfied, DU is assigned the corresponding attributes set $\mathcal{S}$. The $DU's$ Ethereum account address is added as an authorized user in the smart contract. Then, DO generates secret key for as follows:
- $\mathbf{UserRegistration}(MK, \mathcal{S}) \rightarrow (SK_s, SK_d)$: The user registration algorithm is run by DO. It takes as inputs system master key MK and $DU's$ attributes set $\mathcal{S}$. It outputs secret key SK_s and SK_d. The $DU's$ secret key is encrypted using AES algorithm and embedded into an Ethereum transaction (where the encryption key is the shared key generated by the Diffie-Hellman key exchange protocol [42]), then DO sends the transaction id and smart contract address, smart contract ABI, smart contract source code to DU through a secure channel.

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.2018.2851611, IEEE Access

As shown in step ③④⑤of the Fig. 4 of system framework.

- **Encrypt**: The encrypt algorithm is run by DO. It consists of the following three sub-algorithms.
  1) $\mathbf{FileEncrypt}(F) \rightarrow (CT_F, K, kw)$: The file encrypt algorithm takes as input the shared file F. It outputs file ciphertext CT_F, file encryption key K and a keyword set kw. DO selects a keyword set kw from the file , selects AES key K from the key space, encrypts the file F, uploads the ciphertext CT_F to IPFS, and records the file location $h_{location}$ returned by IPFS. As shown in step ⑥⑦ of the Fig. 4 of system framwork.
  2) $\mathbf{KeyEncrypt}(PK, K, h_{location}, \mathcal{P}) \rightarrow CT_{md}$: The key encrypt algorithm takes as input system public parameters PK, file encryption key K, file location $h_{location}$ and access policy $\mathcal{P}$. It outputs ciphertext CT_{md}. DO uses K to encrypt $h_{location}$ as CT_l, and uses a selected ABE algorithm to encrypt file encryption key K, that is, it uses system public paremeters PK and access policy $\mathcal{P}$ to encrypt K as CT_k. DO randomly selects AES key K_1 from key space to encrypt CT_l and CT_k as CT_{md}, embed these encrypted information into an Ethereum transaction, record the transaction id and K_1 after the transaction was approved. As shown in step ⑧⑨ of the Fig. 4 of system framework.
  3) $\mathbf{IndexGen}(kw, MK) \rightarrow index$: The index generation algorithm takes as input a keyword set kw and system master key MK. It outputs encrypted keyword indexs $index$. DO builds encrypted keyword indexs $index$ based on the keyword set kw, and stores $index$ to the smart contract. As shown in step ⑩ of the Fig. 4 of system framework.
- $\mathbf{TokenGen}(\widetilde{kw}, SK_s) \rightarrow token$: The token generation algorithm is run by DU. It takes as input a keyword $\widetilde{kw}$ and secret key SK_s. It outputs search token $token$. DU reads the transaction data associated with its own secret key from the Ethereum blockchain, and decrypts it to obtain secret key SK_s and SK_d. DU generates search token $token$ based on the keyword of interested $\widetilde{kw}$, and invokes smart contract to search. As shown in step ⑪⑫⑬ of the Fig. 4 of system framework.
- $\mathbf{Test}(token, index) \rightarrow result$: The match test algorithm is run by smart contract. It takes as input search token $token$ and encrypted keyword indexs $index$. It outputs matched result $result$. The smart contract will match according to the DU passing arguments $token$. If the match is success, the set of relevant transactions id and the set of corresponding key matching the success will be returned. As shown in step ⑭ of the Fig. 4 of system framework.
- $\mathbf{Decrypt}(CT_l, CT_k, SK_d, PK) \rightarrow F$: The decrypt algorithm is run by DU. It takes as input file location ciphertext CT_l, key's ABE ciphertext CT_k, $DU's$ secret key SK_d, the system public parameter PK. It outputs original file F. DU reads relevant transactions data from the Ethereum blockchain based on the search result returned by the smart contract and decrypts it to obtain CT_l and CT_k. If the attributes set meets the access policy $\mathcal{P}$, DU decrypts CT_k to recovery the key K, and uses K to decrypt CT_l recovery file location $h_{location}$, and downloads the encrypted file CT_F from the IPFS based on $h_{location}$, and uses K to decrypt CT_F to obtain the original file F. As shown in step ⑮⑯⑰⑱ of the Fig. 4 of system framework.

V. CASE STUDY

Our proposed framework can be directly used by combining with most symmetric searchable encryption schemes and ABE schemes to achieve fine-grained access control in decentralized data storage and sharing. Since there is a corresponding cost for storing data on the Ethereum blockchain, it should be noted that the ABE ciphertext of the scheme should be as short as possible when selecting the ABE scheme, which can reduce corresponding costs. In the keyword search function, in order to make smart contracts perform as few calculations as possible, most symmetric searchable encryption schemes with less computation can be used, such as the searchable encryption scheme of inverted index type used in [43].

In the case study, in order to prevent anyone from being able to call smart contracts for search, we have added a set of authorized users in the smart contracts. Unauthorized user's search requests will be rejected by smart contracts. In order to verify the feasibility of the scheme, we modified the scheme of in [38] to test. The scheme in [38] supports the AND gate access policy for multiple attribute values and wildcards, and the ciphertext length is a constant size. At the same time, our scheme is designed to ensure the fairness of search by the smart contract to execute search process. At present, as the cost of bilinear pairing operation in the smart contract is very expensive, therefore, the search in the smart contract is more practical to choose less calculation of symmetric searchable encryption algorithm. For a simple testing, our keyword search algorithm is built with an inverted index for keyword.

A. CONCRETE CONSTRUCTION

- $\mathbf{Setup}(1^\lambda) \rightarrow (PK, MK)$: Let $\mathbb{G}_0$ and $\mathbb{G}_T$ be two cyclic multiplicative groups of big prime order p. Suppose g is a generator of $\mathbb{G}_0$ and $\hat{e} : \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_T$ is a bilinear map. Assume there are n attributes in universe and the attribute set is $\mathcal{U} = \{\omega_1, \omega_2, \cdots, \omega_n\}$. Each attribute has multiple values, and suppose $S_i = \{v_{i,1}, v_{i,2}, \cdots, v_{i,n_i}\}$ is the multi-value set for ω_i and $|S_i| = n_i$. Define two collision-resistant hash functions $H_0 : \mathbb{Z}_p^* \times \{0,1\}^{log_2 n} \times \{0,1\}^{log_2 m} \rightarrow \mathbb{Z}_p^*$ and $H_1 : \mathbb{Z}_p^* \rightarrow \mathbb{G}_0$. Define a pseudorandom function $F : \{0,1\}^* \times \{0,1\}^\lambda \rightarrow \{0,1\}^l$. Also, DO chooses $x, y \in_R \mathbb{Z}_p^*$, and computes $X_{i,k_i} = g^{-H_0(x||i||k_i)}, Y_{i,k_i} =$

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.2018.2851611, IEEE Access

$\hat{e}(g,g)^{H_0(y||i||k_i)}$ for $1 \leq i \leq n$ and $1 \leq k_i \leq n_i$. Finally, the system public parameter is published as $PK = <p, g, \{X_{i,k_i}, Y_{i,k_i}\}_{1\leq i\leq n, 1\leq k_i\leq n_i}>$, and the master key is $MK = <x, y>$. DO encrypts the system master key $MK =< x, y >$ by using its own Ethereum account public key, and embed it into transaction TX_{mk}, then broadcast TX_{mk} to the Ethereum blockchain. When TX_{mk} are included in the Ethereum blockchain block, DO compiles and deploys a smart contract for storing encrypted keyword indexs and performing search operations. After the contract is successfully deployed, record the contract account address and the ABI of the contract.

- **UserRegistration**$(MK, \mathcal{S}) \rightarrow (SK_s, SK_d)$: When a user sends a registration request, he need to submit his own Ethereum account public key. The DO authenticates the user's identity and assigns appropriate attributes set $\mathcal{S}$ for the user, then adding the user's account address to the set of authorized users the smart contract. DO chooses $sk \in_R \mathbb{Z}_p^*$ for the user. Then for $1 \leq i \leq n$, suppose $\mathcal{S}_i = v_{i,k_i}$, DO computes:

$$\widehat{\sigma}_i = \sigma_{i,k_i} = g^{H_0(y||i||k_i)} H_1(sk)^{H_0(x||i||k_i)},$$

  Finally, the corresponding attribute secret key is $SK_d = <sk, \{\widehat{\sigma}_i\}_{1\leq i\leq n}>$, search secret key is $SK_s = <K_s>$, $K_s \in \mathbb{Z}_p^*$, K_s is same for different user. DO calculates the shared key based on the Ethereum account public key submitted by the user(Diffle-Hellman key exchange protocal [42]), and obtains CT_{usk} by using AES algorithm to encrypt SK_d and SK_s. DO embedes CT_{usk} into the transaction TX_{usk}, and broadcasts TX_{usk} to the Ethereum blockchain. DO transmits his Ethereum account public key, transaction id, user's attributes set, smart contrat address, smart contract ABI and smart contract source code to user through a secure channel. User can verify the smart contract that deployed on the Ethereum blockchain(Verification through the Verify Contract Code provided by Etherscan [44]).
- **Encrypt**:
  1) **FileEncrypt**$(F) \rightarrow (CT_F, K, kw)$: DO selects a keyword set kw from the file F, randomly selects AES key K from the key space, computes CT_F=$Enc_K(F)$, where $Enc_K(F)$ denotes using AES algorithm to encrypt F, the encryption key is K. DO uploads the ciphertext CT_F to IPFS, records the file location $h_{location}$ returned by IPFS.
  2) **KeyEncrypt**$(PK, K, h_{location}, \mathcal{P}) \rightarrow CT_{md}$: DO computes CT_l=$Enc_K(h_{location})$. In order to encrypt the AES key K under the access policy $\mathcal{P}$, DO computes

$$<X_{\mathcal{P}}, Y_{\mathcal{P}}> = < \prod_{i\in\mathcal{I}_{\mathcal{P}}} \hat{X}_i, \prod_{i\in\mathcal{I}_{\mathcal{P}}} \hat{Y}_i >,$$

     where $<\widehat{X}_i, \widehat{Y}_i> = <X_{i.k_i}, Y_{i,k_i}>$, $\mathcal{I}_{\mathcal{P}}$ is a subscript set of $\mathcal{P}$. Then, DO randomly selects $s \in_R \mathbb{Z}_p^*$, computes $CT_K = <\mathcal{P}, C_0, C_1, C_2>$, where $C_0 = K \cdot Y_{\mathcal{P}}^s$, $C_1 = g^s$, $C_2 = X_{\mathcal{P}}^s$. DO randomly selects AES key K_1, computes $CT_{md} = Enc_{K_1}(CT_K, CT_l)$, embedes CT_{md} into transaction TX_{ct}, and broadcasts TX_{ct} to the Ethereum blockchain. when transaction has been approved, record the transaction id $txid$ and corresponding key K_1. The length of $txid$ and K_1 is l.
  3) **IndexGen**$(kw, MK) \rightarrow index$: DO computes $h_i = F(kw_i||0, K_s)$, $d_i = F(kw_i||1, K_s)$, $txid_i = d_i \oplus txid$, $K_{1_i} = d_i \oplus K_1$ for $kw_i \in kw$. DO stores $index = (h_i, txid_i, K_{1_i})$ to smart contract.
- **TokenGen**$(\widetilde{kw}, SK_s) \rightarrow token$: DU computes the shared key based on $DO's$ Ethereum public key. DU reads transaction data from Ethereum blockchain, decrypts it to obtain secret key SK_s and SK_d, then computes $token = F(\widetilde{kw}||0, K_s)$. DU invokes smart contract with $token$ as arguments.
- **Test**$(token, index) \rightarrow result$: The smart contract first judges whether the user is an authorized user, and if not, rejects the search; otherwise, the search result $result = (S_{txid}, S_{k1})$ is performed according to the $token$, here S_{txid} is encrypted $txid$ set and S_{k1} is corresponding encrypted K_1 set. Return $result$ to user.
- **Decrypt**$(CT_l, CT_k, SK_d, PK) \rightarrow F$: DU computes $d = F(\widetilde{kw}||1, K_s)$, computes $txid_j = d \oplus \widetilde{txid_j}$, $K_{1_j} = d \oplus \widetilde{K_{1_j}}$ for $\widetilde{txid_j} \in S_{txid}$, $\widetilde{K_{1_j}} \in S_{k1}$. DU reads transaction $txid_j$ data from the Ethereum blockchain, and computes $(CT_K, CT_l) = Dec_{K_{1_j}}(CT_{md})$, where $Dec_{K_{1_j}}(CT_{md})$ denotes using AES algorithm to decrypt CT_{md}, the decryption key is K_{1_j}. DU first checks whether his attributes set $\mathcal{S} \vDash \mathcal{P}$, and if not, returns $\perp$ and reads the next transaction data; otherwise, DU computes $\sigma_{\mathcal{P}} = \prod_{i\in I_{\mathcal{P}}} \widehat{\sigma}_i$, then AES key K is recovered as computes

$$K = \frac{C_0}{\hat{e}(\sigma_{\mathcal{P}}, C_1) \cdot \hat{e}(H_1(sk), C_2)},$$

  where $\widehat{\sigma}_i = \sigma_{i,k_i} = g^{H_0(y||i||k_i)} H_1(sk)^{H_0(x||i||k_i)}$. DU computes $h_{location}$=$Dec_K(CT_l)$, then downloads CT_F from IPFS based on $h_{location}$, computes F=$Dec_K(CT_F)$ to recovery original file F.

The security proof and correctness verification of the ABE scheme please refer to literature [38].

B. SMART CONTRACT DESIGN

In this section we mainly introduce the smart contract related interface and algorithm logic used in this paper. In the Ethereum smart contract is programmed by solidity language [45], there are special variables and functions which always exist in the global namespace and are mainly used to provide information about the blockchain.

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.2018.2851611, IEEE Access

In this paper, we mainly use the following special variables:

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

msg.value: number of wei sent with the message. For subsequent use we use $msg.value to represent the number of wei attached to a message and $cost to represents a fixed number of wei. $1\ ether = 10^{18}\ wei$.

tx.origin: sender of the transaction (full call chain). When an external own account EOA calls the smart contract and another smart contract is called in the smart contract, a call chain is formed, saying that the EOA is tx.origin.

dataSharing contract

The smart contract is deployed by DO and we call it the dataSharing contract.

dataSharing contract initialization: This process defines some variables of the contract when the contract is created.

1) The dataOwner variable of address types, which defines the address of the DO.
2) The authorizedUsers variable of mapping types, which defines a mapping collection from authorized user address to a bool value. the DO can add, modify, delete the collection through the relevant function interfaces of the contract.
3) The Index variable of mapping types, which defines a mapping collection from encrypted keyword indexs to related information. the DO can add, modify, delete the collection, and the authorized user can read the collection through relevant function interfaces of the smart contract.

The dataSharing contract mainly provides the following seven function interfaces:

addUser(newUserAddress): This function can only be executed by the contract's creator(DO). Each time the user sends a registration request to DO along with his identity certificate(this can be done with a secure out-of-band channel), after authenticating the user's identity, the user's EOA is authorized through this function.

Algorithm 1: addUser

```
Input: newUserAddress
Output: bool
if msg.sender is not dataOwner then
    throw;
end
if newUserAddress has exist then
    return false;
else
    authorizeUsers[newUserAddress] ⇐ true;
    return true;
end
```

removeUser(oldUserAddress)): This function can only be executed by the contract's creator(DO). When the system needs to remove a user, DO remove the user from the authorized set by passing the user's EOA to the function.

Algorithm 2: removeUser

```
Input: oldUserAddress
Output: bool
if msg.sender is not dataOwner then
    throw;
end
if oldUserAddress hasn't exist then
    return false;
else
    authorizeUsers[oldUserAddress] ⇐ false;
    return true;
end
```

addIndex(keywordIndex,txid,key1): This function can only be executed by the contract's creator(DO). When DO newly uploads some files to IPFS, he selects a keyword set from each file and builds the corresponding encrypted keyword indexs, storing it to the smart contract. The first arguments of the function is the encrypted keyword indexs keywordIndex , the second arguments is the transaction id txid, and the third arguments is the encryption key key1.

deleteFile(keywordIndex, txid): This function can only be executed by the contract's creator(DO). When it is necessary to delete a certain file, the encrypted keyword indexs keywordIndex of the file and the transaction id txid associated with the file are passed.

Algorithm 3: addIndex

```
Input: keywordIndex, txid, key1
Output: bool
if msg.sender is not dataOwner then
    throw;
end
mapping keywordIndex to (txid, key1), and add it to
Index variable collection
return true;
```

deleteKeyword(keywordIndex):This function can only be executed by the contract's creator(DO). When it is necessary to delete all file corresponding to a keyword, the function passes in the encrypted keyword indexs keywordIndex.

search(keywordIndex):This function can only be executed by the user in the authorized set and the creator(DO) of the contract. The function passes the encrypted keyword indexs keywordIndex and returns transactions id set and key set associated with the keywordIndex.

withdraw():This function can only be executed by the contract's creator(DO). DO withdraws the search service fee paid by the user.

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.2018.2851611, IEEE Access

Algorithm 4: deleteFile

```
Input: keywordIndex, txid
Output: null
if msg.sender is not dataOwner then
    throw;
end
get Index[keywordIndex] array's length len
if len equal 0 then
    return;
else
    for i ⇐ 0 to len-1 do
        if Index[keywordIndex][i].txid equal txid then
            for j ⇐ i+1 to len-1 do
                Index[keywordIndex][j-1] ⇐
                Index[keywordIndex][j]
            end
            delete Index[keywordIndex][len-1]
            break;
        end
    end
end
```

Algorithm 5: deleteKeyword

```
Input: keywordIndex
Output: null
if msg.sender is not dataOwner then
    throw;
end
get Index[keywordIndex] array's length len
if len equal 0 then
    return;
else
    delete Index[keywordIndex]
end
```

Algorithm 6: search

```
Input: keywordIndex
Output: searchResult
if tx.origin is not dataOwner and $msg.value < $cost
then
    throw;
end
get Index[keywordIndex] array's length len;
if tx.origin is not dataOwner then
    if len equal 0 then
        send $msg.value to msg.sender;
        searchResult ⇐ null;
    else
        send $cost to dataSharing contract address;
        send $msg.value − $cost to msg.sender;
        searchResult ⇐ Index[keywordIndex];
    end
else
    searchResult ⇐ Index[keywordIndex];
end
return searchResult;
```

Algorithm 7: withdraw

```
Input: null
Output: null
if msg.sender is not dataOwner then
    throw;
end
if contract's balance > 0 ether then
    send contract's balance to msg.sender;
end
```

dataUser contract

In the Ethereum smart contract, the return value of the non-constant function can only be obtained through logs event. Therefore, in the above dataSharing contract, the search results returned by the search function can only be obtained through events. However, if data users directly use events to obtain search results, there are security risks. Ethereum events are viewable by anyone, so someone can listen to events and effortlessly get some results. To address this problem, we designed another smart contract. The contract was deployed by a data user. the data user invoked the search function of the dataSharing contract, and saved the search results in the contract. Only the data user have the right to view search results, thus solving this problem. We call this contract the dataUser contract.

dataUser contract initialization: This process defines some variables of the contract when the contract is created.

1) In the dataUser contract, the search function of the dataSharing contract needs to been invoked. Therefore, a dataSharing contract object instance needs to be initialized.
2) The owner variable of address types, which defines the address of the DU.
3) The searchResult variable of struct types, which saves the search result.

The dataUser contract mainly provides the following three function interfaces:

deposit(value): This function is used to deposit ether into the dataUser contract. The smart contract balance are used to pay for the cost of invoking the dataSharing contract search function.

dataSearch(keywordIndex): This function can only be executed by the contract creator(DU). It passes the encrypted keyword indexs keywordIndex as the function argument.

getResult(): This function is identified by the keyword **view**, indicating that the function only performs readonly operations, does not change the state of the blockchain, and therefore will not be recorded on the blockchain. This function can only be executed by the contract creatr(DU).

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.2018.2851611, IEEE Access

TABLE 2. ABE program cost test (gasprice=2 $Gwei$, 1 $ether$=416 USD)

Function	Size(bytes)	Gas Used	Actual Cost(ether)	USD
System master key	208	28344	0.000056688	0.0236
User secret key	1088	94984	0.000189968	0.0791
ciphertext	710	69280	0.00013856	0.0577

Algorithm 8: deposit

Input: deposite value
Output: null
if *msg.value not equal deposite value* **then**
  throw;
end
send $value to dataUser contract address

Algorithm 9: dataSearch

Input: keywordIndex
Output: null
if *msg.sender is not owner* **then**
  throw;
end
call dataSharing contract's search();
save search result to struct searchResult;

Algorithm 10: getResult

Input: null
Output: searchResult
if *msg.sender is not owner* **then**
  throw;
end
return searchResult;

Through this function, the DU can be called locally to obtain the results of the search, and others cannot see this process.

VI. PERFORMANCE AND SECURITY ANALYSIS

A. PERFORMANCE TEST

We implemented a prototype to analyze feasibility and performance of the scheme. The specific configuration of experimental platform and experimental environment is: intel core i7-4790@3.60GHz processor, 4GB RAM, and the system is ubuntu 16.04LTS. The programming language is C++ and solidity.

In the ABE program, we use the cryptography library Miracl [46]. The selected curve is Cocks-Pinch curve, the security level is AES-80, the number of user attributes is 3, the hash function H_0 and H_1 is the function in Miracl library, and the pseudorandom function F is SHA256. When we conducted the experiment, the gasPrice was set to $2Gwei$, where $1Gwei = 10^9 wei = 10^{-9} ether$.

In our scheme, the time complexity of the search algorithm is $\mathcal{O}(1)$, which depends on the block generation time. Therefore, we mainly discuss the space cost of the algorithm.

In order to make data readable, we used BASE64 encoding and converting each part to the data field in the json format string, so the data was somewhat more bloated than the actual data. As shown in Table. 2, the system master key size are constant, and will not increase with the number of attributes. The size in the experiment is 208 bytes. The cost is \$0.236. The size of the user's secret key is the largest. With the increase of the number of attributes, the size of the user's secret key will increase. Fortunately, the process only needs to be performed once for each user. When the number of attributes is 3, we measured the user's secret key size is 1088 bytes and the cost is \$0.0791, the price is acceptable. Since we chose the ABE scheme with a constant ciphertext size, even if the number of attributes increases, the size of the ABE ciphertext will not increase, which is approximately 710 bytes and costs \$0.0577.

The system master key only need to be distributed once. The user secret key needs to be distributed once for each data user. For each shared file, the ABE ciphertext needs to be stored once. We can charge a fixed fee from each user who successfully searched for the corresponding result in the dataSaring contract to help the data owner pay for the relevant storage fees. For the smart contracts used to store encrypted keyword indexs and search, we deployed it on the Ethereum official test network Rinkeby.

Some of the smart contracts costs measured by the experiment are shown in Table. 3 and Table. 4 below:

TABLE 3. smart contract cost test (gasprice=2 $Gwei$,1 $ether$ = 416 USD)

Function	Gas Used	Actual Cost(ether)	USD
dataSharing Contract create	1143873	0.002287746	0.9541
addUser	44349	0.000088698	0.0370
removeUser	14308	0.000028616	0.0119
withdraw	30231	0.000060462	0.0254
dataUser Contract create	529014	0.001058028	0.4410

In Table. 3, the costs of some operations of the smart contract are listed, and multiple executions costs of these operations is almost unchanged. The dataSharing contract creation and dataUser contract creation operations are performed only once and the costs is \$0.9541 and \$0.4410, respectively. When a user joins the system, an addUser operation needs to be performed; when a user is removed from the authorized set, the removeUser operation needs to be performed. The two operations performed costed \$0.0370 and \$0.0119, respectively. When the search operation was successfully executed, the user need to pay a fixed \$cost to the dataSharing contract. In the experiment, we set \$cost as 0.01 $ether$. Once in a while, the dataSharing contract creator can check the balance of the dataSharing contract. If it is

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.2018.2851611, IEEE Access

greater than 0, the balance in the contract can be transferred to creator's EOA with the withdraw operation. In addition to the above operations, the costs of some operations will vary with the number of files, so we tested some of the costs under different number of files and listed in Table. 4.

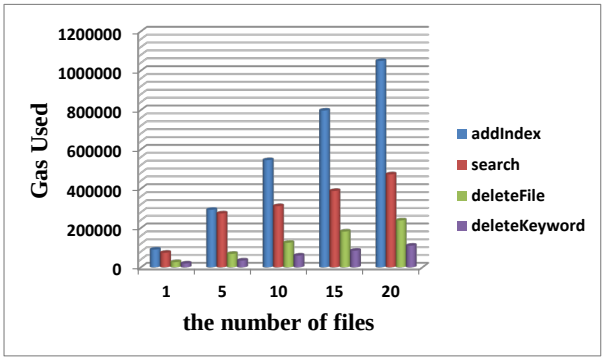

FIGURE 5. Smart contract operation costs under different number of files

We built the encrypted keyword indexs for five different keywords and add number of files 1, 5, 10, 15, and 20, respectively. The addIndex, search, deleteFile, and deleteKeyword operations were tested and the results in Table. 4 and Fig. 5 were obtained. As can be seen from Table. 4 and Fig. 5, with the increase of the number of files, the costs of these four operations increase accordingly. Among them, the addIndex operation increases almost linearly with the increase of the number of files, and the costs for adding five files for one keyword is about $0.2424. The search operation increases as the number of files corresponding to the search keyword increases, and the costs of returning the five search results is approximately $0.2303. When we test the deleteFile operation, we delete the first file record under different keywords. At this time, the smart contract needs to perform the most operations. That is, delete the first file record under each keyword at this time. The costs is the largest. When the number of files is 5, the costs of deleting the first file record is approximately $0.0579. With the increase in the number of files, the costs of the deleteKeyword operation increases, and deleting a keyword with 5 file records costs about $0.0293.

B. SECURITY AND PRIVACY ANALYSIS

In our paper, combining the Ethereum blockchain, the decentralized storage system IPFS, attribute-based encryption (ABE) mechanism, and smart contract technology, we gain more benefits than data storage and sharing in traditional cloud storage systems. In this section, we will discuss the benefits, security, and privacy of this scheme.

Data owners control their own data: In our solution, there is no need for a trusted PKG, and data owners have the ability to distribute keys for users as needed to achieve fine-grained access control. In the traditional ABE scheme, we need to assume that the cloud storage server will perform operations such as storage, transmission, and search honestly to ensure the availability of data. In a decentralized storage system, there is no need to trust storage nodes, and the reliability and availability of data is guaranteed by smart contracts,etc.

Avoid single point of failure: In our solution, compared with the traditional cloud storage, the decentralized storage system IPFS that we adopted can solve the single point of failure problem. Its redundancy backup technology Erasure coding, Proof of Replication and Filecoin incentives ensure data reliability and availability. At the same time, IPFS is running in the peer-to-peer manner, using DHT routing technology and BitTorrent technology, with higher data throughput and lower prices.

Search fairness: In traditional scheme, we need to rely on cloud service providers to honestly perform search operations and return corresponding results. However, cloud service providers may return erroneous results or do not return results to save resources, etc., resulting in users paying for services but unable to enjoy services. In this paper, we proposed scheme to ensure the fairness of the search process through smart contracts. Smart contracts can perform search operations honestly and according to predefined logic, and return corresponding results.

Security and privacy: In the IPFS, large files are chunked and stored on different storage nodes. The files we want to share are encrypted using AES algorithm and stored in IPFS storage nodes. The storage nodes can only see a part of ciphertexts and cannot obtain any information of the files. The encryption key of the file is first encrypted by the ABE algorithm, then encrypted with the AES algorithm together with other information (file location hash ciphertext) and stored on the blockchain. After encryption by the ABE algorithm and AES algorithm, although anyone can see the ciphertext information, a user whose attribute does not meet the access policy cannot decrypt the file encryption key, and cannot download the encrypted file from IPFS. So we achived fine-grained access control over data. It can be said that as long as the Ethereum blockchain network and the ABE scheme are safe, the proposed scheme is safe.

VII. CONCLUSION AND FUTURE WORK

At present, traditional cloud storage may cause users data to be unavailable due to force majeure factors (such as natural disasters, government censors, etc.). The ABE technology and searchable encryption technology on ciphertext are important technologies for solving data privacy and fine-grained access control problems. However, the traditional ABE solution always requires the existence of a trusted private key generator(PKG). The private key generated by the PKG for the users is not flexible enough, and may result in key abuse, disclosure of users data, etc. Traditional searchable encryption schemes require the cloud server to perform search operations honestly, but in actual applications, the cloud server may return incorrect results or even no results to save resources.

The decentralized storage approach can solve single point of failure in traditional cloud storage systems. At the same

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.2018.2851611, IEEE Access

TABLE 4. smart contract cost test under different number of files(gasprice=2 $Gwei$,1 $ether$ = 416 USD)

The number of files	Function	Gas Used	Actual Cost(ether)	USD
1	addIndex	91010	0.00018202	0.0753
	search	74318	0.000148636	0.0623
	deleteFile	26754	0.000053508	0.0225
	deleteKeyword	19781	0.000039562	0.0167
5	addIndex	292879	0.000585758	0.2424
	search	274367	0.000548734	0.2303
	deleteFile	68816	0.000137632	0.0579
	deleteKeyword	34898	0.000069796	0.0293
10	addIndex	545233	0.001090466	0.4513
	search	311548	0.000623096	0.2615
	deleteFile	125451	0.000250902	0.1054
	deleteKeyword	60093	0.000120186	0.0505
15	addIndex	797461	0.001594922	0.6599
	search	388764	0.000777528	0.3269
	deleteFile	182086	0.000364172	0.1531
	deleteKeyword	85288	0.000120186	0.0717
20	addIndex	1049690	0.00209938	0.8686
	search	473160	0.00094632	0.3972
	deleteFile	238721	0.000477442	0.2008
	deleteKeyword	110483	0.000220966	0.0925

time, compared to centralized storage, it also has a series of advantages such as low price and high throughput. In this paper, we study the data storage and sharing problems in decentralized storage systems and propose a framework that combines the decentralized storage system IPFS, Ethereum blockchain, and attribute-based encryption (ABE) technologies. In this framework, no trusted PKG is needed. The data owner has the ability to distribute secret key for users, and encrypts his data under specified access policy to achieve fine-grained access control over data. At the same time, based on the smart contract on the Ethereum blockchain, the keyword search function in the ciphertext of the decentralized storage system is implemented, and the problem that the cloud server does not return results or return wrong results in the traditional cloud storage is solved. In addition, we provide a test case study. Through experimental simulations, we analyze experimental data and demonstrate the rationality and feasibility of the scheme.

However, our scheme does not implement the functions of user's attribute revocation and access policy update. This is our next research direction.

APPENDICES

The dataSharing contract and the dataUser contract was deployed on the Rinkeby Testnet of Ethereum with the following address:

DO account address: 0x92a840C7f4AAb93418eE035F94B719CfA93ea049

DU account address: 0x11969FdCC5eEF405a51F57EC6d9Ce9eDe7716bcF

dataSharing contract address: 0x3ca4b656E24D45Fc1A5f408BF5Ac1b9F3EF05F1F

dataUser contract address: 0xafdBCe7575143ddff0e34874bc95812EA620EccD

Using these address, the transactions can be seen at: https://rinkeby.etherscan.io/.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grants No. 61572019, No. 61173192, the Key Project of Natural Science Foundation of Shaanxi Province of China under Grant No. 2016JZ001. Thanks also go to the anonymous reviewer for their useful comments.

REFERENCES

[1] 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/

[2] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2005, pp. 457–473.

[3] J. Zhang, X. A. Wang, and J. Ma, "Data owner based attribute based encryption," in Intelligent Networking and Collaborative Systems (INCOS), 2015 International Conference on. IEEE, 2015, pp. 144–148.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitco.in/pdf/bitcoin.pdf

[5] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum project yellow paper, vol. 151, pp. 1–32, 2014.

[6] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," Tech. rep. 2016-1.10. Zerocoin Electric Coin Company, Tech. Rep., 2016.

[7] "Blockchain for financial services." [Online]. Available: https://www.ibm.com/blockchain/financial-services/

[8] "Blockchain for supply chain." [Online]. Available: https://www.ibm.com/blockchain/supply-chain/

[9] S. Y. Conner Fromknecht, Dragos Velicanu, "A Decentralized Public Key Infrastructure with Identity Retention," 2014. [Online]. Available: https://eprint.iacr.org/2014/803.pdf

[10] "Proof of Existence." [Online]. Available: https://proofofexistence.com/

[11] "A decentralized network for internet of things." [Online]. Available: https://iotex.io/

[12] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj a peer-to-peer cloud storage network," 2014.

[13] J. Benet, "Ipfs-content addressed, versioned, p2p file system," arXiv preprint arXiv:1407.3561, 2014. [Online]. Available: https://ipfs.io/

[14] P. Labs, "Filecoin: A Decentralized Storage Network," 2018. [Online]. Available: https://filecoin.io/filecoin.pdf

[15] G. Zyskind, O. Nathan et al., "Decentralizing privacy: Using blockchain to protect personal data," in Security and Privacy Workshops (SPW), 2015 IEEE. IEEE, 2015, pp. 180–184.

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.2018.2851611, IEEE Access

[16] Y. Rahulamathavan, R. C.-W. Phan, S. Misra, and M. Rajarajan, "Privacy-preserving Blockchain based IoT Ecosystem using Attribute-based Encryption," 2017.

[17] H. Es-Samaali, A. Outchakoucht, and J. P. Leroy, "A blockchain-based access control for big data," International Journal of Computer Networks and Communications Security, vol. 5, no. 7, p. 137, 2017.

[18] D. Vorick and L. Champine, "Sia: simple decentralized storage," 2014. [Online]. Available: https://prod.coss.io/documents/white-papers/siacoin.pdf

[19] L. Zu, Z. Liu, and J. Li, "New ciphertext-policy attribute-based encryption with efficient revocation," in Computer and Information Technology (C-IT), 2014 IEEE International Conference on. IEEE, 2014, pp. 281–287.

[20] P. Zhang, Z. Chen, K. Liang, S. Wang, and T. Wang, "A cloud-based access control scheme with user revocation and attribute update," in Australasian Conference on Information Security and Privacy. Springer, 2016, pp. 525–540.

[21] J. Li, Y. Shi, and Y. Zhang, "Searchable ciphertext-policy attribute-based encryption with revocation in cloud storage," International Journal of Communication Systems, vol. 30, no. 1, 2017.

[22] J. Li, W. Yao, J. Han, Y. Zhang, and J. Shen, "User collusion avoidance cp-abe with efficient attribute revocation for cloud storage," IEEE Systems Journal, vol. 12, no. 2, pp. 1767–1777, 2018.

[23] A. Kapadia, P. P. Tsang, and S. W. Smith, "Attribute-based publishing with hidden credentials and hidden policies." in NDSS, vol. 7. Citeseer, 2007, pp. 179–192.

[24] Y. Zhang, X. Chen, J. Li, D. S. Wong, H. Li, and I. You, "Ensuring attribute privacy protection and fast decryption for outsourced data security in mobile cloud computing," Information Sciences, vol. 379, pp. 42–61, 2017.

[25] M. Chase, "Multi-authority attribute based encryption," in Theory of Cryptography Conference. Springer, 2007, pp. 515–534.

[26] H. Qian, J. Li, Y. Zhang, and J. Han, "Privacy-preserving personal health record using multi-authority attribute-based encryption with revocation," International Journal of Information Security, vol. 14, no. 6, pp. 487–497, 2015.

[27] H. S. Gardiyawasam Pussewalage and V. A. Oleshchuk, "A distributed multi-authority attribute based encryption scheme for secure sharing of personal health records," in Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies. ACM, 2017, pp. 255–262.

[28] J. Li, Y. Wang, Y. Zhang, and J. Han, "Full verifiability for outsourced decryption in attribute based encryption," IEEE Transactions on Services Computing, 2017.

[29] J. Li, X. Lin, Y. Zhang, and J. Han, "Ksf-oabe: outsourced attribute-based encryption with keyword search function for cloud storage," IEEE Transactions on Services Computing, vol. 10, no. 5, pp. 715–725, 2017.

[30] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on. IEEE, 2000, pp. 44–55.

[31] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in International conference on the theory and applications of cryptographic techniques. Springer, 2004, pp. 506–522.

[32] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in Theory of Cryptography Conference. Springer, 2007, pp. 535–554.

[33] Z. Wan and R. H. Deng, "Vpsearch: achieving verifiability for privacy-preserving multi-keyword search over encrypted cloud data," IEEE Transactions on Dependable and Secure Computing, 2016.

[34] H. Li, F. Zhang, J. He, and H. Tian, "A searchable symmetric encryption scheme using blockchain," arXiv preprint arXiv:1711.01030, 2017.

[35] C. Cai, X. Yuan, and C. Wang, "Towards trustworthy and private keyword search in encrypted decentralized storage," in Communications (ICC), 2017 IEEE International Conference on. IEEE, 2017, pp. 1–7.

[36] H. G. Do and W. K. Ng, "Blockchain-based system for secure data storage with private keyword search," in Services (SERVICES), 2017 IEEE World Congress on. IEEE, 2017, pp. 90–93.

[37] P. Jiang, F. Guo, K. Liang, J. Lai, and Q. Wen, "Searchain: Blockchain-based private keyword search in decentralized storage," Future Generation Computer Systems, 2017.

[38] Y. Zhang, D. Zheng, X. Chen, J. Li, and H. Li, "Computationally efficient ciphertext-policy attribute-based encryption with constant-size ciphertexts," in International Conference on Provable Security. Springer, 2014, pp. 259–273.

[39] M. Crosby, P. Pattanayak, S. Verma, and V. Kalyanaraman, "Blockchain technology: Beyond bitcoin," Applied Innovation, vol. 2, pp. 6–10, 2016.

[40] "Ethereum homestead documentation." [Online]. Available: https://readthedocs.org/projects/ethereum-homestead/

[41] "Ethereum blockchain app platform." [Online]. Available: https://www.ethereum.org/

[42] "Diffie-hellman key exchange." [Online]. Available: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

[43] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," 2018. [Online]. Available: http://nisplab.whu.edu.cn/paper/infocom_2018_1.pdf

[44] "Verify contract code." [Online]. Available: https://etherscan.io/verifyContract

[45] "Units and globally available variables." [Online]. Available: http://solidity.readthedocs.io/en/latest/units-and-global-variables.html

[46] "Miracl." [Online]. Available: https://www.miracl.com/

SHANGPING WANG received his B.S. degree in mathematics in 1982 from Xi'an University of Technology, Xi'an, China. He received his M.S. degree in applied mathematics in 1989 from Xi'an Jiaotong University, Xi'an, China, and earned his Ph.D. degree in cryptology in 2003 from Xidian University, Xi'an, China. Currently, he is a professor in Xi'an University of Technology. His current research interests are cryptography and information security.

YINGLONG ZHANG received his B.S. degree in 2015 from School of Science, Xi'an University of Technology, Xi'an, China. He is currently working toward the M.S. degree from Xi'an University of Technology, Xi'an, China. His research interests include information security and blockchain technology.

YALING ZHANG received her B.S. degree in computer science in 1988 from Northwest University, Xi'an, China. She received her M.S. degree in computer science in 2001, and earned her Ph.D. degree in mechanism electron engineering in 2008, both from the Xi'an University of Technology, Xi'an, China. Currently, she is a professor in Xi'an University of Technology. Her current research interests include cryptography and network security.