Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 6658920, 20 pages
https://doi.org/10.1155/2021/6658920

Research Article

BSSPD: A Blockchain-Based Security Sharing Scheme for Personal Data with Fine-Grained Access Control

Hongmin Gao,[1] **Zhaofeng Ma**,[1] **Shoushan Luo**,[1] **Yanping Xu**,[2] **and Zheng Wu**[3]

[1]*Information Security Center, Beijing University of Posts and Telecommunications, Beijing 100876, China*
[2]*School of Cyberspace Security, Hangzhou Dianzi University, Hangzhou, Zhejiang Province 310018, China*
[3]*School of Electronics and Information Engineering, Hunan University of Science and Engineering, China*

Correspondence should be addressed to Zheng Wu; 18153361706@189.cn

Received 25 November 2020; Revised 31 December 2020; Accepted 29 January 2021; Published 20 February 2021

Academic Editor: Zhuojun Duan

Privacy protection and open sharing are the core of data governance in the AI-driven era. A common data-sharing management platform is indispensable in the existing data-sharing solutions, and users upload their data to the cloud server for storage and dissemination. However, from the moment users upload the data to the server, they will lose absolute ownership of their data, and security and privacy will become a critical issue. Although data encryption and access control are considered up-and-coming technologies in protecting personal data security on the cloud server, they alleviate this problem to a certain extent. However, it still depends too much on a third-party organization's credibility, the Cloud Service Provider (CSP). In this paper, we combined blockchain, ciphertext-policy attribute-based encryption (CP-ABE), and InterPlanetary File System (IPFS) to address this problem to propose a blockchain-based security sharing scheme for personal data named BSSPD. In this user-centric scheme, the data owner encrypts the sharing data and stores it on IPFS, which maximizes the scheme's decentralization. The address and the decryption key of the shared data will be encrypted with CP-ABE according to the specific access policy, and the data owner uses blockchain to publish his data-related information and distribute keys for data users. Only the data user whose attributes meet the access policy can download and decrypt the data. The data owner has fine-grained access control over his data, and BSSPD supports an attribute-level revocation of a specific data user without affecting others. To further protect the data user's privacy, the ciphertext keyword search is used when retrieving data. We analyzed the security of the BBSPD and simulated our scheme on the EOS blockchain, which proved that our scheme is feasible. Meanwhile, we provided a thorough analysis of the storage and computing overhead, which proved that BSSPD has a good performance.

1. Introduction

The development of 5G and Internet of Things technology provides a large amount of training data for the rapid implementation of artificial intelligence (AI). At the same time, data security and privacy protection have become the most interesting topics in data governance and sharing. Powerful data mining and analysis have brought potential threats to personal privacy protection. Traditionally, most people choose to outsource their data to cloud servers for sharing and dissemination. However, most of the data stored in the cloud is very sensitive, especially those data generated by IoT devices that are closely related to human life. These data have their particularities and may contain personal-related information such as life, work, and healthcare; once personal data is stolen or leaked illegally and linked to the data owner's real identity, it may bring great trouble to an individual. Therefore, integrating data and generating value while ensuring data security and privacy have become a significant challenge for all contemporary companies that use big data and AI.

At present, researchers have proposed many secure sharing schemes in the cloud environment [1–9]. These schemes seem to solve the security and privacy issues during data sharing. Nevertheless, these schemes all have a standard feature: they are overly dependent on the Cloud Service Provider (CSP). They believe that the CSP is a trusted third-party organization, and their security models assume that

the CSP is semitrustable, which means that the CSP will be curious about the data but will not destroy it. It means that the following situations are always inevitable:

(1) The CSP itself may make profits from the user's private data, or its insiders may do evil and cause the user's privacy disclosure. Although some methods, such as attribute-based encryption algorithms, can achieve user-defined access policies that seem user-centric, these methods still require a trusted third party to generate and manage user keys. It is impossible to exclude the possibility of collusion between these trusted centers. All these will lead to the fact that once the data owners upload their data to the cloud server, they will no longer have their data's absolute possession

(2) The data is centrally stored on cloud servers and managed by the CSP. An inevitable single point of failure may lead that users cannot obtain their data generally by using the cloud service. The CSP can improve data security and service stability by utilizing disaster recovery backup. However, some irresistible factors will prevent users from using cloud services to obtain their data, such as political factors

(3) To provide better service, the CSP needs to spend more money to buy servers, hire better employees, rent the data center venues, and so on. These costs are increasing gradually, and the CSP cost is also increasing and the construction of the management platform. Users ultimately pay the operating costs of the CSP

From the above point of view, to better protect data security and personal privacy, it is very urgent to design a whole user-centric data-sharing scheme to solve the above problems. In this scheme, we do not need to rely on any trusted third party to store and disseminate data, nor do we worry that the data will be inaccessible. Fortunately, with the emergence and development of Bitcoin [10], as a decentralized and self-organized cryptocurrency, its underlying technology blockchain can elegantly help us realize such a data security sharing scheme [11–14]. In this paper, we proposed a data-sharing scheme based on blockchain. The main contributions of this paper are as follows:

(1) A user-centric data security sharing scheme named BSSPD is proposed, which combines blockchain, CP-ABE, and IPFS. The data owner encrypts his sharing data and stores it on IPFS to maximize decentralization, and BSSPD allows the data owners to have fine-grained access control over their data. Moreover, it supports revoking permissions of a specific data user at an attribute level without affecting others

(2) In BSSPD, the data owner publishes data-related information and distributes decryption keys for data users through the blockchain. To avoid denial of service attacks, data users need to complete a proof of work (PoW) before registering, which is similar to the mining process of Bitcoin, and the data owner can adjust the target of PoW according to the number of data users in the system

(3) BSSPD sets ciphertext keyword indices for each data-related data user. Combined with CP-ABE, it further prevents the privacy disclosure that data labels may cause to the data owner and protects the data user's privacy during retrieval

(4) We experimented with our scheme on the EOS blockchain and provided the detailed implementation of algorithms and Smart Contracts. Together with the security analysis, it proved that our scheme is feasible

(5) We used five MacBooks to build an EOS private chain in the laboratory environment and simulated our scheme. Analysis of storage and computing overhead proved that BSSPD has a good performance

The rest of this paper is organized as follows. Section 2 consists of related works. Section 3 reviews some preliminary knowledge used throughout this paper. In Section 4, we have an overview of our scheme. Specific implementation details are described in Section 5. Security and performance analysis are discussed in Section 6. Finally, the conclusion and future direction are presented.

2. Related Work

As early as 2015, Swan pointed out that there was not yet an acceptable "health data common" model [15] with appropriate privacy and reward systems for public sharing of personal health data and quantified self-tracking data. Simultaneously, the author believes that blockchain can precisely provide such a structure for creating a secure, remunerated, and owner-controlled health data sharing. Zyskind et al. described a distributed personal data management system [16] that ensures users own and control their data. The system encrypts the data collected from the user's mobile phone and stores it off-chain and only stores the data's hash value on the blockchain. Meanwhile, two acceptable transaction types named *Taccess* and *Tdata* are defined, in which *Taccess* is used to implement access control management, and *Tdata* is used for data storage and retrieval. Azaria et al. proposed MedRec system [17], a blockchain-based decentralized record management system for electronic medical records (EMRs). MedRec provides patients with a comprehensive and immutable log, and the patients can access their medical information at any time across providers and locations. However, the system implements permissionless blockchain with PoW consensus, lacking data security, data privacy, and throughput. Xia et al. proposed MeDShare [18], a system that solves the problem of sharing medical data in a trustless environment by custodians of medical big data. Dubovitskaya et al. have proposed a framework for managing and sharing EMR data for cancer patient care [19]. It uses a permission chain to maintain metadata and access control

policies and uses cloud services to store the encrypted data. Patients can define their access control policies to ensure data security and availability. The above-mentioned data-sharing schemes based on blockchain give an ideal blueprint, but most of them only describe the scheme's outline and do not provide the implementation details of the required protocol.

In the following years, many researchers have designed and implemented more robust access control protocols on blockchain to protect data privacy and security during sharing. Liang et al. used the consortium chain Hyperledger Fabric to realize a user-centric health data-sharing model [20] in which the cloud storage is used as a data warehouse and the blockchain ledger is constructed to store operations such as query and update. At the same time, it uses the member management service provided by Hyperledger Fabric to strengthen the users' identity authentication and the channel model to protect users' privacy. Fan et al. focused their attention on mobile network data sharing and privacy protection in the 5G era and proposed an efficient sharing scheme based on blockchain [21]. The main idea is to define a transaction format on blockchain to represent an access strategy. The strategy includes access requestor, content provider, visitor, and the beginning and ending time of access allowed, which is a role-based access control model. Zhang et al. proposed a blockchain-based data-sharing scheme for AI-powered network operations [22]. The scheme sets up two different types of chain, in which DataChain is used as access control tools for data, and BehaviorChain is used to store access records and ensure they cannot be tampered with. They divide access permissions into four levels. Zhou et al. proposed a blockchain-based file-sharing system [23] to address inefficient file sharing during the review of academic papers. The scheme uses Access Control Language (ALC) to exercise access control over the information stored on-chain. It needs to define an access policy on the blockchain for each pair of users and resource. Patel proposed a crossdomain image-sharing framework based on blockchain [24], which uses blockchain as data storage and allows patients to define an access policy. They pointed out that this approach can protect the data from unrelated parties, but no research has been conducted on privacy and security. Tan et al. have proposed a blockchain-based access control scheme for Cyber-Physical Social System (CPSS) big data [25], called BacCPSS. BacCPSS uses an address of blockchain as the user's identity and maintains a user access matrix on the Smart Contract, ensuring that only operations authorized in the access matrix can be performed. The access control methods implemented in the above data-sharing schemes either need to maintain large numbers of access rules on the chain or cannot achieve fine-grained access control. Neither the access control matrix nor the RBAC is suitable for distributed environments like blockchain.

ABE is considered the most appropriate technology to solve data security and privacy protection problems in a distributed environment. Therefore, recently, researchers have used ABE to achieve fine-grained access control over data on the blockchain. Jemel and Serhrouchni proposed a decentralized access control mechanism [26]. For the first time, researchers used blockchain nodes to execute a CP-ABE algorithm to verify user access rights' legitimacy. The scheme designs two types of transactions: *SetPolicy* and *GetAccess*. But it does not use Smart Contracts, and it is obvious that the scheme is unable to achieve more complex requirements. Sun et al. constructed a model of secure storage and effective sharing for electronic medical data based on ABE and blockchain [27], which provides better access control. Doctors use ABE to encrypt patients' medical data and store it on IPFS. However, it also does not use Smart Contracts. It only broadcasts some ABE parameters stored in transactions, which cannot achieve more complex business functions. Wang et al. proposed a sharing scheme [28] in which users distribute secret keys. It realizes that the data owner has a fine-grained access control on his data. At the same time, the Ethereum Smart Contract is used to realize the retrieval of ciphertext keywords. However, it requires multiple off-chain communication between users, and more importantly, it does not implement the permit revocation. Pournaghi et al. proposed a secure and efficient sharing scheme based on blockchain and ABE entitled MedSBA to record and store medical data [29]. It implements the update and revocation of permissions by broadcasting a new strategy to cover the previous transaction, but this will lead to users who do not want to be revoked to update their keys.

3. Preliminary

3.1. Bilinear Groups of Composite Order. Let $n = p_1 p_2$ (p_1 and p_2 are distinct primes), G_1 and G_2 be cyclic groups of order n, and g be a generator of G_1. We call $e : G_1 \times G_1 \to G_2$ as a bilinear pairing, if it is a map with the following properties:

(1) Bilinear: $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ for all $g_1, g_2 \in G_1$ and $a, b \in Z$

(2) Nondegenerate: there exists $g_1, g_2 \in G_1$, such that $e(g_1, g_2) \neq 1$

(3) Computable: There is an efficient algorithm to compute $e(g_1, g_2) \neq 1$ for all $g_1, g_2 \in G_1$

Let G_{p_1} and G_{p_2} denote the subgroups of G_1 with order p_1 and p_2. Then, $G_1 = G_{p_1} \times G_{p_2}$; g^{p_2} and g^{p_1} are the generators of G_{p_1} and G_{p_2}. Let g_1 and g_2 denote the generators of G_{p_1} and G_{p_2}. For all random elements $h_1 \in G_{p_1}$ and $h_2 \in G_{p_2}$, then we have $e(h_1, h_2) = 1$; because of that, $e(h_1, h_2) = e(g_1^a, g_2^b) = e(g^{ap_2}, g^{bp_1}) = e(g, g)^{abp_1p_2}$.

3.2. Linear Secret-Sharing Scheme (LSSS). Let $P = \{P_1, \cdots, P_n\}$ be a set of parties, and (A, ρ) denote an access structure in which A is an $l \times k$ access matrix with ρ mapping its rows. A linear secret-sharing scheme (LSSS) consists of two polynomial-time algorithms:

(1) $Share(A, \rho)$: to share a secret value s, it randomly chooses $v_1, v_2, \cdots, v_{k-1} \in Z$ and let $\mathbf{v} = (s, v_1, \cdots, v_{k-1})^T$. Let A_i denote the vector as the ith

row in matrix A, and then, the share $\sigma_i = A_i\mathbf{v}$ belongs to party $\rho(i)$

(2) Re $con(A, \rho)$: the algorithm takes $\omega \in A$ as input; let $L \in \{i \mid \rho(i) \in \omega\}$. Then, a set of recovery coefficients can be calculated effectively according to $\{\mu_i\}_{i\in L}$, so that $\sum_{i\in L}\mu_i\sigma_i = s$. Research has shown that the monotonic access structure is equivalent to the LSSS. Let (A, ρ) be an access structure and ω be a set of authorization; then $\exists\{\mu_i\}_{i\in L}$ makes that $\sum_{i\in L}\mu_i\sigma_i = s$. For unauthorized sets, such constants do not exist

3.3. Ciphertext-Policy Attribute-Based Encryption (CP-ABE). The CP-ABE mechanism was proposed by Bethencourt et al. [30]. It is a public key encryption scheme, but unlike RSA and ECC, CP-ABE is a one-to-many encryption scheme. In CP-ABE, the user's attributes correspond to the private key, and the access policy is embedded in the ciphertext [31]. Only when the decryption user's attributes satisfy the access policy can the data be decrypted. CP-ABE is mostly used for fine-grained access control. CP-ABE consists of four phases: initialization, key generation, encryption, and decryption, corresponding to the following four algorithms:

(1) $Setup(\lambda, S) \rightarrow (PSK, MSK)$

Initialization algorithm is a randomization algorithm, which is generally executed on a trusted key distribution center. The algorithm inputs a secure parameter λ and the attributes are set S, to generate the system public key PSK and the system master key MSK.

(2) $KeyGen(PSK, MSK, \omega) \rightarrow USK$

Key generation algorithm generates a private key USK for the data user according to the system public key PSK, the system master key MSK, and the data user's attributes ω.

(3) $Encrypt(PSK, M, A) \rightarrow CM$

Encryption algorithm is executed by the data owner. The algorithm inputs the system public key PSK, the message M to be encrypted, and the access control structure A associated with the access policy and outputs the ciphertext CM.

(4) $Decrypt(PSK, CM, USK) \rightarrow M$

Decryption algorithm is executed by the data user. The inputs of the algorithm are the system public key PSK, the user's private key USK, and the ciphertext CM. If the data user's attribute set ω satisfies the access policy, he will decrypt the ciphertext and obtain the corresponding plaintext M.

3.4. Blockchain. A blockchain concept originated from Nakamoto's Bitcoin paper [10], and it is based on cryptography and P2P network. The data on the blockchain is organized into blocks, which are chained in a particular chronological order. Cryptography and consensus mechanisms ensure the security and nonforgery of data. In short, as the underlying technology of cryptocurrencies like Bitcoin, blockchain is a distributed trusted ledger that cannot be tampered with.

3.4.1. Smart Contract. At the early stage of blockchain development, only cryptocurrencies like BTC and LTC were more successful applications. In 2013, Buterin introduced the concept of Smart Contract in his Ethereum white paper [32], demonstrating the first public blockchain with a built-in Turing complete language. Smart Contract [33] was defined as "a computerized transaction protocol that executes the terms of the contract." In the blockchain, Smart Contract is a code that relies on blockchain's trusted environment to automatically execute while enabling the blockchain to realize a more complex business. The smart contract operation mechanism based on blockchain is shown in Figure 1.

From a higher point of view, blockchain can be considered a state machine triggered by transactions, and its public ledger is a world state starting from the Genesis Block. Users can build a transaction and broadcast it from any node in the blockchain network. All block producers will perform the corresponding operation after receiving the transaction. Because of the consensus mechanism, all nodes will eventually get a consistent result and update the world state. The action triggered by a transaction can be to deploy a new Smart Contract or to invoke a Smart Contract from blockchain and execute it in a sandbox environment. Blockchain provides Smart Contract with the following capabilities:

Public state: everyone can see the Smart Contract's execution and its current global status on the public ledger, which cannot be tampered with.

Trusted propagation channel: after encrypting the message by the receiver's public key, the sender can broadcast the message through the blockchain. The receiver will receive the message, and it will be recorded on the blockchain securely and undeniably.

3.4.2. Transaction of EOS. In the EOS blockchain, there are three essential components named address, account, and transaction. Each user has his account in EOS, and each account corresponds to multiple ECDSA key pairs denoted by (pk, sk). The public key calculates an address of EOS through a hash function and base58 coding. The private key and the public key are used to sign and verify the transaction, respectively. If a user wants to invoke a Smart Contract on-chain, he needs to prepare such a transaction Tx [34]:

$$Tx = (Ref_{block}, t, Sig_u(Chain\_ID, Tx), Action(Code, Name, Auth_u, Data)). \quad (1)$$

Ref_{block} denotes the reference to the block number and header of a block which generated recently to prevent transactions from appearing on a forked chain. $Sig_u(Chain\_ID, Tx)$ denotes the user's signature information on the transaction which is used to verify the identity of the user who initiated the transaction by his public key. $Action$ represents the operation to be performed, where $Code$ is the name of the Smart Contract to be invoked, $Name$ is a method in Smart Contract to be called, $Auth_u$ is used to verify whether the user who initiated the transaction has the permission, and $Data$ is the parameters to be passed into the contract.

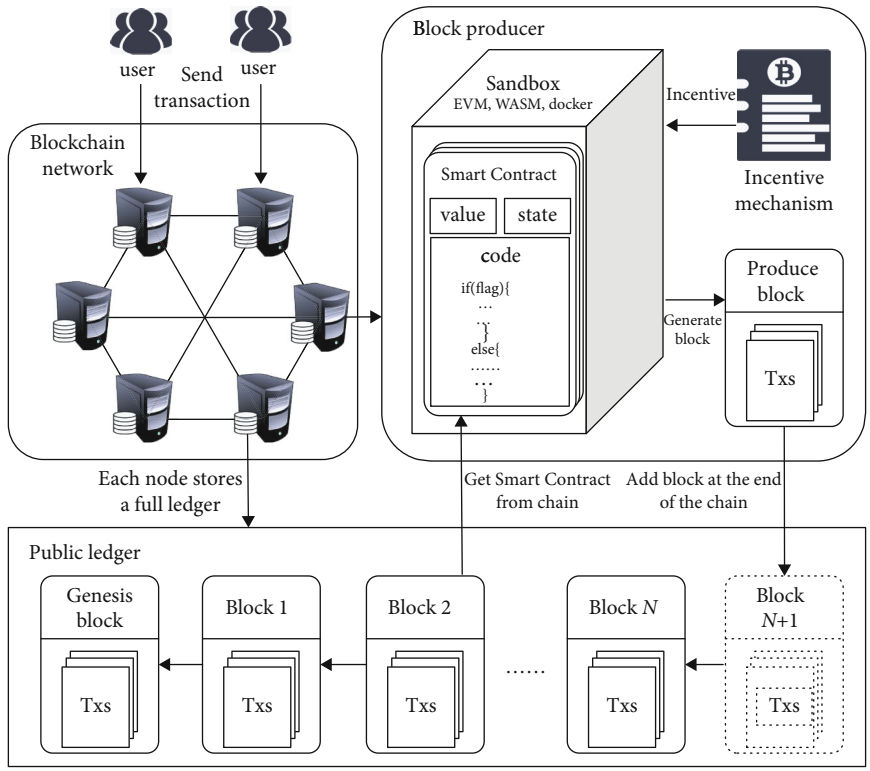

Figure 1: The operation mechanism of smart contract on blockchain.

3.4.3. Data Persistence of EOS. After the Smart Contract is executed, the occupied memory will be released, and all variable data in the program will be lost, so it is necessary to persist the data in Smart Contract. In the Smart Contact of Ethereum, data can only be stored in key-value pairs, which is difficult to meet more complex requirements. In EOS, it imitates Multiindex Containers in Boost library and develops a C++ class: *eosio*::*multi_index* (hereinafter referred to as multi_index). Each multi_index can be regarded as a table in the traditional database. Each row of the table can store an object, and the object's attributes can be any C++ data type. Therefore, the table constructed by multi_index in EOS is no less flexible than traditional databases. A significant feature of multi_index is that a primary key can be set as the main index and 16 secondary indices. Users can obtain any of these indices and use the *emplace*, *erase*, *modify*, and *find* functions of the index to insert, delete, update, and select data.

3.5. IPFS (InterPlanetary File System). The InterPlanetary File System is a globally oriented, point-to-point distributed version of the File System, dedicated to creating persistent and distributed storage and shared file network transmission protocols. By integrating existing technologies such as BitTorrent, DHT, Git, and SFS (self-certifying File System), IPFS provides a high-throughput content block storage model that contains content addressing hyperlinks. Simultaneously, it does not have a single point of failure, and the nodes in the system do not need to trust each other. Any resource, such as text, images, sound, video, and website code, once added to the IPFS network, computes the content to a uniquely encrypted hash value unique to the address. This address can be understood as a URL (Uniform Resource Locator) on the Web. If the user wants to use the file, they just need to go to this address to get them.

4. Overview of Our Scheme

This section will give an overview of the system model and the design of our proposed scheme. Table 1 shows some symbols and abbreviations involved in this paper.

4.1. System Model of BSSPD. Our proposed scheme BSPPD consists of four components: IPFS, blockchain, data owner, and data user. The *DO* encrypts his data and uploads it to IPFS, then invokes the Smart Contract on blockchain to save the returned address along with the decryption key. CP-ABE is used to realize a fine-grained access control of data. The *DO* distributes the private keys for *DU*s through blockchain, and only those who satisfy the access policy can download and decrypt the shared data. The whole process is entirely decentralized. The data is encrypted and stored in the IPFS to ensure the security of data and accessibility. The traces of the *DO* and *DU*s are stored on the blockchain, which cannot be tampered with or denied. The specific functions and responsibilities of these four parts are as follows:

(1) IPFS: provide a secure and reliable storage service. The incentive mechanism ensures that the data on IPFS will never be unavailable

(2) Blockchain: stores the public information and operational records in the whole scheme. Meanwhile, it can

Table 1: The symbols and abbreviations involved in this paper.

No.	Symbol	Description
1	DO	The data owner
2	DU	The data user
3	MSK	System master key
4	PK	System public parameters
5	S	All general attributes set
6	ω	The attributes set of a specific DU
7	$\mathscr{P}$	Access policy
8	uid	A user ID which is unique
9	$SK_{uid,\omega}$	The attribute private key of DU whose ID is uid
10	SK_{search}	The secret key of DU for search
11	$E = (E.Enc, E.Dec)$	An asymmetric encryption algorithm like ECC
12	(Sk_{com}, Pk_{com})	A pair of keys for algorithm E
13	$\varepsilon = (\varepsilon.Enc, \varepsilon.Dec)$	A symmetric encryption algorithm like AES
14	F	Data that the DO intends to share
15	CF	Ciphertext of the data F
16	$href_{location}$	The address where the data is stored on IPFS
17	kw	Keyword
18	t_{kw}	Search token of kw

be used as a reliable broadcast channel for transferring messages from the DO to DU. Without any trusted third party, it is the cornerstone of trust for the scheme. There are two Smart Contracts in BSSPD. *UMContract* is used to manage data users and *DSContract* is used to share data

(3) Data owner: responsible for creating and deploying the Smart Contract in the scheme. The DO can publish his sharing data and set an access policy for it. Meanwhile, the DO can grant and revoke a DU's access rights

(4) Data user: the DU is the person who wants to access the shared data. When DU's attributes meet the policy embedded in the ciphertext, he will decrypt the address and key to obtain the shared data

The system model of the proposed scheme is shown in Figure 2.

The CP-ABE algorithm we adopted was mainly inspired by [35] and extended to use the user's ID as an attribute to support permission revocation. The keyword ciphertext search in BSSPD was learned from [36]. The corresponding description of each step number in Figure 2 is shown as follows:

(i) The DO creates and deploys Smart Contracts. There are two Smart Contracts in our scheme. *UMContract* includes the functions of user registration, attribute management, identity management, and authentication. *DSContract* includes publishing sharing data, updating access policy, permission revocation, and data retrieval

(ii) The DO generates the system master key and system public key locally and stores the system public key in *DSContract*

(iii) The DU invokes *UMContract* to apply for registration, and he needs to provide his account of EOS and a public key. The public key is used to communicate with the DO, and the DO uses it to encrypt the message and broadcasts the ciphertext to the blockchain. Only the corresponding DU can decrypt the ciphertext and obtain the message

(iv) The DO assigns a unique uid to each DU who applies for, and generates a private attribute key and a secret search key for the DU. After encrypting these two keys with the DU's communication public key, the DO will save them in the Smart Contract together with the uid

(v) The DU obtains the ciphertext information of the keys and decrypts them with his private communication key

(vi) The DO randomly selects a key of the symmetric encryption algorithm, uses it to encrypt the sharing data, then uploads the ciphertext to the IPFS network, and IPFS returns an address

(vii) The DO sets an access policy for sharing data and sets a revocation list for each attribute in the policy, then encrypts the address along with the decryption key of shared data. The DUs in the revocation list do not have corresponding attributes when accessing the data

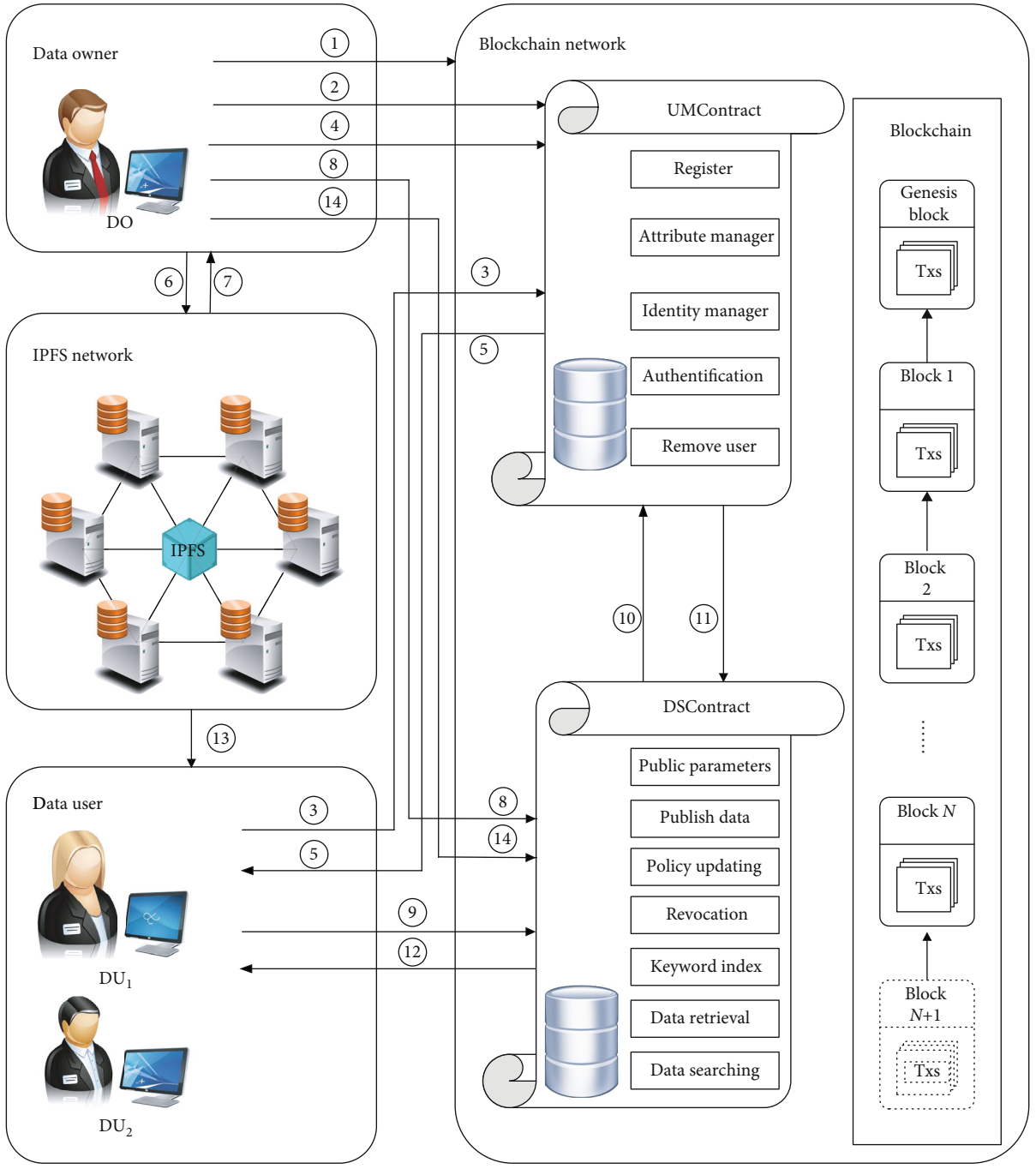

Figure 2: The system model of BSSPD.

(viii) The *DO* selects keywords to generate ciphertext indices for data-related *DU*s and then invokes the *DSContract* to store the indices and data-related information

(ix) The *DU* selects a keyword of the data to be retrieved and uses the trapdoor function to generate a search token

(x) The *DU* invokes *DSContract* to start searching for the desired data. *DSContract* will call *UMContract* to authenticate the *DU* and check whether the *DU* is legal

(xi) *UMContract* returns the authentication result to *DSContract*. If the *DU* is legal, the search function will continue to be executed

(xii) The *DU* obtains the search results from *DSContract*

(xiii) The *DU* uses his attribute private key to decrypt the acquired data-related information. If the *DU*'s unrevoked attributes still satisfy the access policy, he will get the address where the ciphertext data is stored on IPFS and the corresponding decryption key. The *DU* can download the ciphertext of the shared data from IPFS and decrypt it

(xiv) If the *DO* wants to revoke a *DU*'s attribute *A* to a certain shared data, he can add this *DU*'s *uid* to the revocation list of attribute *A*. Then, the *DO* will generate a new ciphertext and invoke *DSContract* to update the data-related information

4.2. Detail Design of BSSPD. The scheme we proposed is mainly composed of the following phases: initialization phase, apply and register phase, encryption and uploading phase, search phase, decryption and downloading phase, and permission revocation phase. This section will describe the detailed design of each phase and the corresponding relationship with the process steps in the previous section.

4.2.1. Initialization Phase. The primary function of the initialization phase is that the *DO* deploys Smart Contracts, then generates the system master key and the public parameters in the scheme, and stores them in the Smart Contract. The core algorithm of this phase is $Setup(1^{\lambda}) \rightarrow (MSK, PK)$, which was executed by the *DO*. The algorithm's input is a security parameter 1^{λ}, and the outputs are the system master key *MSK* and public system parameters *PK*. *MSK* will be kept secret by the *DO*, and *PK* will be stored in *UMContract* by the *DO* initiating a transaction. The corresponding steps in the system flowchart are (i) and (ii).

4.2.2. Apply and Register Phase. The apply and register phase's primary function is that the *DU* invokes to apply for registration, and an asymmetric encryption algorithm public key is required when applying. After that, the *DO* assigns a unique *uid* and distributes private keys for the *DU*. The core algorithm is $KeyGen(MSK, PK, uid, \omega) \rightarrow (SK_{data}, SK_{search})$ which is run by the *DO*. The inputs of the algorithm are the system master key *MSK*, the public parameters *PK*, the *uid* of the *DU*, and the general attribute set ω of the *DU*. It outputs the private attribute key SK_{data} and the search key SK_{search} of the *DU*. The *DO* executes $SK_{uid,\omega} = Enc_{Pk_{com}}(SK_{data} \mid SK_{search})$ and invokes *UMContract* to store $SK_{uid,\omega}$ in the Smart Contract. In this way, the *DU* can obtain his private keys securely and reliably. The corresponding steps in the system flowchart are (iii)–(v).

4.2.3. Encryption and Uploading Phase. The encryption and uploading phase's main function is that the *DO* encrypts sharing data and uploads it to IPFS. After that, the address and decryption key are encrypted and uploaded to *DSContract*, and the ciphertext keyword indices are established for the relevant *DU*s. The core algorithm is $Encrypt(F, (A_{l\times k}, \rho), \{R_{\rho(x)}\}_{x\in 1,\cdots,l}, PK)$ and executed by the *DO*. It consists of the following three substeps:

Step 1. $EncryptFile(F) \rightarrow (K_F, href_{location})$.

The input of the data encryption algorithm is the sharing data *F*, and outputs are the key K_F of a symmetric encryption algorithm and an IPFS address $href_{location}$. The whole process is to randomly select a private key K_F and encrypt *F* to get the ciphertext *CF* and then upload *CF* to IPFS to get the address $href_{location}$. The corresponding step in the system flowchart is (vi).

Step 2. $EncryptKey(K_F, href_{location}, \mathscr{P}, \{R_i\}_{i\in\mathscr{P}}, PK)_{(K_F|href_{location})}$.

The algorithm is used to encrypt the address, and the key whose inputs are the decryption key K_F, the IPFS address $href_{location}$, the access policy $\mathscr{P}$, a revocation list R_i for each attribute in $\mathscr{P}$, and system public parameters *PK*. Its output is the ciphertext of K_F and encrypted with CP-ABE. The corresponding step in the system flowchart is (vii).

Step 3. $IndexGen(kw, K_{search}) \rightarrow t_{kw}$.

In the algorithm that generates the ciphertext keyword index, the *DO* selects a keyword *kw* of data *F*, which is used as inputs together with the search secret key K_{search} of a relevant *DU*. The output is a search token t_{kw} and the corresponding step in the system flowchart is (viii).

4.2.4. Search Phase. The main function of the search phase is that a *DU* uses the trapdoor function to generate the corresponding search token according to the keyword of the shared data which he wants. After that, the *DU* invokes the contract *DSContract* for retrieval. This phase can be divided into two steps, as follows:

Step 1. $Trpdr(kw', SK_{search}) \rightarrow t'_{kw}$.

Generate search token algorithm, which is executed by the *DU*. The *DU* selects the keyword related to the shared data he wants to search, together with his SK_{search} as inputs, and the output is the search token t'_{kw} corresponding to the keyword. This corresponds to step (ix) in the system flowchart.

Step 2. $Search(t'_{kw}) \rightarrow CT_{(K_F|href_{location})}$.

The search algorithm is executed by *DSContract*, which uses the search token t'_{kw} generated by the *DU* in the previous step as input. If such data exists, the algorithm returns data-related information successfully. In this algorithm, the *DU* sends a transaction to *DSContract* to trigger the execution, corresponding to steps (x)–(xii).

4.2.5. Decryption and Downloading Phase. The main function of the decryption and downloading phase is that *DU*s use their attribute private keys to decrypt the data-related information to obtain the address where the shared data stored on IPFS and the decryption key. The core algorithm is $Decrypt(SK_{uid,\omega}, CT_{(K_F|href_{location})}, PK) \rightarrow (K_F, href_{location})$ which was executed by the *DU*. The inputs of the algorithm are the private attribute key $SK_{uid,\omega}$ of the *DU*, the data-related information $CT_{(K_F|href_{location})}$, and the public system parameters *PK*. It outputs the decryption key K_F and the address $href_{location}$. Because the access policy $\mathscr{P}$ and the revocation

list R_i of each attribute are embedded in the ciphertext, if the attribute set of the *DU* that have not been revoked still satisfies access policy $\mathscr{P}$, he will decrypt and obtain K_F and $href_{location}$ successfully. In this way, the *DU* could download CT_F from IPFS and decrypt it to obtain the data F. This corresponds to step (xiii) in the system flowchart.

4.2.6. Permission Revocation Phase. The main function of the permission revocation phase is that the *DO* performs an attribute-level fine-grained permission revocation to a *DU* on a certain ciphertext. At the same time, it does not need to update the keys of other *DU*s related to the ciphertext. The core algorithm of this phase is $\text{Re}voke(K_F, href_{location}, \mathscr{P}, \{R_i\}_{i\in\mathscr{P}}, PK, i, uid) \rightarrow CT'_{(K_F|href_{location})}$ which is run by the *DO*. This is similar to the encryption algorithm, but a *DU*'s *uid* and the attribute i to be revoked are added to the parameters. The algorithm will add *uid* to the revocation list R_i and output a new ciphertext. The *DO* sends a transaction to *DSContract* to update the data-related information. In this way, if the remaining attribute set of the *DU* cannot satisfy the policy P, he can no longer decrypt the data after obtaining the ciphertext, while other *DU*s are not affected. This corresponds to step (xiv) in the system flowchart.

5. Implementation Details of Our Scheme

In order to achieve our goal, we will construct a CP-ABE which supports permission revocation and combine it with the EOS blockchain to implement our scheme. This section will elaborate on the details of our Smart Contracts deployed on EOS blockchain and concrete construction of BSSPD.

5.1. Smart Contract Design. To make the logic clearer, we divide the Smart Contract in the scheme into two parts: *UMContract* and *DSContract*. *UMContract* is used to manage *DU*s' identity, while *DSContract* is used to handle business operations related to data sharing. In the contract, we will use *_self* to represent the account of the *DO* who created the contract. We will describe the detailed design of these two contracts.

5.1.1. User Management Contract (UMContract). The *UMContract* is composed of five function interfaces: *SetTarget*, *GetUserByUid*, *Apply*, *Register*, and *Authenticate*. We initialize *UMContract* as follows.

Let three-tuple (A, uid, Pk_{com}) denote a *DU*, and create a multi_index named *table_user* for it in which A is an EOS account of the *DU*, *uid* is the unique ID assigned by the *DO*, and Pk_{com} is a public key of the *DU* used for communication with the *DO*. Let A be the primary key of *table_user* whose corresponding index is *account_idx*. Let *uid_idx* be a secondary index corresponding to *uid*. Let *target* be the target value of PoW.

(1) *SetTarget*: when *UMContract* receives action (*UMContract*, SetTarget, Auth, (*newTarget*)), this function interface will be triggered to execute. It can only be invoked by the *DO* who created the contract to adjust the difficulty of PoW. When there are too many users in the system, the *DO* can increase the difficulty of PoW

```
   Input: newTarget
   Output: bool
1 if msg.sender is not _self then
2   throw;
3 else
4   target = newTarget;
5   return true;
6 end
```

Algorithm 1: SetTarget.

(2) *GetUserUid*: when *UMContract* receives action (*UMContract*, GetUserByUid, Auth, (*account*)), this function interface will be triggered to execute. It is used to get all the information of a *DU* according to his *uid* and can only be invoked by the *DO* who created the contract

(3) *Apply*: when *UMContract* receives action (*UMContract*, Apply, Auth, (*from*, *pk*, *nonce*)), this function interface will be triggered to execute. It is invoked by the *DU* to apply for registration in the system

(4) *Register*: when *UMContract* receives action (*UMContract*, Register, Auth, (*account*, *id*)), this function interface will be triggered to execute. It is used to complete the registration of a *DU* and can only be invoked by the creator of the contract

(5) *Authenticate*: when *UMContract* receives action (*UMContract*, Authenticate, Auth, (*from*, *method*, *account*, *id*, *args*)), this function interface will be triggered to execute. It is used to authenticate the identity of a *DU*, which is invoked by another contract and returns the result to the invoker

5.1.2. Date Sharing Contract (DSContract). The *DSContract* is composed of six function interfaces: *SetPK*. *SetSK*, *AddData*, *PolicyUpdate*, *Search and EndSearch*, and *Remove*. We initialize *DSContract* as follows.

Let *PK* denote the system public parameters. Let two-tuple (*A*, *SK*) be the corresponding relationship between the *DU*'s account and his attribute private key, and the multi_index *table_sk* is created for it. Let A be the primary key of *table_sk* whose corresponding index is *ua_idx*. Let two-tuple (*fid*, *cf*) denote the shared data in which *fid* is the id of shared data and *cf* is the data-related information. Then, create a multi_index *data_table* for it, where *fid* is the primary key and *fid_idx* is the corresponding index. Let four-tuple (*id*, *A*, *t*, *fid*) be an index of *DU* related to shared data in which A is the EOS account of *DU*, t is the search token, and *fid* is the id of shared data in *data_table*, then create a multi_index *search_table* for it. Let *sa_idx*, *t_idx, sf_idx* be the secondary indices of *search_table*, corresponding to *A*, *t* , and *fid*, respectively.

```
Input: uid
Output: all information of DU
1 if msg.sender is not _self then
2   throw;
3 else
4   user_row = uid_idx.find(uid);
5   return user_row;
6 end
```

Algorithm 2: GetUserByUid.

```
Input: from, pk, nonce
Output: bool
1 u = account_idx.find(from)
2 if u != null then
3   u.Pk_com = pk ;
      account_idx.modify(u);
4   return true;
5 else
6   pow = SHA256(SHA256(from | pk | nonce)) ;
7   if pow > target then
8     return false;
9   else
10      u.A = from;
11      u.Pk_com = pk ;
12      account_idx.emplace(u);
13      return true;
14  end
15 end
```

Algorithm 3: Apply.

```
Input: account, id
Output: bool
1 if msg.sender is not _self then
2   throw;
3 else
4   u = account_idx.find(account);
5   if u==null then
6     return false;
7   else
8     u.uid=id;
10      account_idx.modify(u);
11      return true;
12  end
13 end
```

Algorithm 4: Register.

(1) *SetPK*: when *DSContract* receives action (*DSContract*, SetPK, Auth, (*newPk*)), this function interface will be triggered to execute. It can only be invoked by the *DO* to set and update the system public parameters

(2) *SetSK*: when *DSContract* receives action (*DSContract*, SetSK, Auth, (*account*, *sk*)), this function interface will be triggered to execute. It can only be invoked by the *DO* to set and update the private keys of the *DU*

```
Input: from, method, account, id, args
Output: null
u=account_idx.find(account)
1 if u != null then
2   if u.id == id then
3     send action (from, method, (_self, true, args));
4   else
5     send action (from, method, (_self, false, args));
6   end
7 else
8   send action (from, method, (_self, false, args));
9 end
```

Algorithm 5: Authenticate.

(3) *AddData*: when *DSContract* receives action (*DSContract*, AddData, Auth, ($account$, t_{kw}, $CT_{(K_F|href_{location})}$)), this function interface will be triggered to execute. It is used to publish the sharing data and add the indices for the relevant *DU*s. There can be multiple index relationships. For clarity, we only add an index for one *DU* here. It can only be invoked by the *DO*

(4) *PolicyUpdate*: when *DSContract* receives action (*DSContract*, PolicyUpdate, Auth, (fid, $CT_{(K_F|href_{location})}$)), this function interface will be triggered to execute. It can only be invoked by the *DO* and used to update the access policy for a certain shared data. In this way, the *DO* can revoke the access permission of a *DU* to this shared data

(5) *Search* and *EndSearch*: when *DSContract* receives action (*DSContract*, Search, Auth, (*from*, *uid*, t_{kw})), this function interface will be triggered to execute. These two function interfaces work together to complete the retrieval of shared data. Because we have divided BSSPD into two contracts, it needs to invoke *UMContract* to verify the identity of the *DU* during the retrieval

(6) *Remove*: when *DSContract* receives action (*DSContract*, Remove, Auth, (*fid*)), this function interface will be triggered to execute. It is used to remove a shared data and the search indices related to this data. It can only be invoked by the *DO*

5.2. Concrete Construction of BSSPD. In this section, we will show the concrete construction of our scheme, including the algorithms that the *DO* and *DU*s need to execute at each phase and their interactions with the EOS blockchain. Our initialization is as follows.

Let $N = p_1 p_2$ (p_1 and p_2 are distinct primes) G_1 and G_2 be cyclic groups of order N. Let G_{p_1} and G_{p_2} denote the subgroups of G_1 with order g and Y. Let $e : G_1 \times G_1 \to G_2$ be a

bilinear pairing and $I = \{1, \cdots, m\}$ be the set of all attributes. Let F be a pseudorandom function, where $F : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$, and $U = \{uid_1, \cdots, uid_n\}$ be the uid set of all DUs obtained from $UMContract$.

(1) $setup(1^\lambda, I)$

For each attribute $\{i \mid i \in I\}$, the algorithm first randomly picks two elements $t_i, \gamma_i \in Z_{p_1}$ and computes $T_i = g^{t_1}, h_i \in g^{\gamma_i}$. Next, it picks $\alpha, a \in Z_{p_1}$ randomly, then computes $e(g, g)$.

The public key is PK:

$$PK = \left(N, e(g,g)^\alpha, g, u = g^a, \left\{T_i = g^{t_i}, h_i \in g^{\gamma_i}\right\}_{i \in I}\right). \quad (2)$$

The system master key is MSK:

$$MSK = \left(\alpha, a, \{t_i\}_{i \in I}, Y\right). \quad (3)$$

Among them, t_i and T_i are used for calculations related to attributes, $\gamma_i$and h_i are used for calculations related to attribute revocation, u is used for calculations related to DU's identity, and Y is used for randomization of DU's private key.

Then, send the following transaction to EOS blockchain and store the public key in the $DSContract$:

$$Tx = (\mathrm{Re}\, f_{block}, t, Sig(chain\_id, Tx), Action(DSContract, SetPK, Auth_{DO}, (PK))). \quad (4)$$

(2) $KeyGen(uid, \omega, MSK, PK)$

Firstly, send the following transaction to EOS blockchain to obtain the DU's information including A, Pk_{com}, and uid from $UMContract$:

$$Tx = (\mathrm{Re}\, f_{block}, t, Sig_{DO}(chain\_id, Tx), Action(UMContract, GetUserByUid, Auth_{DO}, (uid))). \quad (5)$$

After that, the algorithm randomly chooses $r \in Z_{p_1}$ and computes $K_0 = g^{a+\alpha r}$. For each attribute $i \in \omega$, randomly pick $r_i \in Z_{p_1}$ and $Y_{i,1}, Y_{i,2}, Y_{i,3} \in G_{p_2}$, then compute the following:

$$K_{i,1} = g^{\alpha r + t_i r_i + a r_i} Y_{i,1}, \quad K_{i,2} = g^{r_i} Y_{i,2}, \quad K_{i,3} = \left(u^{uid} h_i\right)^{r_i} Y_{i,3}. \quad (6)$$

Let $K_i = \{K_{i,1}, K_{i,2}, K_{i,3}\}$, then DU's attribute private key will be $SK_{uid,\omega} = (K_0, \{K_i\}_{i \in \omega})$. As can be seen, the attribute-related part of the private key is embedded with the DU's identity.

Then, the algorithm randomly picks a secret key SK_{search} for search where $SK_{search} = SK_i \leftarrow \{0,1\}^\lambda$. Let $SK_{uid} = E$.

```
Input: newPk
Output: bool
1 if msg.sender is not _self then
2   throw;
3 else
4   PK = newPk;
5     return true;
6 end
```

Algorithm 6: SetPK.

$Enc_{Pk_{com}}(SK_{uid,\omega} \mid SK_{search})$, and send the following transaction to set and update the DU's private keys:

$$Tx = (\mathrm{Re}\, f_{block}, t, Sig_{DO}(chain\_id, Tx), Action(DSContract, SetSK, Auth_{DO}, (A, SK_{uid}))). \quad (7)$$

(3) $Encrypt(F, (A_{l \times k}, \rho), \{R_{\rho(x)}\}_{x \in 1, \cdots, l}, PK)$

The algorithm randomly chooses a private key K_F of ε, and encrypts the sharing data $CF = \varepsilon.Enc_{K_F}(F)$, then uploads the CF to the IPFS network, and the returned address is $href_{location}$, set $M = (K_F \mid href_{location})$.

The algorithm first randomly picks $s, v_2, \cdots, v_k \in Z_{p_1}$ and lets $\mathbf{v} = (s, v_2, \cdots, v_k)^T$. For $i = 1$ to l, it calculates $\lambda_x = \mathbf{A}_x \cdot \mathbf{v}$ where $\mathbf{A}_x$ is the vector corresponding to the $\mathbf{x}$th row of $\mathbf{A}_{l \times k}$. Assume that $R_{\rho(x)} = \{uid_1, \cdots, uid_{l_{\rho(x)}}\}$ in which the number of revocable users $l_{\rho(x)}$ is variable. For each $uid_j \in R_{\rho(x)}$, it randomly chooses a $\eta_{x,j} \in Z_{p_1}$, where $j = \{1, 2, \cdots, l_{\rho(x)}\}$ and $\sum_{j=1}^{l_{\rho(x)}}$, and then computes

$$C_0 = M \cdot e(g,g)^{\alpha s}. \quad C_1 = g^s. \quad (8)$$

For each attribute $\rho(x)$, it computes that

$$C_{x,0} = g^{\lambda_x}, \quad C_{x,1} = T_{\rho(x)}^{\lambda_x}. \quad (9)$$

When $R_{\rho(X)} \neq \varnothing$, it computes for each revoked $uid_j \in R_{\rho(x)}$ as follows:

$$C_{x,j,1} = g^{\eta_{x,j}}, \quad C_{x,j,2} = \left(u^{uid_j} h_{\rho(x)}\right)^{\eta_{x,j}}. \quad (10)$$

```
Input: account, sk
Output: bool
if msg.sender is not _self then
  throw;
else
  u=ua_idx.find(account);
  if u!=null then
    u.SK=sk
    ua_idx.modify(u)
    return true;
  else
    u.A=account;
    u.SK=sk;
    ua_idx.emplace(u)
    return true;
  end
end
```

Algorithm 7: SetSK.

```
Input: account, t_kw, CT_(K_F|href_location)
Output: bool
if msg.sender is not _self then
  throw;
else
  data_row.cf = CT_(K_F|href_location);
  data_table.emplace(data_row);
  search_row.A = account;
  search_row.t = t_kw;
  search_row.fid = data_row.fid.
  search_table.emplce(search_row);
  return true;
end
```

Algorithm 8: AddData.

```
Input: fid, CT_(K_F|href_location)
Output: bool
if msg.sender is not _self then
  throw;
else
  data_row = data_table.find(fid);
  if data_row == null then
    return false;
  else
    date_row.cf = CT_(K_F|href_location);
    data_table.modify(data_row);
    return true;
  end
end
```

Algorithm 9: PolicyUpdate.

The ciphertext CF is set to

$$CT_{(K_F|href_{location})} = \left(C_0, C_1, \left\{C_{x,0}, C_{x,1}, \left\{C_{x,j,1}, C_{x,j,2}\right\}_{j=\left\{1,\cdots,l_{\rho(x)}\right\}}\right\}_{x\in\{1,..,l\}}\right). \tag{11}$$

(4) $IndexGen(A, kw, K_{search}, CT_{(K_F|href_{location})})$

The algorithm calculates a search token for a keyword kw of the sharing data.

$$t_{kw} = F(K_{search}, kw). \tag{12}$$

After that, it will send the following transaction to EOS blockchain to publish the data-related information and add the indices for the relevant DUs:

$$Tx = (\mathrm{Re}\, f_{block}, t, Sig_{DO}(chain\_id, Tx),$$

$$Action\left(DSContract, AddData, Auth_{DO}, \left(A, t_{kw}, CT_{(K_F|href_{location})}\right)\right)). \tag{13}$$

(5) $Trpdr(kw', SK_{search})$

The DU obtains SK_{uid} from the $DSContract$ and decrypts it with his own private key.

$$(SK_{uid,\omega} \mid K_{search}) = E.Dec_{Sk_{com}}(SK_{uid}). \tag{14}$$

Then, it calculates the search token corresponding to kw'.

$$t'_{kw} = F\left(K_{search}, kw'\right). \tag{15}$$

(6) $Search(t'_{kw})$

Send the following transaction to EOS blockchain:

$$Tx = (\mathrm{Re}\, f_{block}, t, Sig_{DU}(chain\_id, Tx),$$

$$Action\left(DSContract, Search, Auth_{DU}, \left(t'_{kw}\right)\right)). \tag{16}$$

If the search is successful, the DU will obtain the data-related information $CT_{(K_F|href_{location})}$.

(7) $Decrypt(SK_{uid}, CT_{(K_F|href_{location})}, PK)$

```
   Input: from, uid, t_kw
   Output: data_rows
1 send action (UMContract,Authenticate,Auth,(_self,Search,from,id, t_kw))
2 if get false then
3   throw;
4 else
5   t_itr=t_idx.find(t_kw);
6   whilet_itr != search_table.end() andt_itr.t == t_kwandt_itr.A == from
7     data_row=search_table.find(t_itr.fid);
8     data_rows.add(data_row);
        t_idx++;
9   end
10   returndata_rows;
11 end
```

Algorithm 10: Search and EndSearch.

```
   Input: fid
   Output: bool
1 ifmsg.sender is not _selfthen
2   returnfalse;
3 else
4   s_itr = sf_idx.find(fid);
5   whiles_itr != sf_idx.end() ands_itr.fid == fid
6     sf_idx.erase(s_itr);
7   end
8   data_row = fid_idx.find(fid)
9   fid_idx.erase(data_row)
10   returntrue;
11 end
```

Algorithm 11: Remove.

Let $L = \{x \mid \rho(x) \in \omega, uid \notin R_{\rho(x)}\}$, then $\omega' = \{\rho(x)\}_{x \in L}$ denote the attribute set of the *DU* that has not been revoked. Assume that ω' still satisfies the access policy $(A_{l\times k}, \rho)$; for any x from 1 to l, it will calculate $D_{x,1}$ and $D_{x,2}$.

$$D_{x,1} = \prod_{j=1}^{l_{\rho(x)}} \left(\frac{e\left(K_{\rho(x),2}, C_{x,j,2}\right)}{e\left(K_{\rho(x),3}, C_{x,j,1}\right)} \right)^{1/uid-uid_j},$$
$$D_{x,2} = \frac{e\left(K_{\rho(x),1}, C_{x,0}\right)}{e\left(K_{\rho(x),2}, C_{x,1}\right)}. \tag{17}$$

Let μ_x be the restitution coefficient corresponding to the **x**th row in $A_{l\times x}$, and finally obtain the plaintext.

$$M = (K_F \mid href_{location}) = C_0 \cdot \frac{1}{e(K_0, C_1)} \cdot \prod_{x \in L} (D_{x,1}, D_{x,2})^{\mu_x}. \tag{18}$$

The *DU* can download *CF* from IPFS according to $href_{location}$ and then use K_F to decrypt *CF* and obtain the shared data *F*.

$$F = \varepsilon.Dec_{K_F}(CF). \tag{19}$$

(8) Re $voke(M, (A_{l\times k}, \rho), \{R_{\rho(x)}\}_{x\in 1,\cdots,l}, PK, uid, i)$

Take the revoking of the attribute *i* of a *DU* to the sharing data *F* as an example; the *DO* needs to add the *uid* of the *DU* to the revocation list corresponding to the attribute *i* and execute the CP-ABE part of *Encrypt* to encrypt the data-related information *M*. Then, the *DO* sends a transaction as following to the EOS blockchain.

$$Tx = (\text{Re}\, f_{block}, t, Sig_{DO}(chain\_id, Tx),$$
$$Action\left(DSContract, PokicyUpdate, Auth_{DO}, \left(fid, CT'_{(K_F \mid href_{location})}\right)\right)). \tag{20}$$

6. Security and Performance Analysis of the Proposed Scheme

6.1. Security and Privacy Analysis of BPSSD

6.1.1. Correctness. Let$L = \{x \mid \rho(x) \in \omega, uid \notin R_{\rho(x)}\}$, then ω' $= \{\rho(x)\}_{x\in L}$ denote the attributes set of the *DU* that has not been revoked. Assume that ω' still satisfies the access policy $(A_{l\times k}, \rho)$; for any x from 1 to l, then

$$D_{x,1} = \prod_{j=1}^{l_{\rho(x)}} \left(\frac{e\left(g^{r_{\rho(x)}} Y_{\rho(x),2}, \left(u^{uid_j} h_{\rho(x)}\right)^{\eta_{x,j}}\right)}{e\left(\left(u^{uid} h_{\rho(x)}\right)^{r_{\rho(x)}} Y_{\rho(x),3}, g^{\eta_{x,j}}\right)} \right)^{1/uid-uid_j}$$
$$= e(g,u)^{-r_{\rho(x)} \sum_{j=1}^{l_{\rho(x)}} \eta_{x,j}} = e(g,g)^{-ar_{\rho(x)}\lambda_x},$$

$$D_{x,2} = \frac{e\left(g^{\alpha r + t_{\rho(x)} r_{\rho(x)} + a r_{\rho(x)}} Y_{\rho(x),1}, g^{\lambda_x}\right)}{e\left(g^{r_{\rho(x)}} Y_{\rho(x),2}, T_{\rho(x)}^{\lambda_x}\right)} = e(g,g)^{\left(\alpha r + t_{\rho(x)} r_{\rho(x)} + a r_{\rho(x)}\right)\lambda_x - t_{\rho(x)} r_{\rho(x)} \lambda_x} = e(g,g)^{\left(\alpha r + a r_{\rho(x)}\right)\lambda_x},$$

$$M' = (K_F \mid href_{location}) = C_0 \cdot \frac{1}{e(K_0, C_1)} \cdot \prod_{x \in L} (D_{x,1} D_{x,2})^{\mu_x} = Me(g,g)^{\alpha s} \cdot \frac{1}{e(g^{\alpha + \alpha r}, g^s)} \cdot e(g,g)^{\alpha r \sum_{x \in L} \lambda_x \mu_x} = Me(g,g)^{\alpha s} \cdot \frac{1}{e(g^{\alpha + \alpha r}, g^s)} \cdot e(g,g)^{\alpha r s} = M. \tag{21}$$

After the proof, the data-related information M can be decrypted by the *DU*.

6.1.2. Security Analysis. The CP-ABE algorithm used in this paper is based on the scheme [37], referring to the revocation idea in [35] that introduces a revocation list for each attribute. The scheme [37] has proved to be completely secure. The detailed proof process can refer to the security analysis in [37], which is based on the standard model, and the security depends on three static assumptions.

This paper focuses on security data sharing based on blockchain. The security of CP-ABE is not within the main scope of this article. We will conduct a brief analysis of the security after adding an attribute revocation mechanism to the scheme [37].

If an adversary $\mathscr{A}$ can win the game with a nonnegligible advantage in the security model in [37], he must be able to calculate $e(g,g)^{\alpha s}$. To obtain such a pairing, the adversary needs to utilize $K_0 = g^{a + r\alpha}$ in the private key and $C_1 = g^s$ in the ciphertext, both of which can get $e(g,g)^{\alpha s} e(g,g)^{\alpha r s}$. This means that $\mathscr{A}$ needs to get $e(g,g)^{\alpha r s}$. Then, $\mathscr{A}$ needs to get $D_{x,1}$ and $D_{x,2}$ corresponding to each attribute and get $e(g,g)^{\alpha r \lambda_x}$ by calculation. If $\mathscr{A}$ does not satisfy the challenge attributes, he cannot obtain the correct attribute keys to calculate $e(g,g)^{\alpha r \lambda_x}$ conforming to the access policy and recover $e(g,g)^{\alpha r s}$.

For collusion attacks, when generating private keys for each *DU*, a random element r is contained in K_0 and random elements r_i and $Y_{i,1}$, $Y_{i,2}$, $Y_{i,3}$ are added to the attributes, so that different *DU*s cannot combine their private keys to launch attacks.

The attribute private key $K_{i,3} = (u^{uid} h_i)^{r_i} Y_{i,3}$ which is related to the revocation contains the *DU*'s identity information *uid*. When $R_{\rho(x)} \neq \varnothing$, each $uid_j \in R_{\rho(x)}$ in the revocation list contains $C_{x,j,1} = g^{\eta_{x,j}}$ and $C_{x,j,2} = (u^{uid_j} h_{\rho(x)})^{\eta_{x,j}}$, which need to be eliminated when decrypting. $1/(uid - uid_j)$ is used when eliminating. If *uid* is in the revocation list, $1/(uid - uid_j)$ will not be calculated to achieve the purpose of revoking the attribute j of *uid*.

6.1.3. Other Security Problem

(1) Data Security. Data security includes the confidentiality, integrity, and availability of the shared data. In our scheme, the large-capacity sharing data of the *DO* is encrypted using an efficient asymmetric encryption algorithm such as AES and uploaded to IPFS. The IPFS will split the encrypted data and store them on different IPFS nodes in a distributed manner. The access will be routed through the dynamic hash table maintained by each node, and a certain redundancy mechanism will ensure fault tolerance. Besides, IPFS also provides version control like Git. Thus, data encryption and storage in blocks ensure the confidentiality of the shared data. The integrity is guaranteed by dynamic hash table routing, and the tampered data blocks will not be available. The redundant storage and incentive mechanisms of IPFS ensure that users can access their data at any time. As long as IPFS is secure, then the data stored on it in our scheme is secure.

(2) Privacy Analysis. In a data-sharing system, privacy includes the content of the *DO*'s shared data and the traces of the *DU* when using the data. In our scheme, the *DO* will encrypt the address of the shared data and the corresponding decryption key with CP-ABE according to the established access policy. Then, the ciphertext is stored on the blockchain, and only the *DU*s whose attribute set satisfies the access policy can obtain the data. The content of the data will not be leaked. For the traces generated by *DU*s, we encrypt the keywords corresponding to the sharing data. The *DU* invoked the trapdoor function to calculate the search token for the keyword that he needs to retrieve and then uses the search token for retrieving on the blockchain without revealing any information he wants. More importantly, the user's identity is represented in the form of an address on the blockchain, and the real information of the user will not be exposed.

(3) Fine-Grained Access Control. In our scheme, the fine-grained access control of shared data is realized by CP-ABE. The *DO* can make different access policies through LSSS and assign different attributes to *DU*s. Meanwhile, fine-grained access control should also include fine-grained revocation. The proposed scheme draws on the identity-based broadcast encryption scheme, in which the *DO* assigns a unique *uid* for each *DU*, and the *uid* will be used as a user attribute, embedded in the ciphertext together with the general attributes. Each general attribute in the ciphertext carries a revocation list, and the *DU* whose *uid* in this list no longer has the corresponding attribute, so that it achieves the purpose of directly revoking a *DU*'s attribute.

(4) Avoid a Single Point of Failure. Compared with traditional cloud storage solutions, there is no centralized third party in our proposed scheme. Blockchain and IPFS used in BSSPD are all distributed technologies. Even if some of the nodes fail, the availability of the whole scheme will not be affected. More importantly, the BitTorrent protocol adopted by IPFS can enjoy

a high throughput only by requiring paying a small number of fees to incentive storage nodes. Simultaneously, the EOS blockchain is free to users, only the *DO* needs to mortgage some system tokens in exchange for storage and CPU resources, and these tokens can also be redeemed.

(5) User-Centric. In our proposed scheme, the *DO* can generate public parameters and the system master key and generate and distribute the private keys for *DU*s according to their attributes. Moreover, the *DO* can formulate access policies arbitrarily to assign and revoke the permission of *DU*s. All of these are controlled by the *DO* without any trusted third party. In this manner, the *DO* has complete control over his shared data.

(6) Identity Authentication. The user generates his identity in the blockchain through an asymmetric encryption algorithm with generating key pairs, whose cost is too low. In our proposed scheme, since the *uid* is embedded in the ciphertext of CP-ABE as an attribute, the *DU*s may register a large number of *uid*s and use different *uid*s to search and decrypt the shared data, which increases the burden of the *DO*. In order to prevent such attacks, BSSPD requires identity authentication. Before applying for registration, the *DU* needs to perform a PoW, which is similar to Bitcoin mining. The *DO* can adjust the difficulty of PoW according to the total number of *DU*s in the system. User management and identity authentication are carried out on the blockchain, and only authenticated users can perform operations. These are all executed in Smart Contract ensuring transparency and security.

6.2. Experiments and Performance Analysis of BPSSD

6.2.1. Functional Comparison. We compared the scheme proposed in this article with the recent blockchain-based data-sharing models from the following aspects, including security and privacy, identity management, fine-grained access control, immediate access revocation, and ciphertext keyword retrieval, as shown in Table 2.

From the comparison in the table, it can be concluded that due to the blockchain's decentralized and trustless nature, the data-sharing models based on blockchain allow *DO*s to formulate access control policies for their data on-chain, so they all can guarantee security and privacy. Early schemes like Ref. [18] mostly only described the model's outline without the specific implementation details. Generally, they only describe how blockchain can benefit security and privacy during the sharing, so the function is relatively simple. Reference [21] implemented a role-based access control model on the blockchain, but it turns out that RBAC is not suitable for implementing fine-grained access control and revocation in a distributed environment. Reference [28] utilized CP-ABE to achieve fine-grained access control, but it does not achieve permission revocation. However, in the access control scheme based on CP-ABE, an immediate access revocation is indispensable.

In our proposed scheme, we utilized CP-ABE to achieve fine-grained access control and realized the identity management of *DU*s. The *DO* assigns and manages unique *uid*s and attributes for registered *DU*s. Maintaining a revocation list for each attribute in the ciphertext can directly revoke a particular attribute of a *DU* without updating others' keys. BSSPD uses ciphertext keyword search to protect the privacy of *DU*s on-chain. Therefore, our proposed scheme has better applicability and usability.

Table 2: Functional comparison between BSSPD and other blockchain-based data-sharing scheme.

No.	BSSPD	Ref. [18]	Ref. [21]	Ref. [28]
Security and privacy	√	√	√	√
Identity management	√	×	×	×
Fine-grained access control	√	×	×	√
Immediate access revocation	√	×	×	×
Keyword ciphertext retrieval	√	×	×	√

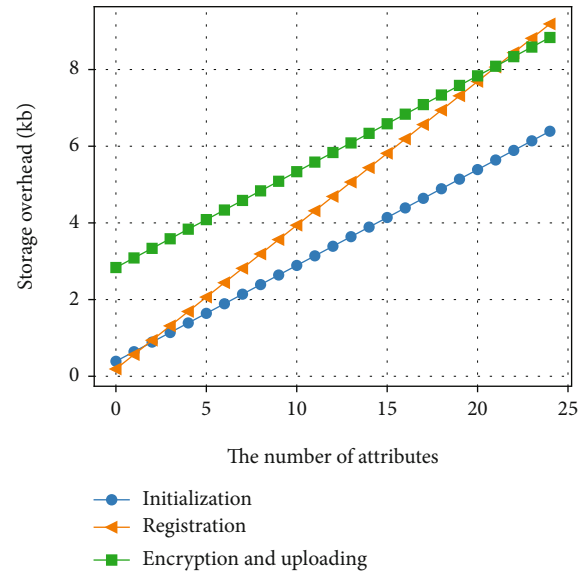

Figure 3: Storage overhead of BSSPD at each phase varies with the number of attributes.

6.2.2. Storage Analysis. BSSPD is a user-centric data-sharing scheme based on the EOS blockchain, and it stores the public system parameters, user information, and data-related information in the persistent database of Smart Contract. Because the storage resource on-chain is valuable and the acquisition of RAM in the EOS blockchain requires mortgaging system tokens, so it is necessary to analyze the size of the data stored in the Smart Contract.

We first define some symbols; we set $|G_1|$, $|G_2|$, $|G_{p_1}|$, $|G_{p_2}|$ to represent the bit length of an element in group G_1, G_2, G_{p_1}, and G_{p_2}, respectively. Let $|Z_N|$ be the bit length of an element in filed Z_N, $|K_{AES}|$ be the bit length of a key of AES algorithm, and $|Sk_{com}|$ and $|Pk_{com}|$ be the bit length of private key and public key of ECC, respectively; $|S|$ denote the number of attributes in system, and K_{search} denote the bit length of a secret key of pseudorandom function F.

According to the experiment simulation in our scheme, we set $|G_1| = |G_2| = |G_{p_1}| = |G_{p_2}| = 1024$bits; $|Z_N| = 128$ bits; $|K_{AES}| = 256$ bits; $|Sk_{com}| = 256$ bits; $|Pk_{com}| = 272$ bits; $|K_{search}| = 128$bits; the bit length of *account*, *uid*, *fid*, and search token t_{kw} to be 64; and the bit length of an IPFS address to be 256. The storage overhead of BSSPD at each phase varies with the number of attributes which is shown in Figure 3.

In our proposed scheme, there are three operations that interact with the blockchain to store data in the Smart Contract, which are as follows:

(1) Initialization

The *DO* uploads the system public parameters to the Smart Contract; the storage overhead is

$$|Z_N| + |G_1| + 2|G_{p_1}| + |S|\left(2 \mid G_{p_1} \mid\right) = (3200 + 2048 \mid S \mid) \text{ bits}. \tag{22}$$

(2) Registration

The *DU* uploads information to the Smart Contract when applying for registration, and the *DO* assigns a unique *uid* and private keys for the *DU*. The storage overhead is

$$|account| + |uid| + |Pk_{com}| + |K_{search}| + \left|G_{p_1}\right| + 3\left|G_{p_1}\right||S| = (1552 + 3072|S|) \text{ bits}.$$

(3) Encryption and uploading

The *DO* uploads data-related information and the private keys of the *DU* to the Smart Contract, as well as the indices for the *DU*. The storage overhead is

$$|K_{AES}| + |address| + 2\left|G_{p_1}\right| + 2\left|G_{p_1}\right||S| + 2\left|G_{p_1}\right||R| + |fid| + |t_{kw}| + |fid| + |account| = 2752 + 2048|S| + 2048|R|. \tag{24}$$

For simplicity, the figure shows that the storage overhead varies with the number of attributes when there are 10 *DU*s in the revocation list. As the number of *DU*s in the revocation list and the relevant *DU*s increases, the storage overhead will also increase to a certain extent.

The RAM in the EOS blockchain is obtained by collateralizing system tokens, and the current price is 0.05 EOS/KB. The *DO* can purchase RAM according to the scale of his system. Unlike Ethereum transactions that need to consume ETH as gas, the tokens mortgaged when acquiring RAM in EOS can still be redeemed at the original price. Above all, the proposed scheme is feasible and practical.

6.2.3. Performance Analysis. As we all know, the computing resource on the blockchain is also precious, and the computational efficiency of the existing blockchains is often criticized. For example, Bitcoin takes 10 minutes to produce a block. Ethereum has dramatically improved the block generation time, but it also takes about 15 seconds. In this section, we will conduct experiments on our proposed scheme and evaluate the scheme's performance and user scale.

We used 5 nodes to build an EOS private chain in a laboratory environment. The 5 nodes we chose were all MacBook Pro (2017) with Intel (R) Core (TM) i5 CPU that clocks at 3.1 GHz and has 16.0 GB of RAM. The version of the EOS blockchain we chose is v2.0.6. The code of the indices of the two tables related to the sharing data in our Smart Contract is as follows:

In our scheme's initialization phase, the operation on-chain is to set and update the public system parameters. The previous section shows that the storage overhead will continue to expand as the attributes increase. However, it can be seen from Figure 4 that as the attributes increase, the computing overhead will not be significantly affected in this phase.

In the encryption and uploading phase of our scheme, the operations that need to be performed on-chain are uploading the data-related information to Smart Contract and establishing the keyword indices for the data-related *DU*s. As shown in Figure 5, the increase in the number of attributes will not have too much influence on the computing overhead of *AddData*. In the case of a different number of attributes, the computing overhead of *AddData* is generally stable. What impacts the computing overhead of *AddData* is the scale of *DU*s, especially the number of *DU*s related to the sharing data. It can be seen from Figure 5 that the computing overhead of 500 *DU*s is obviously higher than that of 100 *DU*s, and the time cost is mainly spent on establishing search indices for the relevant *DU*s.

Since BSSPD sets the search token as a secondary index of the *search_table* in the Smart Contract, no matter how many pieces of index data exist in the system, the time complexity of retrieving according to the search token is $O(1)$. As shown in Figure 6, when there are 10 billion pieces of index data, the search time is not much different from that of 1 million, and the search time is in milliseconds.

The deletion of a certain data in our scheme is to remove the data-related information and the indices to the data. As shown in Figure 7, as the number of data-related *DU*s continues to expand, the computing overhead of deletion will increase too. The main time cost is spent on deleting the search indices to the data.

Since only the ciphertext data needs to be updated according to the shared data's primary key id when revoking a *DU's* attribute of a specific shared data, there is no need to operate on the relevant indices, and its computing overhead is similar to set and update the public system parameters in the initialization phase, which is stable.

In summary, in our proposed scheme, the total number of attributes will not impact much on the computing overhead on-chain. According to experience, it only affects

```
typedef eosio::multi_index<"sharedatas"_n, my_data> data_table;
ttypedef eosio::multi_index<"searchindexs"_n, s_index, indexed_by<"username"_n,
const_mem_fun<s_index, uint64_t, &s_index::by_secondary>>,indexed_by <"searchtoken "_n, const_mem_fun<s_index, checksum256, &s_index::by_thirdary>>,indexed_by<"fid"_
n, const_mem_fun<s_index, uint64_t, &s_index::by_forthary>>> search_table;
```

Code 1: The code of the tables in Smart Contract.

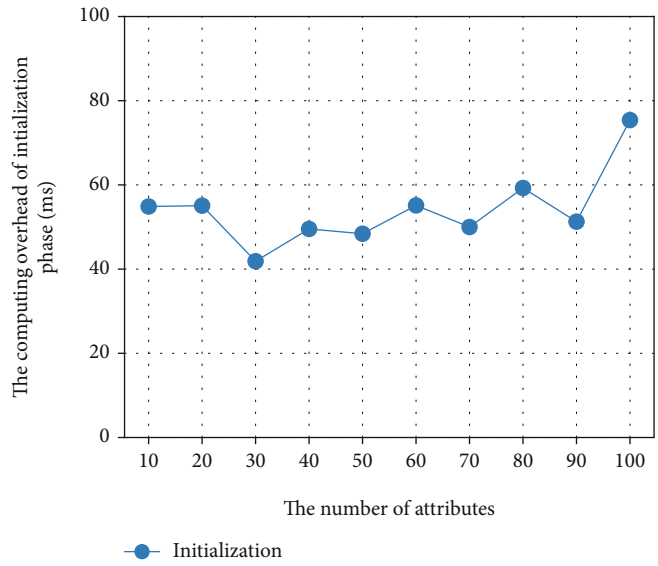

Figure 4: Computing overhead of the initialization phase varies with the number of attributes.

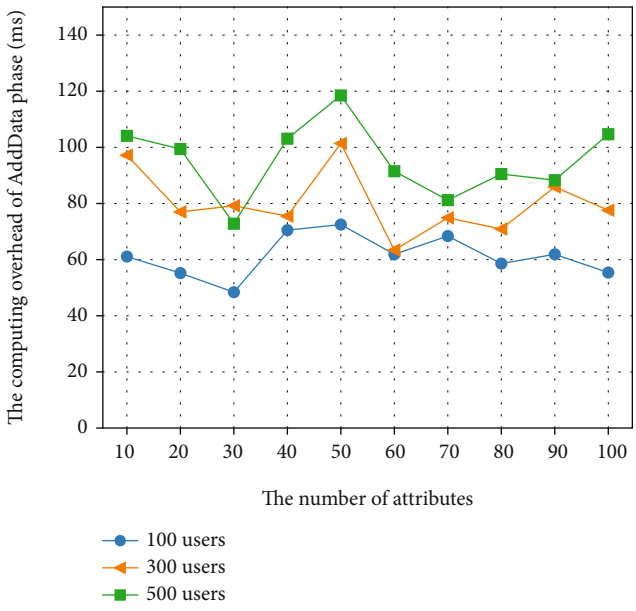

Figure 5: Computing overhead of the AddData varies with the number of attributes.

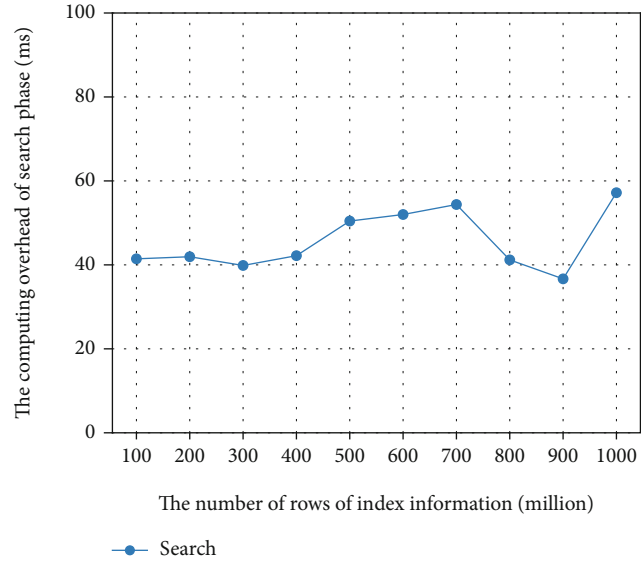

Figure 6: Computing overhead of the search phase varies with the number of rows of index information.

operations off-chain, such as key generation, encryption, and decryption. However, the expansion of the user scale will increase the time cost of some operations. Specifically, it is increased with the number of *DU*s related to certain shared data because search indices will be established. When the related search indices of a specific data increase to 500, the computing overhead is still in milliseconds. For all operations on-chain in our scheme, the computing overhead is less than 100 milliseconds. The configuration of the EOS main network's block producer is much better than the laptop we

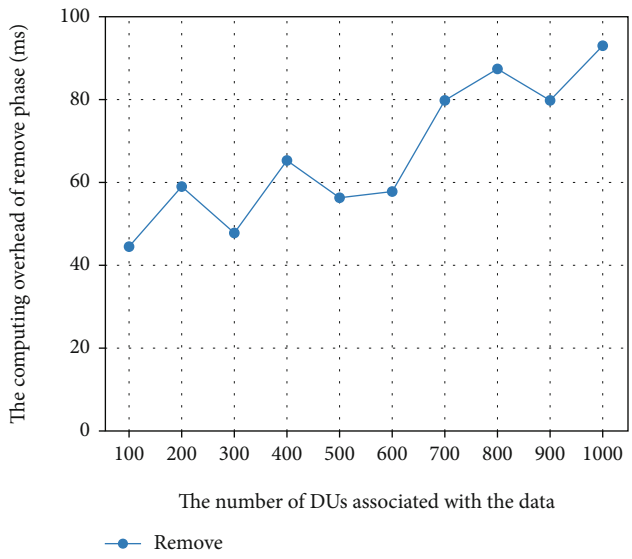

Figure 7: Computing overhead of the remove varies with the number of DUs associated with the shared data.

use, so when the contract is deployed on the main network of EOS, the computing overhead will be much lower than that of our simulation. Now, since EOS takes 0.5 seconds to generate a block, our scheme's operation will be confirmed soon after execution. Therefore, the experiment has proved that our scheme has a good performance.

7. Conclusion

In the AI-driven era, a user-centered sharing model is proposed to open data while ensuring data privacy. We combined blockchain, CP-ABE, and IPFS to propose a blockchain-based security data-sharing scheme with fine-grained access control and permission revocation. In our proposed scheme, the *DO* encrypts his data and uploads it to IPFS, then encrypts the returned address and decryption key by CP-ABE. Only *DU*s whose attributes satisfy the access policy can decrypt and obtain the data. There is no centralized node in the scheme, and the *DO* has complete control over his shared data, which promises privacy and security. To achieve the goal, we have implemented our scheme on the EOS blockchain. The security and performance analysis proves that our scheme is feasible and practical and has a good performance. We can also add a cryptocurrency to introduce an economic system for data sharing and further enrich our scheme's functions. At the same time, there are many shortcomings in our scheme. For example, the CP-ABE we designed with permission revocable does not have the best performance. There are also many types of research on CP-ABE [38–42]. We can use a CP-ABE with better performance to improve our scheme. Besides, for the searchable encryption algorithm used in our scheme, the *DO* needs to distribute a secret key for each *DU* and store it on-chain. It also needs to maintain large amounts of indices for each shared data, which can be further optimized. At present, some researchers have proposed using blockchain to solve the fairness problem in searchable encryption algorithm [43–47]. In the future, we will study and discuss the endowment of a better ciphertext searchable algorithm to further optimize our scheme. Simultaneously, to make our scheme more practical, we can combine some studies [48–52] with ours and put forward a data governance scheme that is more in line with the practical application.

Data Availability

The data that support the findings of this study are available from the corresponding author, upon reasonable request.

Conflicts of Interest

The author(s) declare(s) that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61272519, 61170297, 61472258, and 61802094 and National Natural Science Foundation of Zhejiang Province under Grant LY20F020012.

References

[1] J. Li, Y. Zhang, X. Chen, and Y. Xiang, "Secure attribute-based data sharing for resource-limited users in cloud computing," *Computers & Security*, vol. 72, pp. 1–12, 2018.

[2] S. Sundareswaran, A. Squicciarini, and D. Lin, "Ensuring distributed accountability for data sharing in the cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 4, pp. 556–568, 2012.

[3] Cheng-Kang Chu, S. S. M. Chow, Wen-Guey Tzeng, Jianying Zhou, and R. H. Deng, "Key-aggregate cryptosystem for scalable data sharing in cloud storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 468–477, 2014.

[4] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *2010 Proceedings IEEE INFOCOM*, pp. 1–9, San Diego, CA, 2010.

[5] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou, "Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 131–143, 2013.

[6] Z. Cai, Z. He, X. Guan, and Y. Li, "Collective data-sanitization for preventing sensitive information inference attacks in social networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 1–590, 2018.

[7] Z. Cai and X. Zheng, "A private and efficient mechanism for data uploading in smart cyber-physical systems," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 2, pp. 766–775, 2020.

[8] X. Zhou, W. Liang, K. Wang, R. Huang, and Q. Jin, "Academic influence aware and multidimensional network analysis for research collaboration navigation based on scholarly big data," *IEEE Transactions on Emerging Topics in Computing*, no. 1, 2018.

[9] Z. Cai, X. Zheng, and J. Yu, "A differential-private framework for urban traffic flows estimation via taxi companies," *IEEE*

Transactions on Industrial Informatics, vol. 15, no. 12, pp. 6492–6499, 2019.

[10] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," 2008, https://bitcoin.org/bitcoin.pdf.

[11] Y. Xu, C. Zhang, G. Wang, Z. Qin, and Q. Zeng, "A blockchain-enabled deduplicatable data auditing mechanism for network storage services," *IEEE Transactions on Emerging Topics in Computing*, 2020.

[12] Y. Xu, J. Ren, Y. Zhang, C. Zhang, B. Shen, and Y. Zhang, "Blockchain empowered arbitrable data auditing scheme for network storage as a service," *IEEE Transactions on Services Computing*, vol. 13, no. 2, pp. 289–300, 2020.

[13] Y. Xu, C. Zhang, Q. Zeng, G. Wang, J. Ren, and Y. Zhang, "Blockchain-enabled accountability mechanism against information leakage in vertical industry services," *IEEE Transactions on Network Science and Engineering*, 2020.

[14] Y. Xu, J. Ren, G. Wang, C. Zhang, J. Yang, and Y. Zhang, "A blockchain-based nonrepudiation network computing service scheme for industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3632–3641, 2019.

[15] M. Swan, "Blockchain thinking: the brain as a decentralized autonomous corporation [commentary]," *IEEE Technology and Society Magazine*, vol. 34, no. 4, pp. 41–52, 2015.

[16] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing privacy: using blockchain to protect personal data," in *2015 IEEE Security and Privacy Workshops*, pp. 180–184, San Jose, CA, 2015.

[17] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: using blockchain for medical data access and permission management," in *2016 2nd International Conference on Open and Big Data (OBD)*, pp. 25–30, Vienna, 2016.

[18] 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.

[19] A. Dubovitskaya, Z. Xu, S. Ryu, M. Schumacher, and F. Wang, "Secure and trustable electronic medical records sharing using blockchain," *AMIA Annual Symposium Proceedings*, vol. 2017, pp. 650–659, 2017.

[20] X. Liang, J. Zhao, S. Shetty, J. Liu, and D. Li, "Integrating blockchain for data sharing and collaboration in mobile healthcare applications," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pp. 1–5, Montreal, QC, 2017.

[21] K. Fan, Y. Ren, Y. Wang, H. Li, and Y. Yang, "Blockchain-based efficient privacy preserving and data sharing scheme of content-centric network in 5g," *IET Communications*, vol. 12, no. 5, pp. 527–532, 2017.

[22] G. Zhang, T. Li, Y. Li, P. Hui, and D. Jin, "Blockchain-based data sharing system for AI-powered network operations," *Journal of Communications and Information Networks*, vol. 3, no. 3, pp. 1–8, 2018.

[23] I. Zhou, I. Makhdoom, M. Abolhasan, J. Lipman, and N. Shariati, "A blockchain-based file-sharing system for academic paper review," in *2019 13th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pp. 1–10, Gold Coast, Australia, 2019.

[24] V. Patel, "A framework for secure and decentralized sharing of medical imaging data via blockchain consensus," *Health informatics journal*, vol. 25, no. 4, pp. 1398–1411, 2018.

[25] L. Tan, N. Shi, C. Yang, and K. Yu, "A blockchain-based access control framework for cyber-physical-social system big data," *IEEE Access*, vol. 8, pp. 77215–77226, 2020.

[26] M. Jemel and A. Serhrouchni, "Decentralized access control mechanism with temporal dimension based on blockchain," in *2017 IEEE 14th International Conference on e-Business Engineering (ICEBE)*, pp. 177–182, Shanghai, 2017.

[27] X. Sun, S. Yao, S. Wang, and Y. Wu, "Blockchain-based secure storage and access scheme for electronic medical records in ipfs," *IEEE Access*, vol. 8, pp. 59389–59401, 2020.

[28] S. Wang, Y. Zhang, and Y. Zhang, "A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems," *IEEE Access*, vol. 6, pp. 38437–38450, 2018.

[29] S. M. Pournaghi, M. Bayat, and Y. Farjami, "MedSBA: a novel and secure scheme to share medical data based on blockchain technology and attribute-based encryption," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 11, pp. 4613–4641, 2020.

[30] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, pp. 321–334, Berkeley, CA, 2007.

[31] B. Waters, "Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization," in *Proceedings of the 14th International Conference on Practice and Theory in Public Key Cryptography Conference on Public Key Cryptography, PKC'11*, pp. 53–70, Berlin, Heidelberg, 2011.

[32] V. Buterin, "Ethereum: a next-generation smart contract and decentralized application platform," 2013, https://github.com/ethereum/wiki/wiki/White-Paper.

[33] N. Szabo, "Smart contracts," 1994, https://szabo.best.vwh.net/smart.contracts.html.

[34] H. Gao, Z. Ma, S. Luo, and Z. Wang, "Bfr-mpc: a blockchain-based fair and robust multi-party computation scheme," *IEEE Access*, vol. 7, pp. 110439–110450, 2019.

[35] N. Attrapadung and H. Imai, "Conjunctive broadcast and attribute-based encryption," in *Proceedings of the 3rd International Conference Palo Alto on Pairing-Based Cryptography, pairing '09*, pp. 248–265, Berlin, Heidelberg, 2009.

[36] H. Li, F. Zhang, J. He, and H. Tian, "A searchable symmetric encryption scheme using blockchain," 2017, https://arxiv.org/abs/1711.01030.

[37] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: attribute-based encryption and (hierarchical) inner product encryption," in *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques, EUROCRYPT'10*, pp. 62–91, Berlin, Heidelberg, 2010.

[38] 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.

[39] Y. Xu, Q. Zeng, G. Wang, C. Zhang, J. Ren, and Y. Zhang, "An efficient privacy-enhanced attribute-based access control mechanism," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 5, article e5556, 2020.

[40] X. Yan, Y. Xu, X. Xing, B. Cui, Z. Guo, and T. Guo, "Trustworthy network anomaly detection based on an adaptive learning rate and momentum in IIoT," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6182–6192, 2020.

[41] Z. Cai and Z. He, "Trading private range counting over big IoT data," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 144–153, Dallas, TX, USA, 2019.

[42] G. Yu, X. Zha, X. Wang et al., "Enabling attribute revocation for fine-grained access control in blockchain-IoT systems," *IEEE Transactions on Engineering Management*, vol. 67, no. 4, pp. 1213–1230, 2020.

[43] L. Chen, W. K. Lee, C. C. Chang, K. K. R. Choo, and N. Zhang, "Blockchain based searchable encryption for electronic health record sharing," *Future Generation Computer Systems*, vol. 95, pp. 420–429, 2019.

[44] Y. Xu, G. Wang, J. Yang, J. Ren, Y. Zhang, and C. Zhang, "Towards secure network computing services for lightweight clients using blockchain," *Wireless Communications and Mobile Computing*, vol. 2018, 12 pages, 2018.

[45] X. Zhou, W. Liang, K. I. Wang, H. Wang, L. T. Yang, and Q. Jin, "Deep-learning-enhanced human activity recognition for Internet of healthcare things," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6429–6438, 2020.

[46] X. Yan, Y. Xu, B. Cui, S. Zhang, T. Guo, and C. Li, "Learning URL embedding for malicious website detection," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, pp. 6673–6681, 2020.

[47] 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," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 792–800, Honolulu, HI, 2018.

[48] X. Zhou, Y. Hu, W. Liang, J. Ma, and Q. Jin, "Variational LSTM enhanced anomaly detection for industrial big data," *IEEE Transactions on Industrial Informatics*, 2020.

[49] X. Zhou, Y. Li, and W. Liang, "CNN-RNN based intelligent recommendation for online medical pre-diagnosis support," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2020.

[50] X. Zhou, W. Liang, K. I. Wang, and L. T. Yang, "Deep correlation mining based on hierarchical hybrid networks for heterogeneous big data recommendations," *IEEE Transactions on Computational Social Systems*, vol. 8, no. 1, pp. 171–178, 2021.

[51] Y. Liang, Z. Cai, J. Yu, Q. Han, and Y. Li, "Deep learning based inference of private information using embedded sensors in smart devices," *IEEE Network*, vol. 32, no. 4, pp. 8–14, 2018.

[52] X. Yan, B. Cui, Y. Xu, P. Shi, and Z. Wang, "A method of information protection for collaborative deep learning under GAN model attack," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2019.