Hindawi
Security and Communication Networks
Volume 2021, Article ID 7249470, 11 pages
https://doi.org/10.1155/2021/7249470

Research Article

A Collusion-Resistant Blockchain-Enabled Data Sharing Scheme with Decryption Outsourcing under Time Restriction

Xieyang Shen,[1] **Chuanhe Huang**,[1] **Xiajiong Shen**,[2] **Jiaoli Shi**,[3] **and Danxin Wang**[1]

[1]*School of Computer Science, Wuhan University, Wuhan, China*
[2]*Henan Key Laboratory of Big Data Analysis and Processing, Henan University, Kaifeng, China*
[3]*School of Information Science and Technology, Jiujiang University, Jiujiang, China*

Correspondence should be addressed to Chuanhe Huang; huangch@whu.edu.cn

Received 14 June 2021; Accepted 26 July 2021; Published 27 August 2021

Academic Editor: Yinghui Zhang

With the ever-increasing demands on decentralization and transparency of cloud storage, CP-ABE (Ciphertext Policy-Attribute-Based Encryption) has become a promising technology for blockchain-enabled data sharing methods due to its flexibility. However, real-world blockchain applications usually have some special requirements like time restrictions or power limitations. Thus, decryption outsourcing is widely used in data sharing scenarios and also causes concerns about data security. In this paper, we proposed a secure access control scheme based on CP-ABE, which could share contents during a particular time slot in blockchain-enabled data sharing systems. Specifically, we bind the time period with both ciphertexts and the keys to archive the goal of only users who have the required attributes in a particular time slot can decrypt the content. Besides, we use time slots as a token to protect the data and access control scheme when users want to outsource the decryption phase. The security analysis shows that our scheme can provide collusion resistance ability under a time restriction, and performance evaluations indicate that our scheme uses less time in decryption compared to other schemes while ensuring security.

1. Introduction

Traditional blockchain-enabled data sharing schemes usually assume that CSP (cloud service provider) can be trusted to keep data confidential. However, this assumption causes more concerns about the security and integrity of data since more and more end users tend to outsource the decryption phase to CSP due to their resource-constrained devices, for example, more and more smart devices with the duty of data storage and computation collecting private information under smart city scenarios [1]. To mitigate users' concerns about their data privacy and security, an access control scheme that can either prevent curious CSP from scanning data stored on the cloud or disclose nothing during the outsourcing decryption must be proposed [2].

Attribute-based encryption is considered by scholars as a novel solution for solving the problems stated above. ABE was first proposed by Sahai and Waters [3] and further developed two categories: Ciphertext-Policy ABE (CP-ABE) and Key-Policy ABE (KP-ABE) [4], depending on whether the access policies are embedded with the ciphertext or the user's private key. ABE can prevent both unauthorized users and curious servers from accessing the data and support data owners to encrypt their data before sending them to cloud servers. In CP-ABE, the access policy is binding with the ciphertext so that data owners do not need to update the ciphertext when attributes are changed. Thus, CP-ABE is more suitable for cloud access control environments and can be deployed in many scenarios.

Besides, time restriction is more and more common nowadays due to the sensitivity of the data in blockchain-enabled data sharing systems, such as video content [5] and personal health record [6]. The fine-grained access control has been paid much more attention in attribute-based encryption schemes, but it is still not easy to get the goal of adding time restrictions in these schemes. Furthermore, the semitrusted cloud server providers make these issues more serious as the providers themselves are curious about the

content stored in the cloud. From the time restriction aspect, for example, a malicious cloud service provider can easily predict the policy update or attribute revoke operation of a company (as a data owner), then the provider can delay the updating operation for a few hours or even a few minutes to let the revoked users get the data illegally and shirk its responsibility to the high latency of the network. Compared with recent ABE schemes [7], our scheme set time as a part of the key to examine both the cloud servers and the users. In this case, the time restriction can be seen as an attribute of the user and an examined standard on a cloud server.

In this paper, we focus on designing a collusion resistance access control scheme based on ABE and ensuring the safety of data after decryption outsourcing. We propose a secure access control scheme based on CP-ABE under time restrictions. For the above goals, we bind the time slot with both ciphertexts and secret keys in a blockchain-enabled data sharing scheme so that only the legal user (which means satisfying the access policy and the time restriction at the same time) can decrypt the data. Besides, we use a time slot as a token to make sure that the outsourcing party cannot get any information from the calculation phase.

The main contributions are summarized as follows:

(1) We propose a blockchain-enabled data sharing scheme based on CP-ABE by binding the time slot with both ciphertexts and secret keys. In this way, users must meet any request between attributes and time slot to decrypt the data.

(2) We propose a method in the multiservers scenario to prevent a new kind of collusion that a malicious cloud server provider does not execute the owner's update/revoked order in time to gain some time for revoked users to get the data illegally.

(3) We propose a method to change the time slot in outsourcing into a token to guarantee that the calculation phase in the outsourcing party will not leak any information about the data and the access policy.

The rest of our paper is organized as follows.

Related work is introduced in Section 2. In Section 3, we first list some preliminaries and then proposed our system architecture. A detailed scheme is presented in Section 4. We also propose our collusion resistance updating method in this section. Security analysis and performance evaluations are conducted in Section 5. Conclusion and further discussion are in Section 6.

2. Related Work

Outsourcing is a common solution for power limited devices to complete the task they could not afford in blockchain-enabled data sharing schemes [8]. Despite the consideration of computation and storage, outsourcing services are also applied to many scenarios such as big data analysis [9], attack detections [10], machine learning [11]. CP-ABE [12] is regarded as one of the most practical models for access control schemes in blockchain-enabled data sharing, for it not only allows the data owner to define the access policy from several attribute authorities [13] but also does not need a trustworthy third party to realize decentralization and transparency requirements for blockchain [14]. DAC-MACS (Data Access Control for Multi-Authority Cloud Storage) designed by Yang et al. [5], is one of the multiauthority schemes which propose effective and secure data access control schemes for video content sharing. However, users are required to transfer their private keys to the cloud for generating a decryption token for efficiency. A series of constructions exist to realize fine-grained access control for data sharing with CP-ABE in different ways. Yang et al. focused on efficient revocation [15] and multiauthority [12], respectively. Shi et al. designed a version key mechanism for direct revocation [16]. Unfortunately, most of the above schemes did not take time into consideration.

Time is a quite unique factor in some scenarios like video content sharing [5], online storage service [17], and weather reporting [18]. It has become an important prominent factor, especially in blockchain-enabled data sharing that can even decide the worth of the data. However, it also raises concerns about data security and end devices affordability. With the proposal of sharing time-sensitive data in a particular time period, several ABE schemes have taken time into consideration. Liu et al. [19] proposed a time-based proxy reencryption scheme, so that in a particular time slot, the access policy can control the access for users. Conversely, with the change of time period, data owners need to reencrypt the ciphertext, which is not suitable for blockchain-enabled data sharing systems. Yang et al. [5] proposed a time domain multiauthority ABE method that binds time with ciphertext and secret key, but computation cost on both data owner and user increases linearly. Hong et al. [20] designed an access control scheme based on both time and attributes, where cloud servers play an important role, including generating a token and updating ciphertext online at each of the time periods. However, the time period of this scheme is defined at the beginning of the system initialization. Thus, the time period cannot fit most of the situations in the real world. As in our scheme, the time slot information is considered an essential problem to achieve the goal. Each data owner can define the time slot on their own demand. Furthermore, we take the blockchain-enabled data sharing environment into consideration and further combine the collusion resistance ability with our scheme.

On the other hand, disclosure of private keys increases the security risks of data such as PHRs (Personal Health Record) or even the information of the COVID-19 pandemic [21]. Liu et al.[6] established a patient-centric framework in the multiauthority model and used signcryption to guarantee data security. Besides, Li et al. concentrate on scalability in access control schemes. In their scheme, users in the PHR system were divided into various security domains and different policies would be published to different domains according to the definition of PHR owners. ABE as cryptographic primitives were applied and the rules of encryption and key-distributed were also based on those primitives. What's more, they use a hash chain to ensure forward security. However, the work in [6] just

applied to the PHR environment and may lose their varieties of other scenarios.

Revoking is also an important part of attribute-based encryption in blockchain-enabled data sharing systems, as authorities must keep the data consistency of each user. The first Hybrid Revocable ABE scheme is proposed by Attrapadung and Imai [22], which allowed data owners to choose how to revoke an attribute online: direct revocation or an indirect one. Thus, the scheme can take both advantages of direct and indirect revocation and avoid the disadvantages. Other schemes like [23], proposed by Sahai et al., solved the problem of attribute dynamic updating by proposing an attribute delegation method. Furthermore, they use a segmented secret key to ensure attributes are granted or revoke that even under a more restrictive access policy. But the backward security cannot be assured because the scheme needs to reencrypt the ciphertext so that when a new user comes to join the system, with the later time slot, he or she can still decrypt the data. Yang and Jia [13] try to solve the key escrow problem by putting forward a novel CP-ABE scheme in which a two-party computation protocol was executed between Key Generation Center (KGC) and Data Storage Center (DSC). In the above schemes, we could see that attribute revocation requests were mostly demanded by attribute authorities rather than users, or to say revoked users might not want to request for revocation for many reasons.

3. System Architecture

In this section, we first introduce the related preliminary knowledge, then present the system model of our scheme, and introduce the proposed access control scheme. At last, we give the security model. For convenience, some notations are summarized in Table 1.

3.1. Preliminaries

3.1.1. Bilinear Maps. There exist two multiplicative cyclic groups G and G_T with prime order p and generator g; e: $G \times G \longrightarrow G_T$ is a bilinear map if and only if the following three properties are satisfied:

(1) Bilinearity: if $\forall u, v \in G$ and $x, y \in Z_p$, then we have $e(u^x, v^y) = e(u, v)^{xy}$

(2) Nondegeneracy:$e(g, g) \neq 1$

(3) Computability: $\forall u, v \in G$, $e(u, v)$ is an admissible algorithm

3.1.2. Collusion Resistance. A collusion attack [24] in ABE means two or more entities (users, cloud servers, or even authorities) can successfully decrypt the data they can not decrypt individually after some operations like exchange their secret key or share information with each other. However, collusions using time restriction have not been mentioned before. For example, Alice has just been revoked by the data owner within the current time slot t_1 with the attribute which satisfied the access policy, while Bob is a curious cloud server provider that stores the ciphertext. It is obvious that both of them cannot decrypt the ciphertext individually. However, if they are working together with each other, Alice can send the data access request at the end of t_1 to Bob, while Bob needs to update the ciphertext at the beginning of the next time slot t_2. However, Bob can respond to the request of Alice using the ciphertext in t_1 while shifting the blame to network latency. As a result, Alice can get the decrypt the ciphertext in the time slot she should not have access right in. This kind of collusion can be easily implemented as it is easy to predict the attribute revoke time for Alice. Our scheme must have the ability to resist this kind of collusion attack in any circumstances.

Table 1: List of notations.

$\mathcal{T}$	Time slot
D	Attribute index
U_d	Attribute set authorized by AA_d
U	Universal attribute set, s
CT	Ciphertext
gid	Global identity
λ	Global parameters
S_{gid}	Set of user's attributes s
$S_{gid,t}$	Attribute set of global id gid at time t
$SK_{gid,x}$	Secret key of attribute x for user with gid
ω	A set of random number $\in Z_p$
DEK	Decryption key

3.1.3. Time Slot. A time slot is a particular time period defined by the data owner. The length of a time slot can be a day, an hour, or even a minute. However, it is not achievable for a cloud server to update every ciphertext with the newest time slot because the overhead of updates increases exponentially. So, it is a good choice for the cloud to update the ciphertext when they get the access request. On the other side, the data owner can define the attribute of a user across different time slots. For example, the time slot defined by the owner is an hour while the attribute of Alice is granted and revoked at 9 : 30 and 11 : 30, respectively. For a clear explanation, we call a time slot a decryptable time slot if and only if a user has the validity completely covered and the time slot can decrypt the ciphertext.

3.2. System Structure. We build our blockchain-enabled data sharing system with time restrictions as follows. As we cannot predict the changing trend of user's attributes, we divide time into time slots to separate the operation of attributes. We define time slot as $\mathcal{T} = \{n \mid n \in 1, 2, \ldots, N\}$. As shown in Figure 1, the system model is constituted by four types of entities: cloud server providers (servers in the cloud), attribute authorities (AAs), data owners (owners), and data users (users).

Cloud servers play the role of data storing and executing computation steps in policy updating, users/attributes revoking part. Normally, we consider that cloud servers are curious but honest, which means that cloud server providers will give their best to get the data stored in the cloud as a prerequisite of doing what data owners want correctly. As in

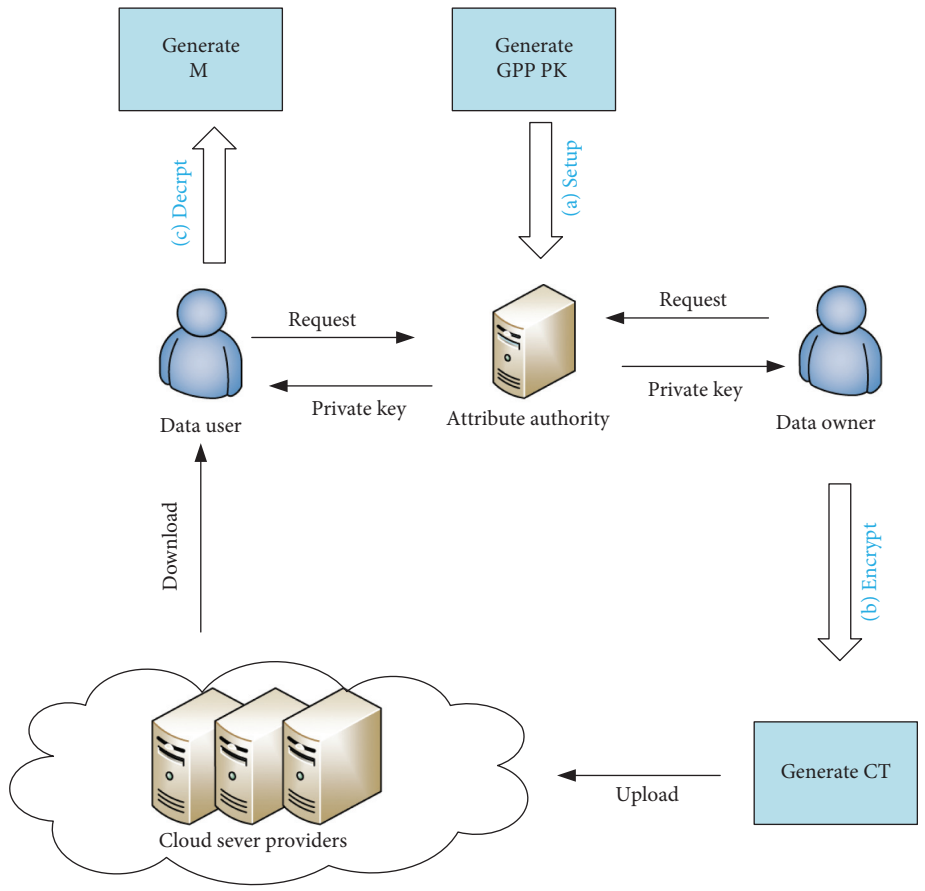

Figure 1: System model.

our model, we consider the situation that cloud servers may delay the update/revoke computation part for a short period of time to provide convenience for revoked users to gain data illegally. We derived this kind of situation into collusion between the cloud server and revoked users.

For attribute authority (AA for short), each of them is independent and responsible for granting or revoking attributes of users according to their roles or identities in their own domains. In our model, we consider that each attribute is only associated with a single AA (which may suit most of the situations in reality). However, each AA is in charge of a different number of attributes. That is to say, an attribute can only be authorized by one authority. We will identify the attributes by the index of the authorities below. For each authority, we use $\phi: U \longrightarrow D$ to map all attributes which belong to the AA to an identifier of the authority. AA can control the attributes or the structure in its domain. This kind of authorizing is required as the attributes changed periodically. As for time slots, it is not necessary to keep the length of every time slot the same for the reason that there may be different demands on time restrictions. In practice, this kind of requirement may largely reduce the computation cost on both authorities and data owners.

The data owner makes the definition of access policy before the data encryption on his side. Besides, he also makes a time span to set a time slot first. In the updating phase, the owner can update the slot or a new tree of time slot changing (usually not necessary). The encryption part on owners can be fast and light-weighted as owners only need to encrypt the data with the access policy designed by themselves. We define the ciphertext as CT_{A,t_e}. Besides, only those users who have the attributes satisfying the access control policy in the time slot t_e ($e \in \mathcal{T}$) can decrypt the data in CT_{A,t_e}.

In our system, when a user with gid gains a new attribute x, a new secret key $\mathrm{SK}_{\mathrm{gid},x}$ will be granted at the same time by corresponding AA. If the user wants to decrypt the data, he has to obtain the update keys first at this time slot (i.e., $\mathrm{UK}_{x,t}$) from the authority who can publish the attribute. After that, the user can compute decryption keys for a time slot t based on his secret keys and further uses them to decrypt the ciphertext. In this case, we can guarantee that users can only get the data in a single time slot because those users who do not update their attributes with the time slot in the past cannot satisfy any access control policies in our system. Considering the time consistency, we use the time slots we mentioned above to ensure that all entities in our model can check their current time with time slot at any second they want.

3.3. Security Model. In our scheme, we take these points into consideration: (1) The cloud server is curious about the ciphertext stored in the cloud, and they will try their best to decrypt them. (2) Cloud servers may send the data (in the

form of ciphertext) to unauthorized users. (3) Users and cloud servers may collude with each other. The security model is run between a challenger and an adversary $\mathcal{A}$, which is defined by the following game with two phases.

Setup. (1) The challenger first runs *GlobalSetup* algorithm and opens the access of GPP to the adversary. (2) $\mathcal{A}$ randomly select several AA*s* to play the role of corrupted AA and ask them to send their public key PK_d to $\mathcal{A}$.

Phase 1. $\mathcal{A}$ can request secret keys and update keys only by repeating the following steps:

(1) SK Query (gid, x): $\mathcal{A}$ sends a secret key request to those uncorrupted authorities by submitting a tuple (gid, x) where gid is the unique global identifier of a user and x is an attribute which is authorized by one of the uncorrupted authority. After receiving the queries, the challenger runs **SKeyGen** algorithm to return the corresponding secret keys $\mathrm{SK}_{\mathrm{gid},x}$ to $\mathcal{A}$.

(2) UKQuery (t, x): at the beginning of a time slot, $\mathcal{A}$ can ask those uncorrupted authorities to update their attributes or time slot update (if needed) and submit the pair of (t, x) to be updated by authorities. The challenger returns an update key $\mathrm{UK}_{x,t}$ to $\mathcal{A}$.

Challenge Phase. $\mathcal{A}$ submits two equal length messages M_0 and M_1, an access policy A^* (all attributes in A^* belongs to U), a time slot $t^* \in \mathcal{T}$ to the challenger. After this, the adversary should give the public key PK_d of all corrupted authorities whose attributes appear to the challenger. Then, the challenger flips a coin $\beta \in \{0, 1\}$ and sends to $\mathcal{A}$ the encrypted M_β using (A^*, t^*).

Phase 2. $\mathcal{A}$ can make as many queries as he wants according to Phase 1.

Guess. $\mathcal{A}$ submits a guess β' for β. The adversary $\mathcal{A}$ will win the game if $\beta\prime = \beta$ and satisfies the following demand.

(1) UKQ $(t, *)$ can only be queried on time slot after the time of all requests above, which means that the past time slot period of time in the system cannot be traced. Also, for any pair (t, x), UKQ phase can be executed only once because the corresponding authority will not publish the update key after the beginning of the time slot, which means the initialize phase in each slot is run by AA executes only once.

(2) For any queried gid, $S_{\mathrm{gid},t}$ ($S_{\mathrm{gid},t}$ stands for the set of gid and t) does not satisfy A^*. The advantage of $\mathcal{A}$ is defined as $|\Pr[\beta = \beta'] - 1/2|$.

4. Time Slot Access Control Scheme

In this section, we will explain our scheme step by step and list some algorithms if needed. Based on the algorithms defined in Section 3. Our scheme contains the four main phases: System Initialization which runs at the beginning of the whole system, Key Generation ran by each AA, Data Encryption phase for data owners to encrypt data with defined access policy, and Data Decryption ran by Users and computation outsourcing party. The workflow of our scheme is listed in Figure 2.

4.1. System Overview. Phase 1: system initialization: The system initialization phase is run at the beginning of the system and has two steps: Global Setup and Authority Setup.

(1) Global Setup:

$$\mathrm{GlobalSetup}(\lambda) \longrightarrow \mathrm{GPP}$$

The input of the global setup phase is the security parameters and the output is the Global public parameters GPP which will be used in other phases later. Set G and $G_{\mathcal{T}}$ as a bilinear group of prime order p with the bilinear group G which has the generator g. The global public parameter GPP used for key generation is published as $\mathrm{GPP} = (e, H, g, p)$; here e is a bilinear paring, and H is a hash function that maps every gid to elements of the group G.

(2) Authority Setup:

$$\mathrm{AuthoritySetup}(\mathrm{GPP}, U_d) \longrightarrow (\mathrm{PK}_d, \mathrm{MSK}_d)$$

Each AA must run a setup algorithm before publishing authorities. It takes the inputs as the global public parameters GPP , which outputs in *GlobalSetup* phase, the attribute domain U_d of the authority itself. The output of this phase is the master secret key MSK_d which is used for the authority itself and the public key PK_d , which sends to users. For any attribute x belongs to the attribute universe, the algorithm chooses the random exponents $\alpha_x, \beta_x, \gamma_x \in Z_p$. Besides, the algorithm also chooses a random element for the pseudorandom function F as the seed to generate the function. For each attribute that can be published by authorities, it chooses a random number $R = F(\tau_x, a) \in G$ (a_x denotes the attributes set of the authorities) and uses it to generate a secret key for the user. Here, we denote $U_d \times \mathcal{T} \longrightarrow G$ to be a hash function that maps both the attributes of the authorities and time slots in $U_d \times \mathcal{T}$ to elements of G. Then, the public key can be generated as $\mathrm{PK}_d = (\{e(g, g)^{\alpha_x}, g^{\beta_x}, g^{(1/\gamma_x)}\}_{x \in U_d}, H_d)$, where $\alpha_x, \beta_x, \gamma_x$ are combined to build the master secret key which is only kept by the authorities themselves.

So after the initialization phase, we get the global parameters GPP, public keys PK_d , and secret keys MSK_d generated by every authority. Each authority will further use these keys to generate a secret key for those users in their attribute domain.

Phase 2: key generation by AA.

$$\mathrm{SKeyGen}(\mathrm{gid}, x, \mathrm{GPP}, \mathrm{MSK}_{\phi(x)}) \longrightarrow (\mathrm{SK}_{\mathrm{gid},x})$$

Every AA runs the key generation algorithm for users in its domain. Each AA takes the global public parameters GPP, the master secret key $\mathrm{MSK}_{\phi(x)}$ generated on the last phase along with user's global identity gid as input and outputs the secret key for corresponding user as $\mathrm{SK}_{\mathrm{gid},x}$. The key generation algorithm is run by AA, and outputs the secret key $\mathrm{SK}_{\mathrm{gid},x}$, which associates with users' global identity and the corresponding attribute. The algorithm has two steps:

(1) Sets $u_{x,\mathrm{gid}} = 2^{h_x} + \mathrm{count}_x$ and adds the pair $(\mathrm{gid}, u_{x,\mathrm{gid}})$ to a List_x (a list of attribute trees for the

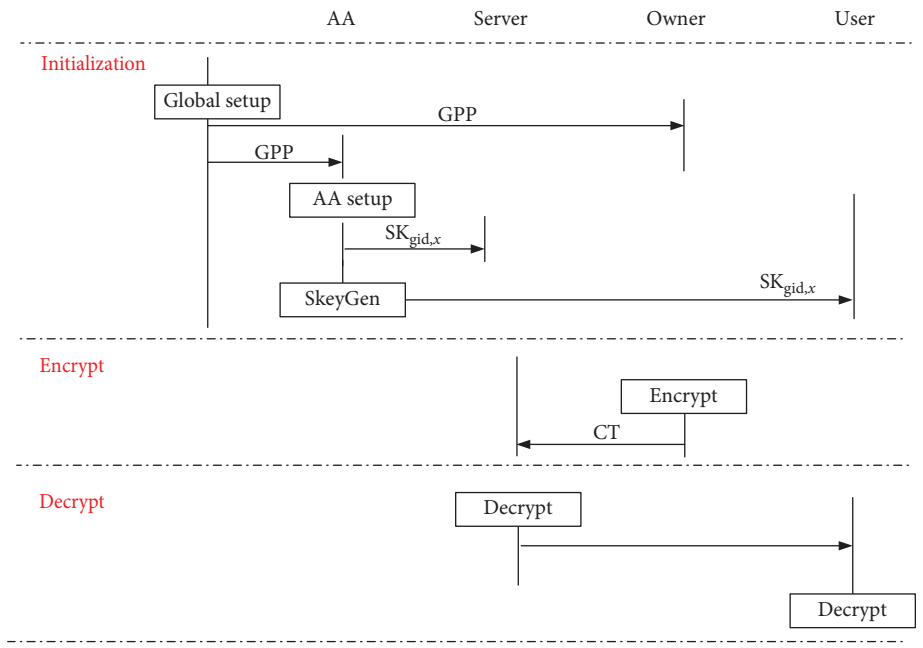

Figure 2: Workflow.

user x with the time restriction) and sets $\text{count}_x = \text{count}_x + 1$.

(2) The authority $\text{AA}_{\phi(x)}$ sends the secret key $\text{SK}_{\text{gid},x}$ to the user. We have to emphasize that every secret key for the attribute is only generated after the attribute is established for the sake of privacy preserving for user information.

Phase 3: data encryption:

$\text{Encrypt}(M, t_e, A, \text{GPP}, \{\text{PK}_d\}) \longrightarrow \text{CT}$

The encryption phase includes the input part of the Message M, access control policy A which is based on attributes may from several authorities, the time slot t_e, the global public parameters GPP, and the public keys $\{\text{PK}_d\}$. The output of this phase is the ciphertext CT which contains the access policy A.

Considering the difference in file type and scale, owners first encrypt their files by using a symmetric encryption algorithm and use our data encryption method to encrypt the secret key of the symmetric encryption. In this way, we could shorten the encrypt message length to reduce the computation cost on the owner side. In this Phase, M consists of two parts: one is the symmetric encryption key K and the other is the ciphertext encrypted by K. A is the access control policy and $\{\text{PK}_d\}$ are the set of public keys generated by the authorities; these public keys are related to the access policy. The data owner chooses a random number $r \in Z_p$ as the secret and keeps it without anyone else knowing it. After this, the owner chooses another two random vectors $\vec{v}$, $\vec{u}$ $\in\in Z_p$, and for each index $i \in \{1, 2, \ldots, m\}$, it selects another random oracle $r' \in Z_p$. So, the ciphertext is computed as follows:

$$\begin{aligned}\text{CT} = \Big\{\Big\{t_e, (A,\rho), C = M \cdot e(g,g)^e, C_{i,1} = e(g,g)^{\lambda_i} e(g,g)^{\alpha_{\rho(i)} r_i},\\ C_{i,2} = g^{\mu_i} g^{\beta_{\rho(i)} r_i}, C_{i,3} = g^{r_i}, C'_{i,3} = \left(g^{(1/r_x)}\right)^{r_i}, C_{i,4} = H_{\phi(\rho(i))}(\rho(i), t_e)^{r_i}\Big\}_{i=1}^{m}\Big\}.\end{aligned} \tag{1}$$

Then, the owner stores the ciphertext in cloud servers for later data sharing.

Phase 4: data decryption by users:

$\text{Decrypt}(\text{CT}, \text{GPP}, \{\text{PK}_d\}, \{\text{DK}_{\text{gid},x,t}\}_{x \in S \in \text{gid},t}) \longrightarrow (M) \text{or} \perp$

The decryption algorithm can run by users or outsourcing party which depends on user's choices. When users want to decrypt by themselves, the input includes the ciphertext CT along with the access policy A, the global public parameters GPP, the public keys PK_D and decryption keys as $\{\text{DK}_{\text{gid},x,t}\}_{x \in S \in \text{gid},t}$ which is related to the user's gid and current time slot. The algorithm outputs the plaintext M when the decryption succeeds or a token $\perp$ implying decryption fails for some reason. All users with proper attributes can download the ciphertext in which their attributes meet the demand of the access policy at the specific time slot t_e. The decryption phase must compute the decryption key first and then decrypt the ciphertext.

4.1.1. Decryption Key Computation. At the beginning of the decryption, the user with gid first needs to compute the decryption key for the current time slot as follows: $\mathrm{DecryptKey}(\mathrm{SK}_{\mathrm{gid},x}, t_e) \longrightarrow (\mathrm{DK}_{\mathrm{gid},x,t})$ or $\perp$. In this phase, the user first checks the attribute set and whether the corresponding attribute x as $x \in S_{\mathrm{gid},t}$ exists. If so, the decryption key can be computed as follows:

$$\mathrm{DK}_{\mathrm{gid},x,t} = \left(D_{\mathrm{gid},x,t} = K_{\mathrm{gid},x,v_x}, D'_{\mathrm{gid},x,t} = \left(E_{v_x}\right)^{K'_{\mathrm{gid},x,v_x}} \cdot K''_{\mathrm{gid},x,v_x}, D''_{\mathrm{gid},x,t} = \left(E'_{v_x}\right)^{K'_{\mathrm{gid},x,v_x}} \right). \tag{2}$$

It is worth noting that if a user has the right attributes with another time slot t' later than t_e they cannot compute the decryption key either. This feature has great resistance to the collusion attacks we mentioned before.

4.1.2. Ciphertext Decryption. After generating the decryption key with adequatea attributes and time slot, the decryption algorithm goes as follows: $\mathrm{Decrypt}(\mathrm{CT}, \mathrm{GPP}, \{\mathrm{PK}_d\}, \{\mathrm{DK}_{\mathrm{gid},x,t}\}_{x \in S_{\mathrm{gid},t}}) \longrightarrow (M)$ or $\perp$. For any $t \neq t_e$ or $S_{\mathrm{gid},t}$ that cannot satisfy (A, ρ), the algorithm outputs $\perp$. For those users who can be satisfied with conditions above, the decrypt algorithm goes in two steps. The first step is to find a $I = \{i \mid \rho(i) \in S_{\mathrm{gid},t_e}\}$ and compute the corresponding constants $\{\omega_i \mid i \in I\}$. For I, compute the following:

$$\begin{aligned}
\overline{C}_i &= \frac{C_{i,1} \cdot e\left(H(\mathrm{gid}), C_{i,2}\right)}{e\left(D_{\mathrm{gid},\rho(i),t_e}, C_{i,3}\right)} \cdot \frac{e\left(D'_{\mathrm{gid},\rho(i),t_e}, C'_{i,3}\right)}{e\left(D''_{\mathrm{gid},\rho(i),t_e}, C'_{i,4}\right)} \\
&= \frac{e(g,g)^{\lambda_i} e(g,g)^{\alpha_{\rho(i)} r_i} \cdot e\left(H(\mathrm{gid}), g^{\mu_i} g^{\beta_{\rho(i)} r_i}\right)}{e\left(g^{\alpha_{\rho(i)}} H(\mathrm{gid})^{\beta_{\rho(i)}}, g^{r_i}\right)} \cdot \frac{e\left(\left(R_{v_{\rho(i)}}\right)^{\alpha_{\rho(i)} \gamma_{\rho(i)} r_{\mathrm{gid},\rho(t),v_{\rho(i)}}}, g^{\left(r_i/\gamma_{\rho(i)}\right)}\right)}{e\left(\left(R_{v_{\rho(i)}}\right)^{\alpha_{\rho(i)} \gamma_{\rho(i)} r_{\mathrm{gid},\rho(t),v_{\rho(i)}}}, g^{r_i}\right)} \cdot \frac{e\left(H_{\phi(\rho(i))}(\rho(i), t_e)^{\varsigma_{v_{\rho(i)}, t} r_{\mathrm{gid},\rho(i),v_{\rho(i)}}}, g^{\left(r_i/\gamma_{\rho(i)}\right)}\right)}{e\left(g^{\left(\varsigma_{v_{\rho(i)}, t} r_{\mathrm{gid},\rho(i),v_{\rho(i)}}/\gamma_{\rho(i)}\right)}, H_{\phi(\rho(i))}(\rho(i), t_e)^{r_i}\right)} \\
&\quad \cdot \frac{e\left(\left(R_{v_{\rho(i)}}\right)^{\gamma_{\rho(i)} r_{\mathrm{gid}\, t, \rho(i), v_{\rho(i)}}}, g^{\left(r_i/\gamma_{\rho(i)}\right)}\right)}{e\left(\left(R_{v_{\rho(i)}}\right)^{r_{\mathrm{gid}\, t, \rho(i), v_{\rho(i)}}}, g^{r_i}\right)} \\
&= e(g,g)^{\lambda_i} \cdot e(H(\mathrm{gid}), g)^{\mu_i}.
\end{aligned} \tag{3}$$

And, the last step computes the following:

$$\begin{aligned}
\prod_{i \in I} \overline{C}_i^{\omega_i} &= e(g,g)^{\sum_{i \in I} \omega_{i\lambda_i}} \cdot e(H(\mathrm{gid}), g)^{\sum_{i \in I} \omega_i \mu_i} \\
&= e(g,g)^r.
\end{aligned} \tag{4}$$

Then, users can recover the symmetric encryption secret key by $K = C / e(g,g)^r$. After that, users can get the message by doing a symmetric decryption in which the computation cost is negligible.

If the user wants to outsource the decryption part to other computation devices, the first thing to do is to generate outsourced key and recover the key for decryption. The user generates the outsourced key as follows:

$$\mathrm{OK}_{\mathrm{gid},x,t} = \left(O_{\mathrm{gid},x,t} = K_{\mathrm{gid},x,v_x}, O'_{\mathrm{gid},x,t} = \left(E_{v_x}\right)^{K'^{(1/\sigma)}_{\mathrm{gid},x,v_x}} \cdot K''_{\mathrm{gid},x,v_x}, O''_{\mathrm{gid},x,t} = \left(E'_{v_x}\right)^{K'^{(1/\sigma)}_{\mathrm{gid},x,v_x}} \right). \tag{5}$$

Here, $\sigma \in Z^*$ is the recover key RK selected by the data user.

The second step is run by CSP or computing devices (they might be a node on the blockchain). After receiving the

OK from the user along with the ciphertext, they will partially decrypt the ciphertext as follows:

$$\overline{C}_i' = \frac{C_{i,1} \cdot e\left(H(\text{gid}), C_{i,2}\right)}{e\left(O_{\text{gid},\rho(i),t_e}, C_{i,3}\right)} \cdot \frac{e\left(O'_{\text{gid},\rho(i),t_e}, C'_{i,3}\right)}{e\left(O''_{\text{gid},\rho(i),t_e}, C_{i,4}\right)} = e(g,g)^{(\lambda_i/\sigma)} \cdot e(H(\text{gid}), g)^{(\mu_i/\sigma)}. \tag{6}$$

After that, they calculate CT′.

$$\text{CT}' = \prod_{i \in I} \overline{C}_i'^{\omega_i} = e(g,g)^{\sum_{i \in I}(\omega_i \lambda_i/\sigma)} \cdot e(H(\text{gid}), g)^{\sum_{i \in I}(\omega_i \mu_i/\sigma)} = e(g,g)^{(r/\sigma)}. \tag{7}$$

The last phase is executed on the user side. After the user gets CT′ from outsourcing entities, the user can use RK to retrieve the plaintext:

$$\begin{aligned} \text{De} &= \text{CT}\prime e(g,g)^{-(1/\sigma)} \\ &= \left(e(g,g)^{-(1/\sigma)}\right) e(g,g)^{-(1/\sigma)} \\ &= e(g,g)^{r}. \end{aligned} \tag{8}$$

4.2. Updating With Time Restriction. As we mentioned before, when a user tries to revoke an attribute or applies for a new attribute from AA, the secret keys associated with his attributes should be updated either. While in our scheme, this problem can be replaced by the changing of time slot. $\text{AA}_{\phi(x)}$ needs to run the update key algorithm when receiving the update request from data owners. The algorithm takes the input as the current time slot t, updates attribute $x \in U_{\phi(x)}$, and outputs the update key $\text{UK}_{x,t}$.

For example, when a data owner U_d tries to revoke an attribute x_i , he must send an updates request to the $\text{AA}_{\phi(x)}$ which is in charge of x_i. Then, $\text{AA}_{\phi(x)}$ selects a random set $\varpi\{\varpi_1, \varpi_2, \ldots, \varpi_k\} \in Z_p$ $(\sum_{n=1}^{k} \varpi_k = 0)$ where the numbers in the random set are equal to the numbers of the attribute authorities he wants to send to. After this phase, $\text{AA}_{\phi(x)}$ updates the attribute key component $\text{ASK}_{K,n}$ for the user U_d with the attribute x_i as $\text{ASK}'_{K,n} = D_i = g^{\gamma_n} H(i)^{\gamma_i}$, $D'_i = (g^{\gamma_i})^{\varpi_k}$.

By using updating algorithm, cloud servers can update the ciphertext without getting any sensitive information from the data owner. Meanwhile, only the cloud server knows about the s'. In this case, the new kind of collusion attacks between cloud and revoked users can be easily traced.

5. Security Analysis and Performance Evaluation

5.1. Security Analysis. As mentioned in the previous sections, in our scheme, the main difference between our scheme and Lewko and Waters scheme [25] is we embedded time slots into both ciphertexts and keys. So, the security of our scheme lies in the below attacks:

(1) Outsourcing entities try to infer information about the ciphertext and may compare the ciphertext to find differences in diverse data owners.

As the outsourcing entities undertake massive predecryption tasks from different users, they may collect different ciphertext. However, they cannot infer any information only from those ciphertexts as each CT has different t_e, access policy and bilinear paring.

(2) Users try to predict the data info from the ciphertext, which they cannot decrypt.

This attack assumption does not hold either because first users cannot get any reason about their failure on decryption but a symbol $\perp$.

Despite these attacks, our scheme is similar to the scheme in [25]. Thus, our scheme is secure in the bilinear group model, which is the same as the proof used in [25]. In the generic bilinear group model and random oracle model, there is no adversary that can break our scheme in polynomial time with a nonnegligible advantage in the security game we mentioned before. Moreover, we will make an analysis of our scheme from the other three parts: collusion resistance, data confidentially, and attributes revoke.

5.1.1. Collusion Resistance. There may exist several kinds of collude operations in our model and we will analyze them one by one. The first one is collusion between users. For those users whose attributes cannot meet the demand of access control policy, it is a common way to combine their secret keys with each other to get a new key that can decrypt the ciphertext. However, since the prime order of their secret key is randomly chosen by authorities, respectively, no matter what kind of attributes set they ever had, they cannot get a proper key in any case.

Another case is the collusion between an unauthorized user with a revoked user with gid_1. Users with gid_1 may want the secret key gid_2 once had to get a combination of attributes with their own gid_1. As mentioned before, this kind of collusion can not exist either because of the random oracle.

The last situation is as we listed in the introduction. For the traditional method, cloud server providers can forge the execution time of updating algorithm or even simply pull back the updating time and pass the buck to serious network delay. Users can select plural servers with their data stored and any one of the servers delay his updating does not make sense to those malicious revoked users. Because that the data owner can select the servers as his wish, it is nearly impossible for a revoked user to make a deal with all servers in the domain. From this point, we could say that this kind of collusion can barely exist.

5.1.2. Data Confidentiality. As we mentioned before, not all the channels are secure in our model. But for the data transmission part, all of the data are transmitted in ciphertext, so we only consider the situation below: the cloud

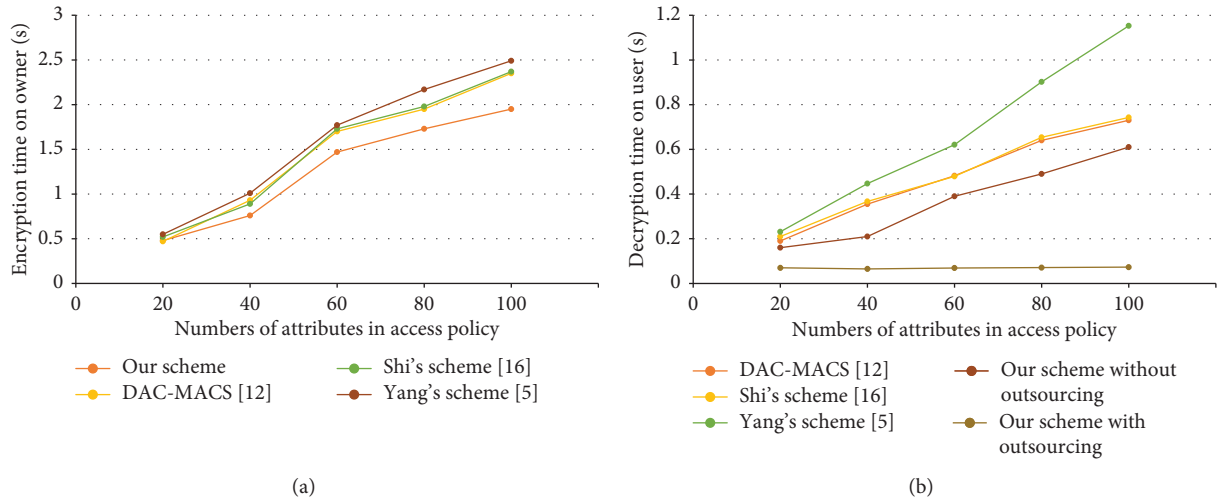

FIGURE 3: Encryption and decryption time cost with a different number of attributes. (a) Encryption time on the owner side. (b) Decryption time on the user side

server or any user who cannot access the data may get the tuple (Cipher, CT) at any time slot (here Cipher stands for the text transformed by plaintext). As the owner encrypts the plaintext such as symmetric encryption first, the Cipher does not leak any information about the data. On the other hand, CT can be decrypted only by those users with appropriate attributes, so attackers can not find any relations between Cipher and CT. Furthermore, any user who wants to recover the time slot or attributes in CT is not possible for the same reason.

5.1.3. Revocation. This part is slightly similar to the collusion resistance part as we mentioned before, and here we only consider the forward security and backward security of our scheme. Due to the fact that the time slot is running continuously, when a new user joins the system and is granted attributes from the authorities, the time slot match with his keys cannot be the period before his join. Thus, forward security can be ensured in this way. Similar to the forward security, when AA wants to revoke attributes of a user, the ciphertext cannot be decrypted by this user as the ciphertext is updated with a new time slot and the user cannot get a new key both with the revoked attributes and the new time slot.

5.2. Performance Evaluation

5.2.1. Experimental Setup. The simulation platform for our scheme is Ubuntu 14.04 with an Intel Core (TM) i5-5600U at 2.6 GHz and 4 GB RAM. The simulation environment is JPBC (Java Pairing-Based Cryptography library ver2.0.0) with 160-bit group order, 512-bit field size.

5.2.2. Algorithm Analysis. In our experiments, we did 10000 trials to limit the error range. We first compare the computation cost on owners. As we can see in Figure 3, the computation time takes about 10% less than Hur's scheme and the DAC-MACS. While in the decryption part, we can see that the computation time cost is much less. In real-world scenarios, it is obvious that the operation of downloading is much more than updating because data users in the cloud environment are huge. Meanwhile, as in a cloud-based system, it is more important to save the computation cost on the user side as mobile devices are widely used nowadays.

5.2.3. Attribute Impact. In this section, we try to simulate the impact on attribute numbers on both the encryption side and the decryption side. We set the 10 AA with different attributes from 1 to 20. Specifically, the cost on the decryption side (with both outsourcing and self-decryption) and the computation on encryption are evaluated in Figure 3 to show the impact on the attribute universe more precisely. The experiment shows that both encryption and decryption costs grow steadily with the rise of attributes number. Outsourced decryption time on the user side almost stays the same as no matter how many attributes are in the policy, the user only needs to compute the paring algorithm for once. Moreover, we also take the key generation time into consideration. As we can see from Figure 4, our attribute authority private key generation time is still between the DAC-MACS case and Shi's scheme, and the attribute authority private key generation time is proportional to the number of attributes.

5.2.4. Attribute Update Evaluation. As lots of the scheme only takes policy updates into consideration, we only compared our attribute updating algorithm with DAC-MACS, which have a similar part to ours. With the increase of the attributes, owners need more time to generate the update key while the time spent on servers almost stays the same because the major part of the computation task has been done on the cloud side. For most situations in reality, the owner can afford the computation cost.

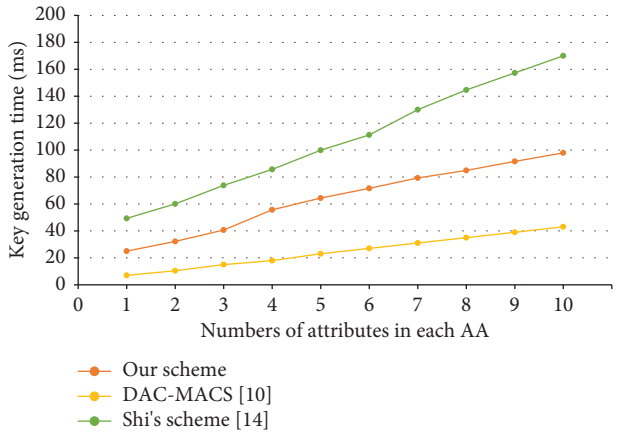

Figure 4: Key generation cost with different numbers of attributes in AA.

From the above aspects, both theoretical analysis and experiments results show that our scheme can provide time slot access control and attributes update at the cost of a slight increase of key generation phase. However, as the key is generated on AA, it barely has any effect since AA normally has enough computation power. Moreover, comparing with the existing schemes, our blockchain-enabled data sharing scheme has a high level of security of collusion resistance.

6. Discussion and Conclusion

In this paper, we have proposed a collusion-resistant CP-ABE blockchain-enabled data sharing scheme to achieve access control under a time restriction. Specifically, we have proposed this scheme using time slots as a token to bind with ciphertexts and keys to make sure that only the user with demanded attributes and in the particular time slot can decrypt the data. Besides, we considered a new kind of collusion, which might be common in our daily life that curious cloud servers may delay the policy update/attribute revoke algorithm for a short time to let the revoked user get data illegally. This kind of collusion is hard to trace because cloud servers can easily shift their responsibility to other reasons like network latency and so on. Furthermore, we used time slots to ensure data security while the decryption phase is outsourced. Further discussion on our schemes is about security issues on revocation and collusion resistance. In the future, we will keep on implementing our scheme and exploring the access control structures with a time restriction and taking time into consideration in the case of collusion between the revoked user and one of the authorities.

Data Availability

The data used to support the findings of this study are available from the authors upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by the National Science Foundation of China (nos. 61772385 and 61572370.)

References

[1] Z. Tian, C. Luo, H. Lu, S. Su, Y. Sun, and M. Zhang, "User and entity behavior analysis under urban big data," *ACM/IMS Transactions on Data Science*, vol. 1, no. 3, 2020.

[2] Z. Tian, C. Luo, J. Qiu, X. Du, and M. Guizani, "A distributed deep learning system for web attack detection on edge devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1963–1971, 2020.

[3] A. Sahai and B. Waters, "Fuzzy identity-based encryption," *Lecture Notes in Computer Science*, vol. 3494, pp. 457–473, 2005.

[4] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 89–98, Alexandria, VA, USA, October 2006.

[5] K. Yang, Z. Liu, X. Jia, and X. S. Shen, "Time-domain attribute-based access control for cloud-based video content sharing: a cryptographic approach," *IEEE Transactions on Multimedia*, vol. 18, no. 5, pp. 940–950, 2016.

[6] J. Liu, X. Huang, and J. K. Liu, "Secure sharing of personal health records in cloud computing: ciphertext-policy attribute-based signcryption," *Future Generation Computer Systems*, vol. 52, pp. 67–76, 2015.

[7] L. I. Youhuizhi, Z. Dong, K. Sha, C. F. Jiang, J. Wan, and Y. Wang, "TMO: time domain outsourcing attribute-based encryption scheme for data acquisition in edge computing," *IEEE Access*, vol. 7, Article ID 40240, 2019.

[8] B. Q. Baodong, R. H. Deng, S. L. Shengli, and S. M. Siqi, "Attribute-based encryption with efficient verifiable outsourced decryption," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1384–1393, 2015.

[9] L. Zhang, C. Xu, Y. Gao, Y. Han, X. Du, and Z. Tian, "Improved Dota2 lineup recommendation model based on a bidirectional LSTM," *Tsinghua Science and Technology*, vol. 25, no. 6, pp. 712–720, 2020.

[10] Z. Gu, W. Hu, C. Zhang, H. Lu, L. Yin, and L. Wang, "Gradient shielding: towards understanding vulnerability of deep neural networks," *IEEE transactions on network science and engineering*, vol. 8, no. 2, 2020.

[11] Z. A. Lei, B. St, and C. Ll, "An finite iterative algorithm for sloving periodic Sylvester bimatrix equations," *Journal of the Franklin Institute*, vol. 357, no. 15, Article ID 10757, 2020.

[12] K. Yang, X. Jia, K. Ren, B. Zhang, and R. Xie, "DAC-MACS: effective data access control for m cloud storage systems," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1790–1801, 2013.

[13] K. Yang and X. Jia, "Expressive, efficient, and revocable data access control for multiauthority cloud storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1735–1744, 2014.

[14] D. Wang, L. Zhang, C. Huang, and X. Shen, "A privacy-preserving trust management system based on blockchain for vehicular networks," in *Proceedings of the 2021 IEEE Wireless Communications and Networking Conference (WCNC)*, Nanjing, China, March 2021.

[15] K. Yang, X. Jia, and K. Ren, "Attribute-based fine-grained access control with efficient revocation in cloud storage

systems," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pp. 523–528, Hangzhou, China, May 2013.

[16] J. Shi, C. Huang, J. Wang, K. He, and J. Wang, "An access control scheme with direct cloud-aided attribute revocation using version key," in *Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing*, pp. 429–442, Dalian, China, August 2014.

[17] T. Kitagawa, H. Kojima, N. Attrapadung, and H. Imai, "Efficient and fully secure forward secure ciphertext-policy attribute-based encryption," *Lecture Notes in Computer Science*, vol. 7807, pp. 87–99, 2015.

[18] L. Zhang, Z. Huang, W. Liu, Z. Guo, and Z. Zhang, "Weather radar echo prediction method based on convolution neural network and Long Short-Term memory networks for sustainable e-agriculture," *Journal of Cleaner Production*, vol. 298, Article ID 126776, 2021.

[19] Q. Liu, G. Wang, and J. Wu, "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment," *Information Sciences*, vol. 258, pp. 355–370, 2014.

[20] J. Hong, K. Xue, Y. Xue et al., "TAFC: time and attribute factors combined access control for time-sensitive data in public cloud," *IEEE Transactions on Services Computing*, vol. 13, no. 1, 2017.

[21] Z. Gu, L. Wang, and X. Chen, "Epidemic risk assessment by A novel communication station based method," *IEEE Transactions On Network Science And Engineering*, 2021.

[22] N. Attrapadung and H. Imai, "Attribute-based encryption supporting direct/indirect revocation modes," in *Proceedings of the IMA International Conference on Cryptography and Coding*, pp. 278–300, Cirencester, UK, December 2009.

[23] A. Sahai, H. Seyalioglu, and B. Waters, "Dynamic cciphertext delegation for attribute-based encryption," *Lecture Notes in Computer Science*, vol. 7417, pp. 199–217, 2012.

[24] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of a Workshop on the Theory and Application of Cryptographic Techniques*, pp. 10–18, Linz, Austria, April 1985.

[25] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proceedings of the Advances in Cryptology - EUROCRYPT 2011*, pp. 568–588, Tallinn, Estonia, May 2011.