Hindawi
Security and Communication Networks
Volume 2023, Article ID 2822846, 21 pages
https://doi.org/10.1155/2023/2822846

Research Article

CP-ABE Optimization via the Flexible Integration of Access Policies Containing Multiple Shared Subpolicies

Yinlong Wang, Ting Guo, and Nurmamat Helil

College of Mathematics and System Science, Xinjiang University, Urumqi 830046, China

Correspondence should be addressed to Nurmamat Helil; nur924@sina.com

Received 27 August 2022; Revised 29 January 2023; Accepted 21 February 2023; Published 18 April 2023

Academic Editor: Jie Cui

Ciphertext-policy attribute-based encryption (CP-ABE) is a cryptographic scheme that is suitable for cloud storage and can realize secure data sharing. However, in most existing CP-ABE schemes, the encryption computational overhead is proportional to the number of attributes in the access policy. Therefore, reducing the computational overhead of the scheme's encryption is an important consideration to improve its efficiency. Furthermore, in the scenario where the access control policies corresponding to different data shared by the same data owner have multiple shared subpolicies, there is still further optimizing space for encryption. This study proposes an optimization approach for CP-ABE via the flexible integration of access policies with multiple shared subpolicies. Regarding the root nodes of policies being both the same and different, we provide optimal policy integration methods to reduce the overall encryption computation cost of the data associated with these access policies. Under the premise of preserving the original security of the CP-ABE scheme, this scheme avoids repeated calculation related to the shared subpolicies during encryption, thus improving the scheme's efficiency. Finally, the correctness and feasibility of the scheme are examined and verified by theoretical analysis and experiments.

1. Introduction

With the rapid development of cloud computing technology [1], more institutions and individuals have been inclined to outsource their data to the cloud, exploiting the advantages of cloud storage. The cloud can provide users with rapid and convenient data access services on demand. However, the confidentiality of outsourcing data is a key problem that needs to be solved. As cloud service providers are often honest but curious, they may become vulnerable to data confidentiality breaches. Encryption is the best way to achieve secure access control of data in these nonfully trusted cloud environments. As a new type of cryptography primitive, attribute-based encryption (ABE) [2] provides an ideal solution for secure data access control in cloud storage. As an extension of identity-based encryption (IBE) [3], in 2005, Sahai and Waters first proposed an identity encryption scheme based on biometric information called Fuzzy identity-based encryption (Fuzzy-IBE) [2]. Subsequently, Goyal et al. and Bethencourt et al. proposed key-policy attribute-based encryption (KP-ABE) [4] and ciphertext-policy attribute-based encryption (CP-ABE) [5], respectively. Comparing the two types of ABE, we find that CP-ABE is more suitable for the access control scenario on cloud storage.

On the premise of ensuring the security of outsourced data in cloud storage, we also need to reduce the computational and storage overhead and improve the efficiency of encryption and decryption. In most existing ABE schemes, the computational and storage overhead of encryption and decryption are related to the number of attributes in the policy. Therefore, reducing the computational and storage overhead is an important consideration to improve the scheme's efficiency.

Scholars have researched the reduction of the computation consumption of encryption and decryption, including [6–16]. These works of literature mainly reduce computational overhead by outsourcing encryption or decryption. By using the substantial computing power of the third party to complete most of the computing tasks, the user can complete

encryption or decryption only through simple calculation, reducing the user's computational overhead and improving the scheme's operational efficiency. In addition, there has also been research on how to improve the efficiency of CP-ABE schemes [17–21]. These efforts have mainly focused on how to improve the scheme's efficiency by reducing the storage overhead.

Scholars have also paid attention to improving the encryption/decryption efficiency of CP-ABE. Under the condition that the access policies of different data of a data owner are partly the same, in that when there are shared subpolicies, the data owner can reuse the intermediate calculation results related to the shared subpolicies during the encryption process. In these schemes in the literature [22, 23], multiple access trees of different data of the same user are integrated into one access tree and encrypted these data together. However, these schemes do not consider the scenario of multiple data with the same subpolicies. The literature [24–26] considers the situation where there are shared subpolicies among access control policies of different data. Among them, the access structure of the literature [24, 25] is the access tree, and that of the literature [26] is linear secret sharing scheme (LSSS) matrix. On the basis of the literature [24], the literature [25] adds decryption outsourcing to further reduce the user's decryption overhead. However, the literature [24, 25] only deals with the situation where there is only one shared subpolicy among different access policies. A more flexible policy integration method is needed to cope with more complicated cases of multiple shared subpolicies. Fewer restrictions should be imposed on the number of shared subpolicies and the threshold of the root node of access trees.

Let us consider an example. Suppose that an engineering department of a company has two projects and three files related to those projects that need to be shared with the company employees. M_1 (Project 1 quality report), M_2 (Project 2 quality report), and M_3 (Integration plan of two projects), and their access policies are shown in Figure 1: $\mathcal{T}_1$: $\{P_1$ AND {ED AND QE} AND {PL OR DIR}$\}$; $\mathcal{T}_2$: $\{P_2$ AND {ED AND QE} AND {PL OR DIR}$\}$; $\mathcal{T}_3$: $\{\{\mathrm{PE}_1$ OR $\mathrm{PE}_2\}$ OR {ED AND QE} OR {PL OR DIR}$\}$. Notations about the attributes are enumerated in Table 1. It can be seen that there are two shared subpolicies $\dot{\mathcal{T}}$: {ED AND QE} and $\ddot{\mathcal{T}}$: {PL OR DIR} among the three access policies, and their root node thresholds are different.

In the previous schemes, each data object is encrypted separately, and computations involving these shared subpolicies are repeatedly performed. The method in the literature [24] cannot also be directly applied to this situation. Therefore, to avoid these repeated computations and deal with multiple shared subpolicies, the scheme of this paper includes improvements based on Li's work [24]. The contributions of this paper are as follows:

(1) For the case where there are multiple shared subpolicies among access policies and the root nodes of all access trees are the same, the optimal policy integration method is provided to reduce the computational overhead of encryption related to share subpolicies.

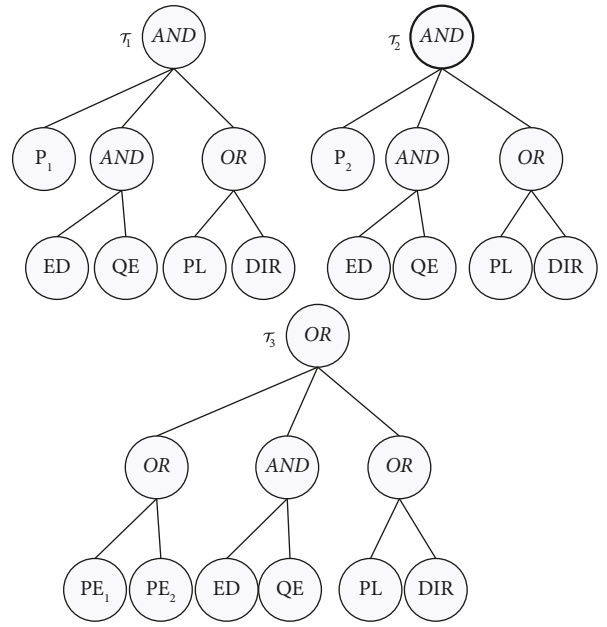

Figure 1: Example of access control policies with multiple shared subpolicies.

Table 1: Notations.

Symbol	Attribute
P_1	Project 1 team member
P_2	Project 2 team member
PE_1	Production engineer for Project 1
PE_2	Production engineer for Project 2
ED	Engineering department
QE	Quality engineer
PL	Project lead
DIR	Director

(2) For the case where there are multiple shared subpolicies among access policies and the root nodes of the access trees are various, the optimal policy integration method is provided to reduce the computational overhead of encryption related to share subpolicies.

2. Related Work

As an extension of IBE [3], in 2005, Sahai and Waters first proposed an identity encryption scheme based on biometric information called Fuzzy-IBE [2]. In 2006, Goyal et al. extended Fuzzy-IBE to KP-ABE [4]. In 2007, Bethencourt et al. proposed CP-ABE [5]. Comparing the two types of ABE, we find that CP-ABE is more suitable for the access control scenario on cloud storage. In order to improve the efficiency of the scheme, scholars have made the following efforts:

The literature [6–16] contains studies on reducing the computation consumption of encryption and decryption. Owing to the limited computing power of users, the server must undertake some heavy encryption and decryption computing tasks to improve the operation efficiency of the

scheme. Green et al. [6] designed an ABE scheme for outsourcing decryption. In this scheme, the improved key generation algorithm generates two keys, in which the short El Gamal key is kept as a private key by the user, and the transformation key is sent to the cloud. The cloud uses the transformation key to convert the ciphertext of the data that the user can access into a short El Gamal ciphertext, the same as the ciphertext obtained by encrypting the same data with the El Gamal key. Users only need to recover the plaintext with their private keys after a simple computational process, reducing users' decryption cost. Zhou and Huang [7] proposed an outsourcing encryption and decryption CP-ABE scheme, which divides the access tree of the associated ciphertext into two parts connected by the AND gate. One part of the access structure is outsourced to the cloud server for encryption, and the user only needs to encrypt the other part, which reduces the user's computational overhead. However, this scheme can only support the access tree with the root node of the AND gate. Asim et al. [8] constructed a CP-ABE scheme for outsourcing encryption and decryption, using two independent semitrusted agents for encryption and decryption, respectively. Hohenberger and Waters [9] designed an online/offline ABE scheme. In the encryption process, the computation task is divided into two parts: the offline precomputation, unrelated to the ciphertext, and the online computation, related to the ciphertext. The aim is to reduce the computational overhead of online encryption. Li and Zhang [10] proposed a CP-ABE scheme with offline/online encryption and outsourced decryption. In the encryption phase, the complex calculations are performed offline on the data owner's device, and the output is an intermediate ciphertext. The entire encryption process can be completed with only one online exponential operation on the intermediate ciphertext. In the decryption phase, the cloud server determines if the attributes of the data user meet the access structure. If so, it performs partial decryption. Partial decryption cannot reveal any information about the plaintext. Then, the user can achieve the plaintext after only a few exponential operations based on the partial decryption results. Cao et al. [11] proposed an ABE scheme, which uses the substantial computing power of a trusted third party to complete most decryption operations. The data owner specifies access policies for files, and the trusted third party embeds the attributes in the access policy into the ciphertext. The ciphertext is predecrypted by the proxy server and then decrypted by the user. Li et al. [12] proposed a multi-authority CP-ABE access control scheme. It supports transferring part of the encryption and decryption computing from Internet of Things (IoT) devices to neighboring fog nodes, reducing the computational overhead of IoT devices and making them more suitable for IoT applications. Leng and Luo [13] proposed an ABE scheme supporting encryption outsourcing. They used the method in the literature [14] to transform the access tree into a shared access matrix. The matrix construction and encryption are outsourced to the cloud server, so the user only needs three exponential operations to complete the encryption. This scheme can be applied to mobile devices. Liu and Guo [15] combined the proxy re-encryption of multiple authorization centers with outsourcing encryption and decryption. During encryption, part of the pairing operations is completed by the cloud. During decryption, the blinded private key is used for outsourcing decryption, and then the user obtains the plaintext through only one exponential operation. Hwang and Lee [16] proposed a CP-ABE scheme that can provide signature-based verifiable outsourcing in the cloud environment. The data owners embed their signature into the ciphertext. When users decrypt the ciphertext, they can verify if the obtained data object is signed by the data owner and can then verify its integrity. During decryption, the outsourcing server performs partial decryption first, and the end users only need to complete two pairing operations to obtain the plaintext. In their scheme, the ciphertext has a constant size.

There has also been research on how to improve the efficiency of CP-ABE schemes [17–21, 27]. Emura et al. [17] proposed a CP-ABE scheme with a constant ciphertext length to reduce the encryption and decryption time. Still, the scheme only supports the "AND" gate access structure, and the expression ability of the access policy is not rich enough. Subsequently, Herranz et al. [18] proposed a CP-ABE scheme with a constant ciphertext length and threshold access structure. Odelu et al. [19] designed a provably secure CP-ABE scheme with constant-size keys and ciphertext based on the RSA "AND" gate access structure, which is suitable for the mobile cloud computing environment. Tamizharasi et al. [20] proposed a CP-ABE scheme for privacy protection, which eliminates redundant attributes in the policy by constructing an assignable matrix, and reduces the ciphertext size to a constant that is independent of the number of attributes of the data user. Yang et al. [21] proposed a completely secure CP-ABE scheme with constant-size ciphertext, whose decryption computation only requires three pairing operations and some multiplication operations, independent of the number of attributes involved. This scheme is suitable for lightweight devices in IoT applications. In addition, Fugkeaw and Sato [27] combined role-based access control (RBAC) [28] with CP-ABE, improved attribute revocation and the key update in CP-ABE, and reduced the overhead of users' key distributions and updates.

There are also some works to optimize the scheme and improve the efficiency of the scheme by modifying multiple access policies of different data. Wang et al. [22] proposed an integrated access structure CP-ABE scheme, which integrates multiple access trees of different data of the same user into one access tree and encrypts these data together. This integration structure can save storage overhead and encryption and decryption costs. Based on the literature [22], Li et al. [23] further extended the tree access structure and set level nodes under nonleaf nodes. Once the access structure of one level node is satisfied among level nodes of the same level under the same node, all others can also be treated as satisfied. This scheme achieves more flexible access control while keeping the encryption and decryption time unchanged. Still, these schemes do not consider the scenario of multiple data with the same subpolicies. The authors of [24] proposed a CP-ABE scheme based on shared subpolicy.

It combines tree access policies of different data, avoids repeated computations in the encryption process as much as possible, and realizes unified encryption for the shared subpolicy, reducing the computational overhead. Based on the literature [24], Zhao [25] proposed a CP-ABE scheme in which the data owner outsources most of the decryption operations to the cloud server. First, the cloud server performs predecryption to obtain the intermediate ciphertext. Then the user uses the intermediate ciphertext to recover the plaintext through simple exponentiation, reducing the decryption cost for the user. Xue et al. [26] proposed an improved LSSS matrix expression. When users decrypt the ciphertext for the first time, they store parameters about the given subpolicies, which can be reused in subsequent data decryption of access policies containing the same subpolicies to avoid unnecessarily repeated computations.

3. Preliminaries

3.1. Bilinear Pairings. Let G_1 and G_2 be two multiplicative cyclic groups of prime order p. Let g be a generator of G_1 and e be a bilinear map, e: $G_1 \times G_1 \longrightarrow G_2$. The bilinear map e has the following properties:

(1) Bilinearity. For all $u, v \in G_1$ and $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$.

(2) Non-degeneracy. $e(g, g) \neq 1$.

(3) Computability. There is an efficient algorithm to compute $e(x, y)$ for all $x, y \in G_1$.

3.2. Access Tree. Let $\mathcal{T}$ be an access tree, which is composed of leaf nodes and nonleaf nodes. Each nonleaf node represents a threshold gate, described by its children and a threshold value. Let num_x represent the number of children of a nonleaf node N_x, and k_x be the threshold value of N_x. Then, $0 < k_x \leq \text{num}_x$. When $k_x = 1$, the threshold gate is an "OR" gate. When $k_x = \text{num}_x$, the threshold gate is an "AND" gate. Each leaf node is associated with an attribute, and the threshold value of the leaf node is set to 1; that is, $k_x = 1$.

For each node N_x in the access tree, defining $\text{parent}(N_x)$ to represent the parent node of node N_x, each parent node assigns an index value to all its children, and $\text{index}(N_x)$ is defined to represent the index value of node N_x; that is, $1 \leq \text{index}(N_x) \leq \text{num}_x$. For the leaf node, defining $att(N_x)$ represents the attribute associated with node N_x.

3.3. Secret Sharing Scheme. Randomly select $s \in \mathbb{Z}_p^*$ as the secret value to be shared, and then set the value corresponding to the root node N_0 of the access tree as the secret value s. The number of children is num_0, and the assignment rules of the children of N_0 are as follows:

(i) If the root node N_0 is an "OR" gate, set the value of each child of N_0 to s

(ii) If N_0 is an "AND" gate, a $\text{num}_0 - 1$ degree polynomial f is randomly selected so that $f(0) = s$; for the child node of N_0 whose index value is j, set its value to $f(j)$

(iii) If the root node is other threshold, set the threshold value of N_0 as k_0, and randomly select a $k_0 - 1$ degree polynomial f so that $f(0) = s$; for the child node of N_0 whose index value is j, set its value to $f(j)$.

According to the above rules, assign a value to each node in the access tree from the root node.

3.4. Lagrange Interpolation Formula. Select a prime number p and let $f(x)$ be a polynomial of degree n on $\mathbb{Z}_p$. If $n + 1$ different points $(x_i, f(x_i)), i = 0, \ldots, n$ of $f(x)$ are given, the polynomial $f(x)$ can be uniquely determined by the following Lagrange interpolation formula:

$$f(x) = \sum_{i=0}^{n} f(x_i) \left(\prod_{j=0, j \neq i}^{n} \frac{x - x_j}{x_i - x_j} \right) (\text{mod } q), \tag{1}$$

where $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} x - j / i - j$ is defined as the Lagrange coefficient polynomial of $f(i)$. Here, $i \in \mathbb{Z}_p$, S is a set of $n + 1$ elements in $\mathbb{Z}_p$.

3.5. Shared Subpolicy

Definition 1 (Shared subpolicy). If there are identical subtree(s) among multiple access trees, the subtree(s) is called the shared subpolicy (policies) of these access trees.

Access trees may have multiple shared subpolicies; among these shared subpolicies, one cannot be a subtree of another.

3.6. Integrated Access Tree

Definition 2 (Integrated access tree). This is a directed tree formed via merging multiple access trees containing shared subpolicies.

Definition 3 (Root node of integrated access tree). When access trees containing shared subpolicies are integrated, the cross-node, after integrating all access trees, is called the root node (cross-node) of the integrated access tree, and the cross-node is unique.

3.7. CP-ABE Security Model. The security model in this study is expressed through a security game between a challenger and an adversary. The process is as follows:

(i) Setup: The adversary selects a challenge access structure $\mathcal{T}$ and sends it to the challenger. The challenger runs the $\text{Setup}(1^\lambda)$ algorithm, obtains the public key PK and the master key MSK, sends PK to the adversary and keeps MSK for itself.

(ii) Query phase 1: The adversary queries the challenger for a series of secret keys related to the attribute set $A_{q_i}, i = 1, 2, 3, \ldots$; that is, the adversary sends the attribute A_{q_i} to the challenger, and then the challenger returns the corresponding private key to the

adversary. The challenger answers the adversary's query until the adversary stops querying.

(iii) Challenge: The adversary sends two messages of equal length M_0 and M_1 to the challenger, and the challenger randomly tosses a coin $b \in \{0, 1\}$, encrypts M_b with $\mathcal{T}$ as the access control policy, obtains the challenge ciphertext CT, and sends the ciphertext CT to the adversary.

(iv) Query phase 2: The adversary can continue to send more secret key requests for other attribute sets to the challenger. Similarly, it is required that all private key queries do not meet the challenge access structure $\mathcal{T}$.

(v) Guess: The adversary outputs a guess $b' \in \{0, 1\}$ about b. If $b' = b$, the adversary wins the game. The adversary's advantage in the game is defined as $|\Pr[b' = b] - 1/2|$.

Definition 4. A CP-ABE scheme is said to be safe if no polynomial-time adversary breaches the above security game by a non-negligible advantage.

4. Integration of Access Trees: Access Trees with Multiple Shared Subpolicies

Li [24] considered the integration of different access trees with a single shared subpolicy. However, this paper's work considers the scenario of multiple shared subpolicies among different access trees. We start with the case of the integration of two access trees containing two shared subpolicies. Accordingly, this integration method is extended to integrate multiple access trees with multiple shared subpolicies, consequently forming a general access tree integration method. During the integration, the nodes we need to consider are limited to the parent nodes of the shared subpolicies. As for higher-level nodes, their relative positions remain unchanged during integration. Therefore, the root node of the original access tree (the access tree before modification and the integration) refers to the parent node of the shared subpolicies hereafter.

Below, the symbols $\mathcal{T}_i(\text{Other}), i = 1, \ldots, n$ are used to represent the nonshared subpolicy part in access trees $\mathcal{T}_i, i = 1, \ldots, n$; $\dot{\mathcal{T}}$ (AND) and $\ddot{\mathcal{T}}$ (OR) represent two kinds of shared subpolicies with the AND root node and OR root node, respectively. In addition, there are no restrictions on the form of the shared subpolicies and other parts of access trees.

4.1. Modification of Access Tree. Before integration, access trees need to be modified as preparation. Accordingly, first, we give the specification of the modification of access trees.

Definition 5 (Modification of access tree). If the root node of an access tree is the AND (OR) gate, then, we add an AND (OR) child node just under the root node, denoted by $N^{(1)}$ ($N^{(2)}$). Then, all shared subpolicies in the access tree are merged under the $N^{(1)}$ ($N^{(2)}$) node. Therefore, $N^{(1)}$ ($N^{(2)}$) becomes the parent node of all shared subpolicies in this access tree. This process is called the modification of the access tree.

Consider two access trees (Figure 2), $\dot{\mathcal{T}}$ (AND) and $\ddot{\mathcal{T}}$ (OR), which are shared subpolicies of the two access trees. For the access tree whose root node is AND, the two shared subpolicies are merged with an AND node, which is denoted as $N^{(1)}$. For the access tree whose root node is OR, the two shared subpolicies are merged with an OR node, which is denoted as $N^{(2)}$. After modification, we achieve two modified access trees, as shown in Figure 3.

4.2. Integration of Access Trees

4.2.1. Integration of Two Access Trees with the Same Root Nodes. We assume that the access trees that need to be integrated are shown in Figure 4(a), and they both have the AND root nodes. First, we modify them according to Definition 5. We take the node that are added when merging the shared subpolicies of each access tree, make these two nodes one, and choose it to be the root node of the integrated access tree. The final integrated access tree is shown in Figure 4(b). If there are two access trees with the OR root nodes, as shown in Figure 5(a), the integration process is similar. After integrating the access trees, the result is as shown in Figure 5(b).

4.2.2. Integration of Two Access Trees with the Different Root Nodes. We may have two or more access trees that need to be integrated, and they have multiple shared subpolicies. Unfortunately, these trees have various root nodes. In this case, only one of the subpolicies can be chosen, so its root node becomes the cross-node of the whole integrated access tree. The following question then arises: how to select this cross-node (subpolicy)? To reduce the overall encryption time cost of data whose access trees are being integrated, we need to compare the encryption time costs related to every shared subpolicy and choose the cross-node of the integrated access tree on this basis. Therefore, the cross-node (subpolicy) selection rule is provided first.

Rule 1 (Cross-node selection rule). Knowing all shared subpolicies among access trees, we select the cross-node by following the rules mentioned:

(i) Compare the number of leaf nodes contained in each shared subpolicy:

  (1) If the numbers of leaf nodes are different, then the root node of the shared subpolicy with the largest number of leaf nodes is selected as the root node of the integrated access tree.

  (2) If the numbers of leaf nodes are the same, then the root node of the shared subpolicy with the largest number of the AND gate children is selected as the root node of the integrated access tree.

The number of leaf nodes is given the highest priority in Rule 1, and the reason is explained later.

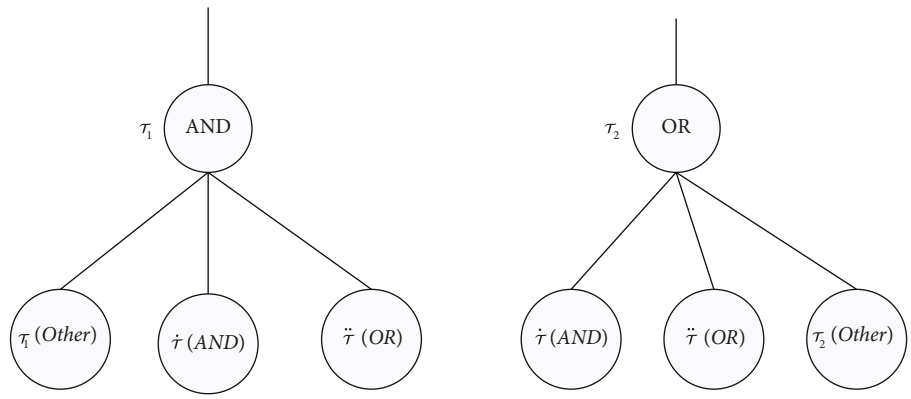

Figure 2: Example of access trees with the AND and OR root nodes.

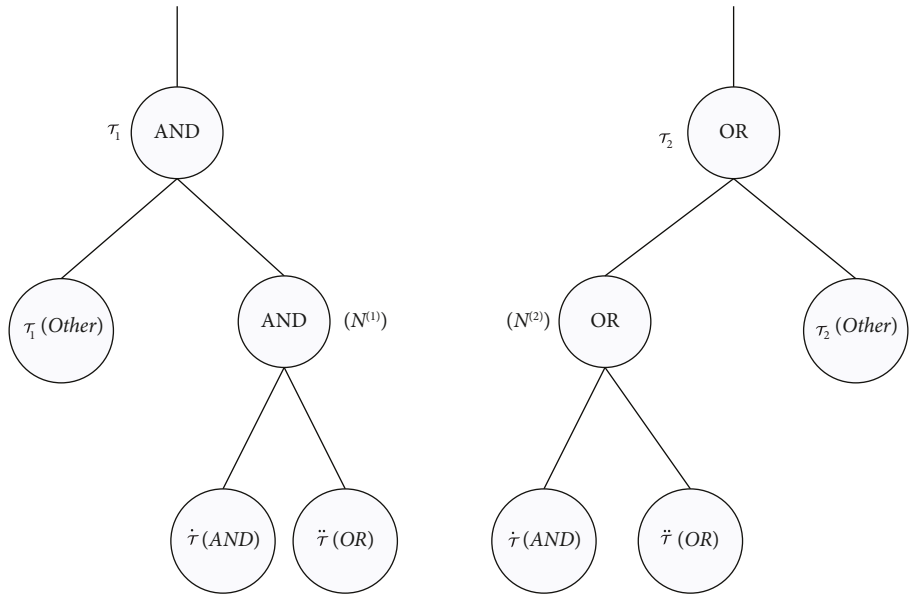

Figure 3: Example of modified of access trees with the AND and OR root nodes.

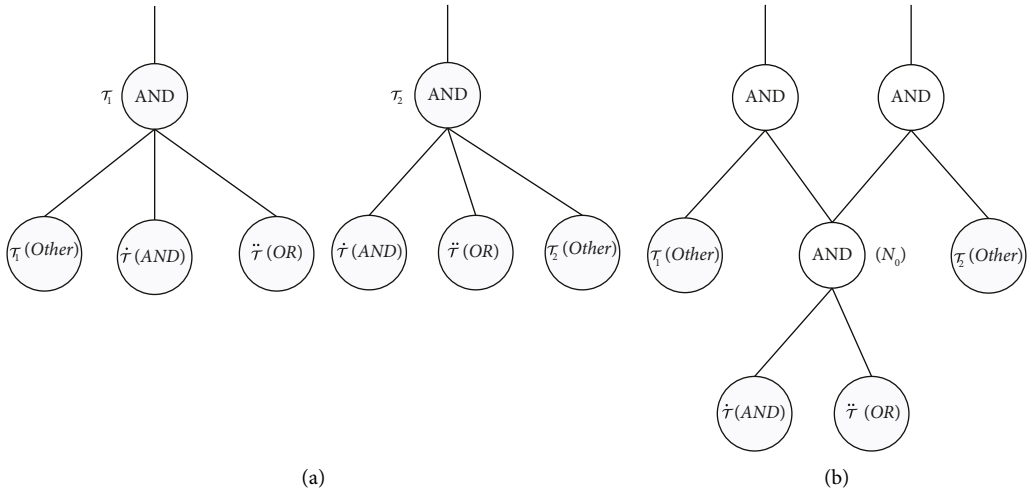

Figure 4: Integration of two access trees with the AND root nodes.

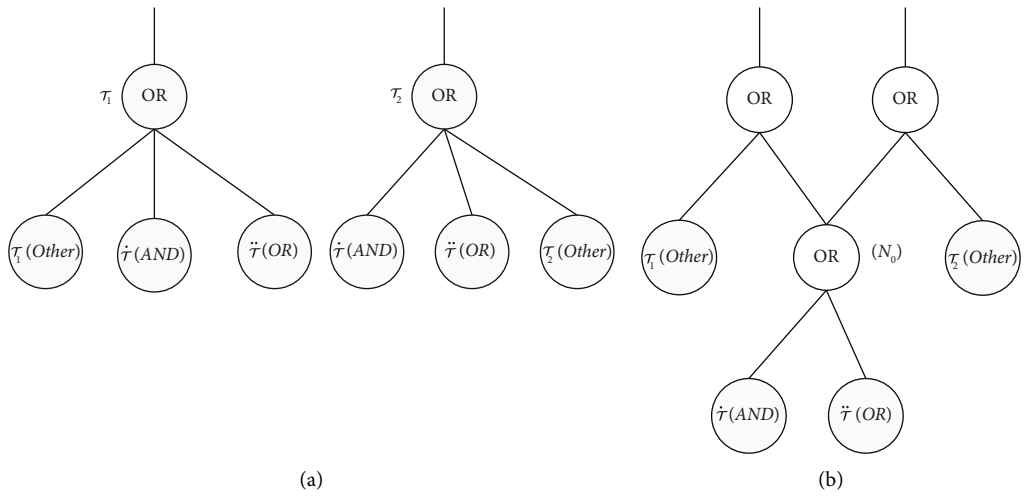

Figure 5: Integration of two access trees with the OR root nodes.

Definition 6 (Core shared subpolicy). When we integrate access trees with different root nodes, if the root node of a shared subpolicy becomes the cross-node of the integrated access tree, this shared subpolicy is called the core shared subpolicy.

Taking the access trees in Figure 2 as an example, we consider the integration of the two access trees, whose root nodes are AND and OR, respectively. First, modify the access trees according to Definition 5. Because they have two shared subpolicies, we select the cross-node (subpolicy) according to Rule 1. All nodes from the two access trees except nodes in the chosen subpolicy are connected to the cross-node in their original order, keeping their relative positions unchanged. Finally, we establish an integrated access tree, as shown in Figure 6; the selected cross-node becomes the root node of the entire integrated access tree. During encryption of the corresponding data of these two access trees, the encryption-related computation for the core shared subpolicy can be done only once.

In Figure 2, the root nodes of the two shared subpolicies are the AND gate and OR gate. In fact, if the root nodes of the shared subpolicies are both OR or AND, the cross-node can also be selected for integration according to Rule 1.

4.2.3. Integration of More than Three Access Trees. As a typical example, the integration of four access trees is discussed below to give a general integration method for multiple access trees. Consider the four access trees in Figure 7, which contain shared subpolicies with root nodes of different gates. The root nodes of $\mathcal{T}_1$ and $\mathcal{T}_3$ are AND, and the root nodes of $\mathcal{T}_2$ and $\mathcal{T}_4$ are OR. First, the four access trees are modified according to Definition 5; second, the access trees whose root nodes are the same gate are integrated first according to 4.2.1. That is, the access trees $\mathcal{T}_1$ and $\mathcal{T}_3$, $\mathcal{T}_2$ and $\mathcal{T}_4$ are integrated first, respectively; subsequently, the two integrated access trees are further integrated. At this time, we select the cross-node of the integrated access tree, according to Rule 1. Then, we connect each access tree to the cross-node according to the connection mode of 4.2.2; the final integrated access tree of the four access trees is shown in Figure 8, and the cross-node is the root node of the integrated access tree.

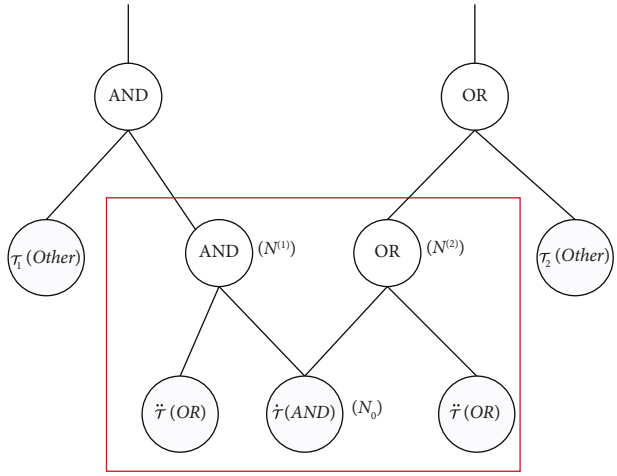

Figure 6: An integrated access tree of two access trees with the AND and OR root nodes.

In Figure 7, the root nodes of the two shared subpolicies are the AND gate and OR gate. In fact, if the root nodes of the shared subpolicies are both OR or AND, the cross-node can also be selected for integration, according to rule 1.

4.3. General Integration Method of Access Trees. Based on the above cases, an integration method of multiple access trees containing multiple shared subpolicies is provided.

(i) (Integration Method 1): Integration of access trees with the same root nodes containing multiple shared subpolicies:

Step 1: Modify access trees according to Definition 5 to merge multiple shared subpolicies into one shared subpolicy for each access tree.
Step 2: Take the root nodes of the new shared subpolicy built from multiple shared subpolicies

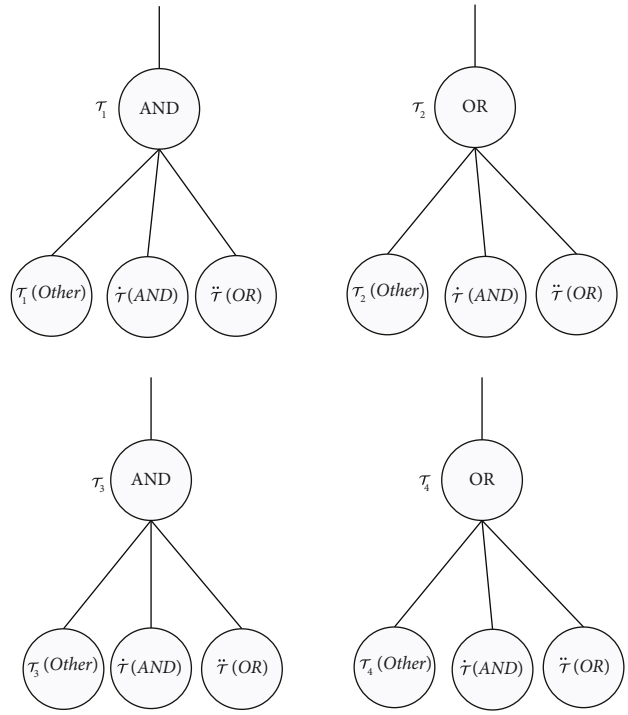

Figure 7: Four access trees.

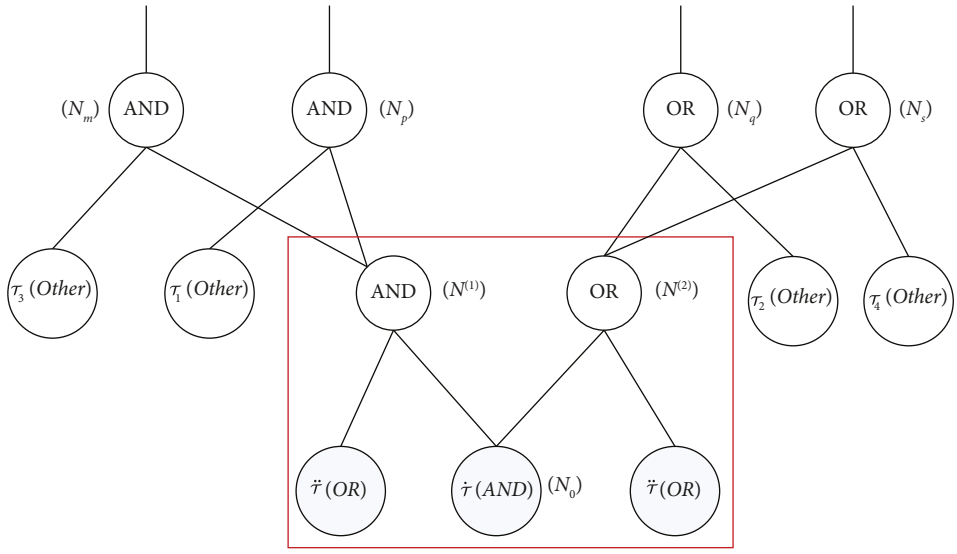

Figure 8: Integrated access tree of four access trees.

from all access trees, make these nodes all-in-one, and choose this node to be the cross-node of the integrated access tree.

Step 3: Connect all nodes from access trees except nodes in the newly built shared subpolicy to the cross-node in their original order, keeping their relative positions unchanged. Make this cross-node the root node of the integrated access tree.

The whole process and the integrated access tree can be seen in Figures 4 and 5.

(ii) (Integration Method 2): Integration of access trees with various root nodes containing multiple shared subpolicies:

Step 1: Integrate access trees whose root nodes are the same according to the Integration Method 1, and achieve two integrated access trees.

Step 2: Select the cross-node (subpolicy) for the two new integrated access trees according to the Rule 1.

Step 3: Connect all nodes from access trees except nodes in the chosen subpolicy to the cross-node in

their original order, keeping their relative positions unchanged. Make this cross-node the root node of the whole integrated access tree.

Using Integration Method 1, we first build two integrated access trees with the AND cross-node and OR cross-node, respectively. Then, we integrate them into one final integrated access tree. The whole process and the integrated access tree can be seen in Figure 8.

According to Integration Method 2, we integrate the example in the introduction (Figure 1), and the final integrated access tree is shown in Figure 9:

4.4. Secret Value Assignment for Nodes in an Integrated Access Tree. Inspired by the method in [24], we develop a secret sharing method suitable for our integrated access tree.

We take the access trees illustrated in Figure 8 as an example. N_0 is the root node of the integrated access tree. It starts from the root node, and a polynomial $f_t(x)$ is generated for each node N_t in the integrated access tree. Let the degree of the polynomial be the threshold value of the node minus 1; that is, $d_t = k_t - 1$. Start from the root node N_0. For each node, generate the polynomial as follows: First, choose a random number $s \in \mathbb{Z}_p$, and set $f_0(0) = s$. Then, randomly select $s_{d_o}, s_{d_o-1}, \ldots, s_1 \in \mathbb{Z}_p$ to determine $f_0(x)$, and then $f_0(x) = s_{d_o}x^{d_o} + s_{d_o-1}x^{d_o-1} + \cdots + s_1x + s$ can be obtained.

$N^{(1)}$ is an AND node. Let $f_1(x) = a_{d_1}x^{d_1} + a_{d_1-1}x^{d_1-1} + \cdots + a_1x + a_0$. Let the index value of the N_0 is 1; that is, $\text{index}(N_0) = 1$. Then $s = f_1(1) = a_{d_1} + a_{d_1-1} + \cdots + a_1 + a_0$. Select the remaining d_1 points to determine $a_0 = s - (a_{d_1} + a_{d_1-1} + \cdots + a_1)$.

As $N^{(2)}$ is an OR node, set $f_2(0) = f_0(0) = s$.

For the parent nodes N_p and N_m of $N^{(1)}$, and parent nodes N_q and N_s of $N^{(2)}$, there are

$$\begin{aligned} f_p\left(\text{index}\left(N^{(1)}\right)\right) &= f_m\left(\text{index}\left(N^{(1)}\right)\right) = f_1(0) = a_0; \\ f_q\left(\text{index}\left(N^{(2)}\right)\right) &= f_s\left(\text{index}\left(N^{(2)}\right)\right) \\ &= f_2(0) = f_0(0) = s. \end{aligned} \tag{2}$$

Let

$$\begin{aligned} f_p(x) &= p_{d_p}x^{d_p} + p_{d_p-1}x^{d_p-1} + \cdots + p_1x + p_0; \\ f_m(x) &= m_{d_m}x^{d_m} + m_{d_m-1}x^{d_m-1} + \cdots + m_1x + m_0; \\ f_q(x) &= q_{d_q}x^{d_q} + q_{d_q-1}x^{d_q-1} + \cdots + q_1x + q_0; \\ f_s(x) &= s_{d_s}x^{d_s} + s_{d_s-1}x^{d_s-1} + \cdots + s_1x + s_0. \end{aligned} \tag{3}$$

Set the index value of $N^{(1)}$ in $f_p(x)$ and $f_m(x)$ as 1 and the index value of $N^{(2)}$ in $f_q(x)$ and $f_s(x)$ as 1; that is, $\text{index}(N^{(1)}) = \text{index}(N^{(2)}) = 1$. Consequently, $a_0 = f_p(1) = f_m(1)$, $s = f_q(1) = f_s(1)$, so

$$\begin{aligned} a_0 &= p_{d_p} + p_{d_p-1} + \cdots + p_1 + p_0 \\ &= m_{d_m} + m_{d_m-1} + \cdots + m_1 + m_0; \\ s &= q_{d_q} + q_{d_q-1} + \cdots + q_1 + q_0 = s_{d_s} + s_{d_s-1} + \cdots + s_1 + s_0. \end{aligned} \tag{4}$$

Select the remaining $d_m + d_p + d_q + d_s$ points and calculate:

$$\begin{aligned} p_0 &= a_0 - \left(p_{d_p} + p_{d_p-1} + \cdots + p_1\right); \\ m_0 &= a_0 - \left(m_{d_m} + m_{d_m-1} + \cdots + m_1\right); \\ q_0 &= s - \left(q_{d_q} + q_{d_q-1} + \cdots + q_1\right) = s; \\ s_0 &= s - \left(s_{d_s} + s_{d_s-1} + \cdots + s_1\right) = s. \end{aligned} \tag{5}$$

Thus, $f_p(x)$, $f_m(x)$, $f_q(x)$ and $f_s(x)$ are determined.

For other nodes N_t, let $f_t(0) = f_{\text{parent}(N_t)}(\text{index}(N_t))$, and select the remaining d_t values to determine $f_t(x)$. Therefore, the polynomial of each node in the integrated access tree can be determined.

Note that according to the above construction, the polynomial construction and assignment of secret values for the nodes in the box illustrated in Figure 8 only depend on the secret value of the root node of the integrated access tree and the corresponding polynomial. Fortunately, the secret value of the root node of the integrated access tree and the corresponding polynomial are set at the beginning. It is fixed and does not depend on other parameters. Thus, the secret value of each node in the box can be determined right after determining the secret value and the polynomial of the root node of the integrated access tree. In sum, the polynomials and secret values for the nodes in the box can be shared among access trees that form the integrated access tree to the greatest extent. The encryption-related computation for these nodes can be done once or twice, depending on the subpolicy type (described later), avoiding repeated computation.

4.5. Optimality Analysis of the Integration Method. The integration method provided in Section 4.3 has two key points: One is the selection of the cross-node of the integrated access tree if access trees have different root gates, and the other is the merging of shared subpolicies and integration. We posit that the integration method provided in Section 4.3 is optimal. The analysis is presented in the following:

4.5.1. Selection of the Cross-Node of the Integrated Access Tree. The reasons for selecting the cross-node (subpolicy) of the integrated access tree, as given in Rule 1, are as follows:

First, among multiple shared subpolicies, one subpolicy is selected so that its root node becomes the cross-node of the integrated access tree. Therefore, during the encryption of corresponding data of access trees that form the integrated access tree, the polynomials and secret values for the nodes in this subpolicy can be shared among all access trees. The encryption-related computation for this subpolicy can be done only once, avoiding repeated computation. In a word, for this subpolicy, the data owner can achieve the purpose of precomputation, one-time computation, and reuse.

Second, Rule 1 mainly has two considerations. First, during encryption, each leaf node involves exponential operations. Thus, the more leaf nodes a tree have, the higher

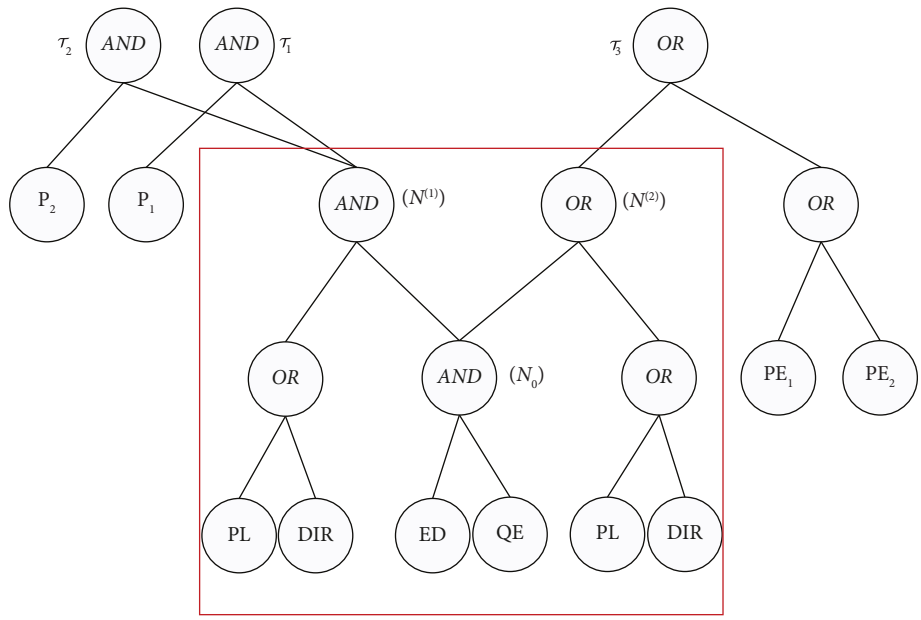

Figure 9: The integration result of the three access trees in Figure 1.

the computational cost. Second, according to the secret value assignment for nodes in the integrated access tree, there is a polynomial construction and exponential operations at the AND gate, but there is no exponential operation at the OR gate, and the secret value is directly propagated to it. By comparison, the computational overhead at the AND gate is higher than that of the OR gate. Therefore, the subpolicy with the largest amount of encryption-related computation is selected if the cross-node is selected according to Rule 1. The encryption-related computation for this subpolicy can be done only once.

4.5.2. Optimality Analysis of the Integration Method. All shared subpolicies produce one of two different results: The first is the core shared subpolicy, for which the encryption-related computation can be done only once. For the noncore shared subpolicies, each of them is embedded under $N^{(1)}$ and $N^{(2)}$ twice. So for the noncore shared subpolices, encryption-related computation needs to be done twice. Therefore, selecting the cross-node and embedding the noncore shared subpolicies into the integrated access tree ensures that the integrated access tree is optimal terms of encryption-related computation.

If we do not follow the integration method in this paper, that is, if we do not find a cross-node to integrate access trees, then for each access tree, the encryption operations related to these shared subpolicies should be performed separately during encryption. So the integration method is optimal for reducing the amount of encryption computation.

This integration method of access trees ensures that the core shared subpolicy with the largest amount of encryption-related computation can be selected. For this subpolicy, the encryption-related computation can be done once, and the computation results can be shared among these access trees. For the noncore shared subpolices, encryption-related computation needs to be done twice; the computation results under node $N^{(1)}$ are shared among the access trees whose root nodes are AND, and the computation results under node $N^{(2)}$ are shared among the access trees whose root nodes are OR.

5. CP-ABE Scheme for Multiple Shared Subpolicies

5.1. System Model. This scheme involves four entities: attribute authority, cloud storage platform, data owner, and data user. The system model is shown in Figure 10:

5.2. Scheme Construction. This scheme is composed of four algorithms: Setup, KeyGen, Encrypt, and Decrypt. The algorithms are defined as follows.

(1) $\text{Setup}(1^{\lambda}) \longrightarrow (\text{PK}, \text{MSK})$: The algorithm is run by the attribute authority. The input of the algorithm is the system security parameter λ, and the output is the system public key PK and the master key MSK.

(2) $\text{KeyGen}(\text{PK}, \text{MSK}, A) \longrightarrow \text{SK}$: The algorithm is run by the attribute authority and generates the corresponding private key for the data user according to their attribute sets. The inputs of the algorithm are PK, MSK and the attribute set A of the data user, and the output is the private key SK of the data user.

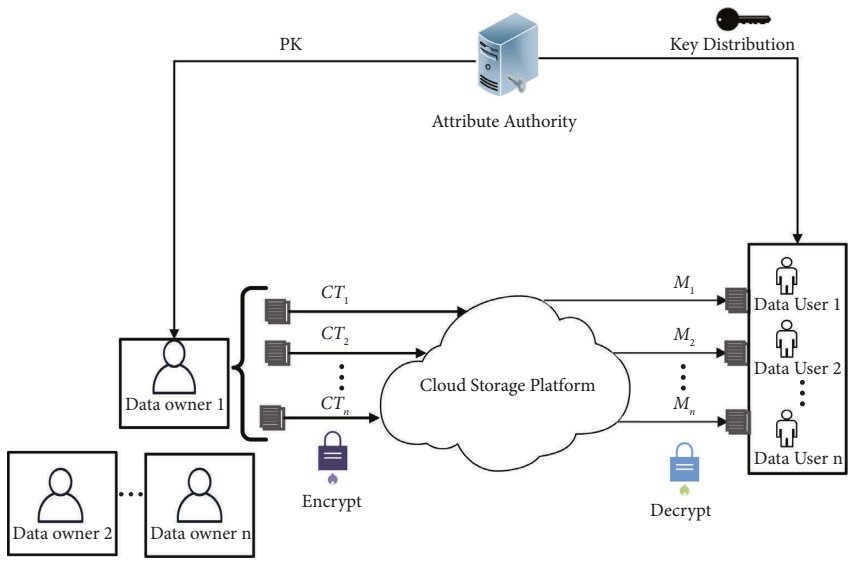

Figure 10: System model.

(3) Encrypt $(\mathrm{PK}, \{M_j\}_{1\le j\le n}, \{\mathscr{T}_j\}_{1\le j\le n}) \longrightarrow \{\mathrm{CT}_j\}_{1\le j\le n}$: The algorithm is run by the data owner, and the access tree $\mathscr{T}_j$ is used to encrypt the data M_j. Shared subpolicies exist among the corresponding access trees $\mathscr{T}_j$, $j = 1, 2, \ldots, n$. The inputs of the algorithm are PK, the plaintexts $\{M_j\}_{1\le j\le n}$ and the access trees $\{\mathscr{T}_j\}_{1\le j\le n}$, and the outputs are the ciphertexts $\{\mathrm{CT}_j\}_{1\le j\le n}$.

(4) Decrypt $(\mathrm{PK}, \mathrm{SK}, \mathrm{CT}_j) \longrightarrow M_j$: The algorithm is run by the data user, who decrypts the ciphertext CT_j with his private key SK. The inputs of the algorithm are PK, SK of the data user, and the ciphertext CT_j, and the output is the corresponding plaintext M_j.

5.3. Scheme Details. The four algorithms of the CP-ABE scheme for multiple shared subpolicies proposed in this paper are described as follows.

5.3.1. Setup. Select two multiplicative cyclic groups G_1, G_2 with the prime order p, and with g as the generator of G_1. Define the bilinear map e: $G_1 \times G_1 \longrightarrow G_2$ and select the hash function $\mathscr{H}$: $\{0,1\}^* \longrightarrow G_1$. The attribute authority selects two random numbers $\alpha, \beta \in \mathbb{Z}_p^*$ and generates the system public key PK and the master key MSK as follows:

$$\begin{aligned} \mathrm{PK} &= \left(G_1, g, g^{\beta}, e(g,g)^{\alpha}\right); \\ \mathrm{MSK} &= (\beta, g^{\alpha}). \end{aligned} \tag{6}$$

The attribute authority publishes PK to all data owners and data users.

5.3.2. KeyGen. According to the attribute set A of the data user, the attribute authority generates the corresponding private key SK. The attribute authority first randomly selects $r \in \mathbb{Z}_p$, then randomly selects $r_i \in \mathbb{Z}_p$ for each attribute i in the attribute set A, and finally calculates and generates SK:

$$\mathrm{SK} = \left(D = g^{\alpha + r/\beta}, \forall i \in A\colon D_i = g^{r} \cdot \mathscr{H}(i)^{r_i}, D_i^{'} = g^{r_i}\right). \tag{7}$$

The attribute authority sends SK to the data user.

5.3.3. Encrypt. The integration of four access trees, including two shared subpolicies, is taken as an example to describe the algorithm. This is a typical example, and more complicated cases can be simplified accordingly.

As shown in Figure 7, $\mathscr{T}_1$, $\mathscr{T}_2$, $\mathscr{T}_3$, and $\mathscr{T}_4$ have shared subpolicies; $\mathscr{T}_1$ and $\mathscr{T}_3$ are access trees with the AND root nodes; and $\mathscr{T}_2$ and $\mathscr{T}_4$ are access trees with the OR root nodes. We can integrate them into an access tree, as shown in Figure 8, where N_0 is the root node of the integrated access tree. In addition, the specific forms of the tree under the shared subpolicies and the nonshared subpolicies in the scheme are not specified. They can be any access subtrees.

The encryption algorithm constructs polynomials and chooses values for each node in the integrated access tree according to the secret value assignment method described in Section 4.4. Then, the ciphertexts are built according to the traditional CP-ABE encryption scheme.

Let $Y(\mathscr{T}_j)$, $j = 1, 2, 3, 4$ be the set of leaf nodes of each access tree, $Y(\mathscr{T})$ be a set of leaf nodes of the shared subpolicies, and $Y(\widehat{\mathscr{T}})$ be a set of leaf nodes of the core shared subpolicy. Then, the ciphertexts of data M_1, M_2, M_3, and M_4 are as follows:

$$
\begin{aligned}
&\mathrm{CT}_1 = \left(\mathcal{T}_1, \widetilde{C} = M_1 \cdot e(g,g)^{\alpha f_p(0)}, C = g^{\beta f_p(0)}\right.,\\
&\forall N_\eta \in Y(\mathcal{T}_1) - Y(\mathcal{T}):\ C_{N_\eta} = g^{f_\eta(0)}, C'_{N_\eta} = \mathcal{H}\left(att\left(N_\eta\right)\right)^{f_\eta(0)},\\
&\forall N_\theta \in Y(\mathcal{T}) - Y(\widehat{\mathcal{T}}):\ C_{N_\theta} = g^{f_\theta(0)}, C'_{N_\theta} = \mathcal{H}\left(att\left(N_\theta\right)\right)^{f_\theta(0)},\\
&\forall N_\rho \in Y(\widehat{\mathcal{T}}):\ C_{N_\rho} = g^{f_\rho(0)}, C'_{N_\rho} = \mathcal{H}\left(att\left(N_\rho\right)\right)^{f_\rho(0)}\Big),\\
&\mathrm{CT}_2 = \left(\mathcal{T}_2, \widetilde{C} = M_2 \cdot e(g,g)^{\alpha f_q(0)}, C = g^{\beta f_q(0)}\right.,\\
&\forall N_\eta \in Y(\mathcal{T}_2) - Y(\mathcal{T}):\ C_{N_\eta} = g^{f_\eta(0)}, C'_{N_\eta} = \mathcal{H}\left(att\left(N_\eta\right)\right)^{f_\eta(0)},\\
&\forall N_\theta \in Y(\mathcal{T}) - Y(\widehat{\mathcal{T}}):\ C_{N_\theta} = g^{f_\theta(0)}, C'_{N_\theta} = \mathcal{H}\left(att\left(N_\theta\right)\right)^{f_\theta(0)},\\
&\forall N_\rho \in Y(\widehat{\mathcal{T}}):\ C_{N_\rho} = g^{f_\rho(0)}, C'_{N_\rho} = \mathcal{H}\left(att\left(N_\rho\right)\right)^{f_\rho(0)}\Big),\\
&\mathrm{CT}_3 = \left(\mathcal{T}_3, \widetilde{C} = M_3 \cdot e(g,g)^{\alpha f_m(0)}, C = g^{\beta f_m(0)}\right.,\\
&\forall N_\eta \in Y(\mathcal{T}_3) - Y(\mathcal{T}):\ C_{N_\eta} = g^{f_\eta(0)}, C'_{N_\eta} = \mathcal{H}\left(att\left(N_\eta\right)\right)^{f_\eta(0)},\\
&\forall N_\theta \in Y(\mathcal{T}) - Y(\widehat{\mathcal{T}}):\ C_{N_\theta} = g^{f_\theta(0)}, C'_{N_\theta} = \mathcal{H}\left(att\left(N_\theta\right)\right)^{f_\theta(0)},\\
&\forall N_\rho \in Y(\widehat{\mathcal{T}}):\ C_{N_\rho} = g^{f_\rho(0)}, C'_{N_\rho} = \mathcal{H}\left(att\left(N_\rho\right)\right)^{f_\rho(0)}\Big),\\
&\mathrm{CT}_4 = \left(\mathcal{T}_4, \widetilde{C} = M_4 \cdot e(g,g)^{\alpha f_s(0)}, C = g^{\beta f_s(0)}\right.,\\
&\forall N_\eta \in Y(\mathcal{T}_4) - Y(\mathcal{T}):\ C_{N_\eta} = g^{f_\eta(0)}, C'_{N_\eta} = \mathcal{H}\left(att\left(N_\eta\right)\right)^{f_\eta(0)},\\
&\forall N_\theta \in Y(\mathcal{T}) - Y(\widehat{\mathcal{T}}):\ C_{N_\theta} = g^{f_\theta(0)}, C'_{N_\theta} = \mathcal{H}\left(att\left(N_\theta\right)\right)^{f_\theta(0)},\\
&\forall N_\rho \in Y(\widehat{\mathcal{T}}):\ C_{N_\rho} = g^{f_\rho(0)}, C'_{N_\rho} = \mathcal{H}\left(att\left(N_\rho\right)\right)^{f_\rho(0)}\Big).
\end{aligned}
\tag{8}
$$

The ciphertext components $C_{N_\rho} = g^{f_\rho(0)}, C'_{N_\rho} = \mathcal{H}(att(N_\rho))^{f_\rho(0)}, \forall N_\rho \in Y(\widehat{\mathcal{T}})$ are calculated only once and shared among CT_1, CT_2, CT_3, and CT_4. The ciphertext components $C_{N_\theta} = g^{f_\theta(0)}, C'_{N_\theta} = \mathcal{H}(att(N_\theta))^{f_\theta(0)}, \forall N_\theta \in Y(\mathcal{T}) - Y(\widehat{\mathcal{T}})$ are calculated once for CT_1 and CT_3 and shared between them; the same ciphertext components are calculated again for CT_2 and CT_4 and shared between them. The remaining parts of each ciphertext are calculated separately, and they cannot be shared among CT_1, CT_2, CT_3, and CT_4.

Finally, the data owner uploads CT_1, CT_2, CT_3, and CT_4 to the cloud storage platform.

5.3.4. Decrypt. Suppose that the attribute set of a data user is A, their private key is SK, and the data object they want to access is M_1. The data user first downloads the ciphertext CT_1 from the cloud storage platform and then decrypts it with SK. The decryption process is the recursive algorithm described in the following:

Let N_j represent the node in $\mathcal{T}_1$. If N_j is a leaf node and $att(N_j) = i \in A$, then calculate:

$$
\begin{aligned}
F_j &= \mathrm{DecryptNode}\left(\mathrm{CT}, \mathrm{SK}, N_j\right)\\
&= \frac{e\left(D_i, C_{N_j}\right)}{e\left(D'_i, C'_{N_j}\right)} = e(g,g)^{r f_j(0)}.
\end{aligned}
\tag{9}
$$

If N_j is a leaf node and $att(N_j) = i \notin A$, make $F_j = \perp$.

If N_j is a nonleaf node, for all children N_z of N_j, let S_j represents a set of any k_j nodes N_z, and for each N_z in the set, there is $F_z \neq \perp$. If S_j does not exist, make $F_z = \perp$. If S_j exists, then calculate:

$$
\begin{aligned}
F_j &= \prod_{z \in S_j} F_z^{\Delta_{z,S_j}(0)} = \prod_{z \in S_j} e(g,g)^{r f_z(0) \Delta_{z,S_j}(0)}\\
&= \prod_{z \in S_j} e(g,g)^{r f_{\mathrm{parent}(N_z)}\left(\mathrm{index}\left(N_z\right)\right) \Delta_{z,S_j}(0)}\\
&= \prod_{z \in S_j} e(g,g)^{r f_j(z) \Delta_{z,S_j}(0)}\\
&= e(g,g)^{r \sum_{z \in S_j} f_j(z) \Delta_{z,S_j}(0)}\\
&= e(g,g)^{r f_j(0)},
\end{aligned}
\tag{10}
$$

where $\Delta_{z,S_j}(x) = \prod_{u \in S_j, u \neq z} x - u / z - u$ is the Lagrange coefficient polynomial.

When the attribute set A of the data user satisfies the access control policy $\mathcal{T}_1$, the function DecryptNode is called at the root node N_p of $\mathcal{T}_1$:

$$
F_p = \mathrm{DecryptNode}\left(\mathrm{CT}, \mathrm{SK}, N_1\right) = e(g,g)^{r f_p(0)}. \tag{11}
$$

Finally, the data M_1 is decrypted through the following calculation:

$$\frac{\tilde{C}\cdot F_p}{e(C,D)}=\frac{M_1\cdot e(g,g)^{\alpha f_p(0)}\cdot e(g,g)^{rf_p(0)}}{e\left(g^{\beta f_p(0)},g^{\alpha+r/\beta}\right)}=M_1. \quad (12)$$

6. Security Analysis

In this section, we prove the security of the proposed scheme based on the security of the CP-ABE scheme, which has been proven to be indistinguishability under chosen-plaintext attack (IND-CPA) secure in the general group model and the random oracle model.

Theorem 1. *If the CP-ABE scheme is secure, then there is no probability that the polynomial-time adversary breaches the proposed scheme with a non-negligible advantage.*

Proof. Assuming that there is a probability polynomial-time adversary $\mathscr{A}$ that can win the security game of this section with a non-negligible advantage $Adv_{\mathscr{A}}$, we can use $\mathscr{A}$ to construct simulator $\mathscr{B}$ so that in a similar security game, $\mathscr{B}$ can also win the CP-ABE scheme with a non-negligible advantage $Adv_{\mathscr{B}}$.

(i) Setup: Adversary $\mathscr{A}$ selects a pair of challenge access structures $\{\mathscr{T}^{**},\mathscr{T}^{*}\}$ sends them to simulator $\mathscr{B}$, and simulator $\mathscr{B}$ submits the access structure to CP-ABE, where there are shared subpolicies between $\mathscr{T}^{**}$ and $\mathscr{T}^{*}$. CP-ABE runs the Setup(1^{λ}) algorithm, obtains the public key PK $= (G_1, g, g^{\beta}, e(g,g)^{\alpha})$ and the master key MSK $= (\beta, g^{\alpha})$, sends PK to simulator $\mathscr{B}$, and keeps MSK to itself; simulator $\mathscr{B}$ sends PK$'$ = PK to the adversary $\mathscr{A}$.

(ii) Query phase 1: Adversary $\mathscr{A}$ queries simulator $\mathscr{B}$ for the private key of attribute set A_{q_1}, and simulator $\mathscr{B}$ submits A_{q_1} to CP-ABE and obtains the corresponding private key $\mathrm{SK}_{q_1} = (D = g^{\alpha+r/\beta}, \{D_i = g^{r}\cdot \mathscr{H}(i)^{r_i}, D_i = g^{r_i}\}_{\forall i\in A_{q_1}})$ from CP-ABE, where r and r_i are randomly selected from $\mathbb{Z}_p$. Then, simulator $\mathscr{B}$ returns $\mathrm{SK}_{q_1}' = \mathrm{SK}_{q_1}$ to adversary $\mathscr{A}$ as a response to the query. Adversary $\mathscr{A}$ then continues to query simulator $\mathscr{B}$ for the private key of attribute set A_{q_2} and repeats the above process. Simulator $\mathscr{B}$ always answers the query of adversary $\mathscr{A}$ until adversary $\mathscr{A}$ stops querying, but all private key queries cannot meet the challenge of access structure $\{\mathscr{T}^{**},\mathscr{T}^{*}\}$.

(iii) Challenge: Adversary $\mathscr{A}$ sends two message pairs of equal length $\{M_0, {M_0}^{*}\}$ and $\{M_1, {M_1}^{*}\}$ to simulator $\mathscr{B}$, which then sends these two message pairs to CP-ABE. Then, CP-ABE randomly tosses a coin $b\in\{0,1\}$, encrypts the message M_b with $\mathscr{T}^{**}$, encrypts the message ${M_b}^{*}$ with $\mathscr{T}^{*}$, and obtains the following ciphertexts.

$$\begin{aligned}
&\mathrm{CT}=\left(\mathscr{T}^{**},\tilde{C}=M_b\cdot e(g,g)^{\alpha f_p(0)},C=g^{\beta f_p(0)}\right),\\
&\forall N_\eta\in Y(\mathscr{T}^{**})-Y(\mathscr{T}):\ C_{N_\eta}=g^{f_\eta(0)},C_{N_\eta}^{'}=\mathscr{H}\left(att\left(N_\eta\right)\right)^{f_\eta(0)};\\
&\forall N_\theta\in Y(\mathscr{T})-Y(\widehat{\mathscr{T}}):\ C_{N_\theta}=g^{f_\theta(0)},C_{N_\theta}'=\mathscr{H}\left(att\left(N_\theta\right)\right)^{f_\theta(0)};\\
&\forall N_\rho\in Y(\widehat{\mathscr{T}}):\ C_{N_\rho}=g^{f_\rho(0)},C_{N_\rho}^{'}=\mathscr{H}\left(att\left(N_\rho\right)\right)^{f_\rho(0)}\Big),\\
&\mathrm{CT}^{*}=\left(\mathscr{T}^{*},\tilde{C}={M_b}^{*}\cdot e(g,g)^{\alpha f_q(0)},C=g^{\beta f_q(0)}\right),\\
&\forall N_\eta\in Y(\mathscr{T}^{*})-Y(\mathscr{T}):\ C_{N_\eta}=g^{f_\eta(0)},C_{N_\eta}^{'}=\mathscr{H}\left(att\left(N_\eta\right)\right)^{f_\eta(0)};\\
&\forall N_\theta\in Y(\mathscr{T})-Y(\widehat{\mathscr{T}}):\ C_{N_\theta}=g^{f_\theta(0)},C_{N_\theta}^{'}=\mathscr{H}\left(att\left(N_\theta\right)\right)^{f_\theta(0)};\\
&\forall N_\rho\in Y(\widehat{\mathscr{T}}):\ C_{N_\rho}=g^{f_\rho(0)},C_{N_\rho}^{'}=\mathscr{H}\left(att\left(N_\rho\right)\right)^{f_\rho(0)}\Big).
\end{aligned} \quad (13)$$

CP-ABE sends the challenge ciphertext pair $\{\mathrm{CT}, \mathrm{CT}^{*}\}$ to simulator $\mathscr{B}$, returning it to adversary $\mathscr{A}$.

(iv) Query phase 2: Adversary $\mathscr{A}$ can continue to send more private key requests for other attribute sets to simulator $\mathscr{B}$. Similarly, it is required that all private key queries do not meet the challenge access structures $\mathscr{T}^{**}$ and $\mathscr{T}^{*}$. Simulator $\mathscr{B}$ responds to adversary $\mathscr{A}$ in the same way as in query phase 1.

(v) Guess: Adversary $\mathscr{A}$ outputs a guess $b'\in\{0,1\}$ about b. Simulator $\mathscr{B}$ outputs b' in its own security game, so the advantage of simulator $\mathscr{B}$ in winning the security game with CP-ABE is $Adv_{\mathscr{B}} = |\Pr[b' = b] - 1/2| = Adv_{\mathscr{A}}$.

Therefore, if probability polynomial-time adversary $\mathscr{A}$ can win the security game of this section with a non-negligible advantage $Adv_{\mathscr{A}}$, simulator $\mathscr{B}$ can breach

CP-ABE with the same advantage. In sum, the encryption scheme proposed in this paper is IND-CPA secure.

7. Performance Analysis

The test environment was a PC with an AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz, with 16 GB RAM, operating on 64 bit Windows 10 Home. The prototype system was developed in Java and run on Java Development Kit (JDK) 15. We relied on the Java Pairing-Based Cryptography (JPBC) library 1.2.0 to implement the bilinear operation. All results are the average of 20 runs.

7.1. Experiment for Rule 1. Rule 1 first compares the number of leaf nodes in the shared subpolicy. Then, we choose the root node of the shared subpolicy with a maximum number of leaf nodes as the cross-node of the integrated access tree. If the numbers of leaf nodes are the same, then the root node of the shared subpolicy with a maximum number of the AND gate children is selected as the cross-node of the integrated access tree.

Let $T(m_1)$ denote the computational cost of exponential operations on group G_1, and $T(z)$ denote the computational cost of exponential and multiplication operations on a big integer set. We ran a test measuring the time cost of exponential operations on group G_1, and exponential and multiplication operations on a big integer set, for 50 runs. The average time of the 50 runs was taken as our final test result, as shown in Table 2.

The test results show that the computational overhead of an exponential operation on a group G_1 is more than 15 times that of exponential and multiplication operations on the big integer set. During encryption, the computation on leaf nodes is exponential operations on group G_1, and there are two exponential operations for each leaf node. The assignment of a secret value to a child node of the AND gate node consists of exponential and multiplication operations on a big integer set. There is no exponential operation at the OR gate, and the secret value is directly propagated to it. Therefore, in Rule 1, the number of leaf nodes is given the highest priority. When the number of leaf nodes is the same, it is reasonable to further consider the number of the AND gate nodes.

7.2. Performance Analysis

7.2.1. Performance Analysis of Different Schemes. As shown in Table 3, the CP-ABE scheme does not consider the presence of the shared subpolicies. The literature [24, 25] can only deal with a single shared subpolicy. Our proposal can further deal with multiple shared subpolicies, and there is no restriction on the threshold of the root node of the access tree. Therefore, our scheme can handle access trees more flexibly than schemes proposed in the literature [24] does.

Let $T(m_i)$ denote the computational cost of exponential operations on group $G_i (i = 1, 2)$ and $T(pm_2)$ denote the computational cost of multiplication operations on group G_2. A denotes the attribute set of the data user. $Y(\mathcal{T})$ denotes the set of leaf nodes in all shared subpolicies. $Y(\widehat{\mathcal{T}})$ denotes the set of leaf nodes of the core shared subpolicy.

(1) First Case, the Root Nodes of All Access Trees are the "AND" ("*OR*") *Gates*. Assume a data owner has n data objects, and the access trees have shared subpolicies and need to be encrypted. Let $Y(\mathcal{T}_j), j = 1, \ldots, n$ be the set of leaf nodes of each access tree.

In the original CP-ABE scheme [5], each data object is encrypted separately, so the computation cost of $\widetilde{C}$ is $T(m_2) + T(pm_2)$, the computation cost of C is $T(m_1)$, and the total computation cost of C_{N_y} and C'_{N_y} is $2|Y(\mathcal{T}_j)|T(m_1)$. Therefore, the total computation cost for encrypting n data objects is $\sum_{j=1}^{n}[(2|Y(\mathcal{T}_j)| + 1)T(m_1) + T(m_2) + T(pm_2)]$. In [24, 25], encryption-related computations can be done only once for the leaf nodes on the core shared subpolicy. Compared with CP-ABE, the computation cost is reduced by $2(n-1)|Y(\widehat{\mathcal{T}})|T(m_1)$. In this scheme, the computational costs of $\widetilde{C}$ and C remain the same. However, for the leaf nodes of the shared subpolicies, all shared subpolicies are integrated under one node, and all computations can be done only once, so the computation cost is reduced by $2(n-1)|Y(\mathcal{T})|T(m_1)$ compared with CP-ABE. We can see the comparison in Table 4.

During encryption, the proposed scheme reduces $2(n-1)|Y(\mathcal{T})|T(m_1)$ of the time cost compared with the CP-ABE encryption. When the number of data objects n is given, the more leaf nodes contained in shared subpolicies (the larger the $|Y(\mathcal{T})|$ is), the more computational overhead that can be reduced during encryption. When the leaf nodes contained in shared subpolicies are given ($|Y(\mathcal{T})|$ is determined), the greater the number of data objects n, the more computational overhead that can be reduced during encryption. During encryption, the proposed scheme reduces $2(n-1)(|Y(\mathcal{T})| - |Y(\widehat{\mathcal{T}})|)T(m_1)$ of the time cost compared with the scheme in [24]. Because $Y(\mathcal{T}) \supseteq Y(\widehat{\mathcal{T}})$, the computational overhead of the encryption in this paper is less than that of the scheme in [24]. The encryption computational overhead of the scheme in [25] is the same as that of the scheme in [24].

(2) Second Case, Access Trees Have Different Root Nodes. Assume a data owner has $n + m$ data objects, and the access trees have shared subpolicies and need to be encrypted. Assume that in all access trees, n have the "AND" root nodes, and m have the "OR" root nodes. Let $Y(\mathcal{T}_j), j = 1, \ldots, n + m$ be the set of leaf nodes of each access tree.

In the original CP-ABE scheme [5], each data object is encrypted separately, so the computational cost of $\widetilde{C}$ is $T(m_2) + T(pm_2)$, the computational cost of C is $T(m_1)$, and the total computational cost of C_{N_y} and C'_{N_y} is $2|Y(\mathcal{T}_j)|T(m_1)$. Therefore, the total computational cost for encrypting $n + m$ data objects is $\sum_{j=1}^{n+m}[(2|Y(\mathcal{T}_j)| + 1)T(m_1) + T(m_2) + T(pm_2)]$. In [24, 25], encryption-related computations can be done only once for the leaf nodes on the core shared subpolicy. Compared with CP-ABE, the computation cost is reduced by $2(n + m - 1)|Y(\widehat{\mathcal{T}})|T(m_1)$.

Table 2: Time costs for basic computations involved in CP-ABE encryption.

Symbol	Description	Running time (ms)
$T(m_1)$	Exponential operations on group G_1	15.502
$T(z)$	Total time of exponential and multiplication operations on the integer set	0.969

Table 3: Comparison of different schemes.

Scheme	Single shared subpolicy	Multiple shared subpolicies	The threshold of the root node of the access tree
CP-ABE [5]	No	No	Arbitrary threshold
The literature [24]	Yes	No	Arbitrary threshold
The literature [25]	Yes	No	Arbitrary threshold
This paper	Yes	Yes	Arbitrary threshold

In this scheme, the computation costs of $\widetilde{C}$ and C remain the same, but encryption-related computations can be done only once for the leaf nodes on the core shared subpolicy. The encryption-related computations for leaf nodes on the noncore shared subpolicies can be done twice, so the computation cost is reduced by $2[(n-1)+(m-1)]|Y(\mathcal{T})|T(m_1)+2|Y(\widehat{\mathcal{T}})|T(m_1)$ compared with CP-ABE. We can see the comparison from Table 5.

During encryption, the proposed scheme reduces $2T(m_1)[(n+m-2)|Y(\mathcal{T})|+|Y(\widehat{\mathcal{T}})|]$ of the time cost compared with CP-ABE. When the number of data objects $n+m$ is given, the more leaf nodes contained in shared subpolicies (the larger the $|Y(\mathcal{T})|$ is), the more computational overhead that can be reduced during encryption. When $|Y(\mathcal{T})|$ and $|Y(\widehat{\mathcal{T}})|$ are given, the greater the number of data objects $n+m$, the more computational overhead that can be reduced during encryption. During encryption, the proposed scheme reduces $2(n+m-2)(|Y(\mathcal{T})|-|Y(\widehat{\mathcal{T}})|)T(m_1)$ of the time cost compared with the scheme in [24]. Because $Y(\mathcal{T}) \supseteq Y(\widehat{\mathcal{T}})$, the computational overhead of the encryption in this paper is less than that of the scheme in [24]. The encryption computational overhead of the scheme in [25] is the same as that of the scheme in [24].

7.2.2. Experimental Simulation.

(1) First Case, the Root Nodes of All Access Trees are the "AND" ("OR") *Gates*. When keeping the number of data objects n, the number of attributes $|Y(\mathcal{T}_j)|, j=1,\ldots,n$ in the access tree, and the number of attributes $|Y(\widehat{\mathcal{T}})|$ in the core shared subpolicy unchanged, with the increase of the number of attributes $|Y(\mathcal{T})|$ in the shared subpolicies, the computational overheads of the encryption in the four schemes are shown in Table 6. Figure 11 shows the trends of the computational overhead of the encryption in the four schemes with the increase of $|Y(\mathcal{T})|$. In this experiment, $n=4; |Y(\mathcal{T}_j)|=40, j=1,\ldots,n; |Y(\widehat{\mathcal{T}})|=3; |Y(\mathcal{T})|=\{5,10,15,20,25,30,35\}$.

It can be seen from Figure 11 that with the increase of the number of attributes in the shared subpolicies, the computation cost of CP-ABE, the literature [24] and the literature [25] is almost unchanged, while the computation cost of the encryption in this paper decreases. The computational overhead in this paper is the lowest of the four.

When keeping the number of attributes $|Y(\mathcal{T}_j)|, j=1,\ldots,n$ in the access tree, the number of attributes $|Y(\mathcal{T})|$ in the shared subpolicies, and the number of attributes $|Y(\widehat{\mathcal{T}})|$ in the core shared subpolicy unchanged, the computational overhead of the encryption in the four schemes changes with the increase of the number of data objects n, as shown in Table 7. Figure 12 shows the variation between the encryption computation cost of the four schemes and n. In this experiment, $|Y(\mathcal{T}_j)|=40$, $j=1,\ldots,n; |Y(\mathcal{T})|=35; |Y(\widehat{\mathcal{T}})|=10; n=4,\ldots,10$.

It can be seen from Figure 12 that the computational overhead of the encryption in the four schemes increases with the increase of the number of encrypted data objects. However, our scheme's encryption cost is always the lowest, and the growth rate of encryption cost is the lowest. That is, the more encrypted data objects, the higher the encryption efficiency of the scheme in this paper.

(2) Second Case, Access Trees Have Different Root Nodes. When keeping the number of data objects $n+m$, the number of attributes $|Y(\mathcal{T}_j)|, j=1,\ldots,n+m$ in the access tree, and the number of attributes $|Y(\widehat{\mathcal{T}})|$ in the core shared subpolicy unchanged, with the increase of the number of attributes $|Y(\mathcal{T})|$ in the shared subpolicies, the computational overheads of the encryption in the four schemes are shown in Table 8. Figure 13 shows the trends of the computational overhead of the encryption in the four schemes with the increase of $|Y(\mathcal{T})|$. In this experiment, $n=2, m=2; |Y(\mathcal{T}_j)|=40, j=1,\ldots,n+m; |Y(\widehat{\mathcal{T}})|=3; |Y(\mathcal{T})|=\{5,10,15,20,25,30,35\}$.

It can be seen from Figure 13 that with the increase of the number of attributes in the shared subpolicies, the encryption cost of CP-ABE, literature [24] and the literature [25] is almost unchanged, while the computation cost of the encryption in this paper decreases. The computational overhead in this paper is the lowest among the four.

When keeping the number of attributes $|Y(\mathcal{T}_j)|, j=1,\ldots,n+m$ in the access tree, the number of attributes $|Y(\mathcal{T})|$ in the shared subpolicies, and the number of attributes $|Y(\widehat{\mathcal{T}})|$ in the core shared subpolicy unchanged, the computational overhead of the encryption in the four schemes changes with the increase of the number of data objects $n+m$, as shown in Table 9. Figure 14 shows the variation between the encryption computation cost of the four schemes and $n+m$. In this experiment, $|Y(\mathcal{T}_j)|=40$, j

Table 4: Performance comparison of different schemes with the same shared subpolicies.

Name	CP-ABE [5]	The literature [24]	The literature [25]	This paper
PK size	$4\log p$	$4\log p$	$4\log p$	$4\log p$
MSK size	$2\log p$	$2\log p$	$2\log p$	$2\log p$
SK size	$(2\lvert A\rvert+1)\log p$	$(2\lvert A\rvert+1)\log p$	$(2\lvert A\rvert+1)\log p$	$(2\lvert A\rvert+1)\log p$
CT size	$2\sum_{j=1}^{n}(\lvert Y(\mathscr{T}_j)\rvert+1)\log p$	$2\sum_{j=1}^{n}(\lvert Y(\mathscr{T}_j)\rvert+1)\log p$	$2\sum_{j=1}^{n}(\lvert Y(\mathscr{T}_j)\rvert+1)\log p$	$2\sum_{j=1}^{n}(\lvert Y(\mathscr{T}_j)\rvert+1)\log p$
Encryption overhead	$\sum_{j=1}^{n}[(2\lvert Y(\mathscr{T}_j)\rvert+1)\cdot T(m_1)+T(m_2)+T(pm_2)]$	$\sum_{j=1}^{n}[(2\lvert Y(\mathscr{T}_j)\rvert+1)\cdot T(m_1)+T(m_2)+T(pm_2)]-2(n-1)\cdot\lvert Y(\widehat{\mathscr{T}})\rvert T(m_1)$	$\sum_{j=1}^{n}[(2\lvert Y(\mathscr{T}_j)\rvert+1)\cdot T(m_1)+T(m_2)+T(pm_2)]-2(n-1)\cdot\lvert Y(\widehat{\mathscr{T}})\rvert T(m_1)$	$\sum_{j=1}^{n}[(2\lvert Y(\mathscr{T}_j)\rvert+1)\cdot T(m_1)+T(m_2)+T(pm_2)]-2(n-1)\cdot\lvert Y(\mathscr{T})\rvert T(m_1)$

Table 5: Performance comparison of different schemes with the same shared subpolicies.

Name	CP-ABE [5]	The literature [24]	The literature [25]	This paper
PK size	$4\log p$	$4\log p$	$4\log p$	$4\log p$
MSK size	$2\log p$	$2\log p$	$2\log p$	$2\log p$
SK size	$(2\lvert A\rvert+1)\log p$	$(2\lvert A\rvert+1)\log p$	$(2\lvert A\rvert+1)\log p$	$(2\lvert A\rvert+1)\log p$
CT size	$2\sum_{j=1}^{n+m}(\lvert Y(\mathcal{T}_j)\rvert+1)\log p$	$2\sum_{j=1}^{n+m}(\lvert Y(\mathcal{T}_j)\rvert+1)\log p$	$2\sum_{j=1}^{n+m}(\lvert Y(\mathcal{T}_j)\rvert+1)\log p$	$2\sum_{j=1}^{n+m}(\lvert Y(\mathcal{T}_j)\rvert+1)\log p$
Encryption overhead	$\sum_{j=1}^{n+m}[(2\lvert Y(\mathcal{T}_j)\rvert+1)\cdot T(m_1)+T(m_2)+T(pm_2)]$	$\sum_{j=1}^{n+m}[(2\lvert Y(\mathcal{T}_j)\rvert+1)\cdot T(m_1)+T(m_2)+T(pm_2)]-2(n+m-1)\cdot\lvert Y(\widehat{\mathcal{T}})\rvert T(m_1)$	$\sum_{j=1}^{n+m}[(2\lvert Y(\mathcal{T}_j)\rvert+1)\cdot T(m_1)+T(m_2)+T(pm_2)]-2(n+m-1)\cdot\lvert Y(\widehat{\mathcal{T}})\rvert T(m_1)$	$\sum_{j=1}^{n+m}[(2\lvert Y(\mathcal{T}_j)\rvert+1)\cdot T(m_1)+T(m_2)+T(pm_2)]-2(n+m-2)\cdot\lvert Y(\mathcal{T})\rvert T(m_1)-2\lvert Y(\widehat{\mathcal{T}})\rvert T(m_1)$

Table 6: The encryption computational overhead of the four schemes (unit: s).

The number of attributes in the shared subpolicies	5	10	15	20	25	30	35
CP-ABE [5]	11.275	11.327	11.309	11.274	11.310	11.301	11.250
The literature [24]	10.985	11.032	11.061	10.979	11.014	11.001	10.961
The literature [25]	11.005	11.099	10.984	11.025	10.982	10.922	11.029
This paper	10.821	10.420	9.978	9.520	9.004	8.564	7.961

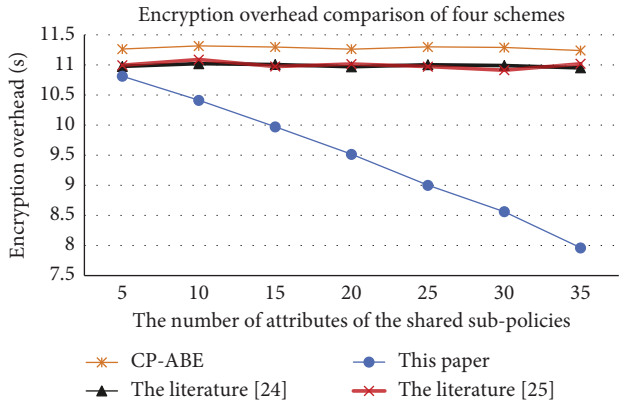

Figure 11: The trends of the computational cost of encryption of the four schemes with the number of attributes in the shared subpolicies.

Table 7: The encryption computational overhead of the four schemes (unit: s).

The number of data objects	4	5	6	7	8	9	10
CP-ABE [5]	11.190	13.452	16.462	19.378	21.781	24.717	27.096
The literature [24]	10.202	12.193	14.872	17.436	19.552	22.132	24.236
The literature [25]	10.270	12.151	14.965	17.453	19.605	22.110	24.251
This paper	7.797	9.118	10.969	12.730	14.126	15.827	17.447

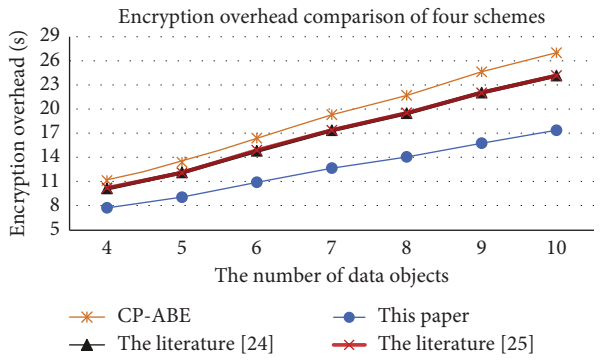

Figure 12: The relationship between the encryption computational cost and the number of data objects.

$= 1, \ldots, n+m$; $|Y(\mathcal{T})| = 35; |Y(\widehat{\mathcal{T}})| = 10; n+m = 4, \ldots, 10$.

It can be seen from Figure 14 that the computational overhead of the encryption in the four schemes increases with the increase of the number of encrypted data objects. However, our scheme's encryption cost is always the lowest, and the growth rate of encryption cost is the lowest. That is, the more encrypted data objects, the higher the encryption efficiency of this scheme in this paper.

When keeping the number of data objects $n+m$, the number of attributes $|Y(\mathcal{T}_j)|$, $j = 1, \ldots, n+m$ in the access tree, and the number of attributes $|Y(\mathcal{T})|$ in the shared subpolicies unchanged, with the increase of the number of attributes $|Y(\widehat{\mathcal{T}})|$ in the core shared subpolicy, the computational overheads of the encryption in the four schemes are shown in Table 10. Figure 15 shows the trends of the computational overhead of encryption in the four schemes with the increase of $|Y(\widehat{\mathcal{T}})|$. In this experiment, $n = 2, m = 2; |Y(\mathcal{T}_j)| = 40$, $j = 1, \ldots, n+m$; $|Y(\mathcal{T})| = 35; |Y(\widehat{\mathcal{T}})| = \{5, 10, 15, 20, 25, 30\}$.

It can be seen from Figure 15 that with the increase of the number of attributes in the core shared subpolicy, the encryption cost of CP-ABE is almost unchanged. In contrast, the computation cost of the encryption in the literature [24], the literature [25] and this paper decreases. The number of attributes in the core shared subpolicy is

Table 8: The encryption computational overhead of the four schemes (unit: s).

The number of attributes in the shared subpolicies	5	10	15	20	25	30	35
CP-ABE [5]	11.552	11.433	11.381	11.444	11.340	11.424	11.531
The literature [24]	11.261	11.133	11.075	11.134	11.043	11.122	11.219
The literature [25]	11.278	11.074	11.052	11.129	11.041	11.133	11.202
This paper	11.146	10.710	10.341	10.093	9.764	9.518	9.184

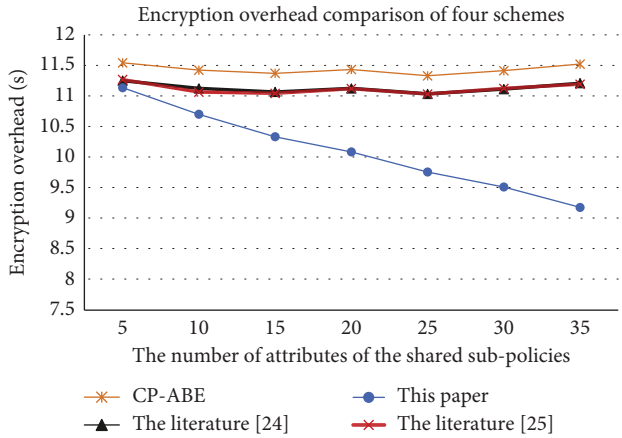

Figure 13: The relationship between the encryption computational cost and the number of attributes in the shared subpolicies.

Table 9: The encryption computational overhead of the four schemes (unit: s).

The number of data objects	4	5	6	7	8	9	10
CP-ABE [5]	11.242	13.322	16.490	19.424	21.655	24.719	27.046
The literature [24]	10.256	12.072	14.866	17.477	19.377	22.149	24.196
The literature [25]	10.277	12.036	14.909	17.413	19.340	22.055	24.173
This paper	8.652	9.754	11.684	13.545	14.660	16.667	17.968

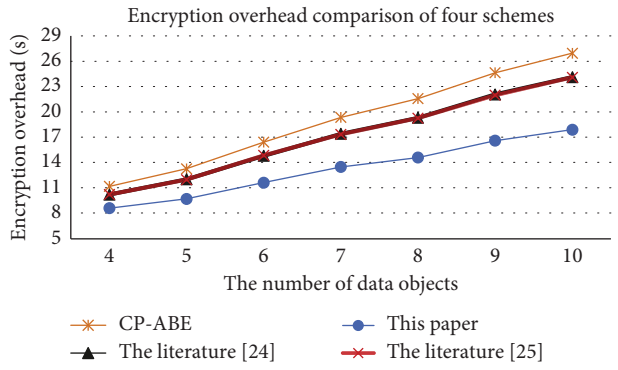

Figure 14: The relationship between the encryption computational cost and the number of data objects.

Table 10: The encryption computational overhead of the four schemes (unit: s).

The number of attributes in the core shared subpolicy	5	10	15	20	25	30
CP-ABE [5]	11.072	11.242	11.157	11.210	11.116	11.357
The literature [24]	10.568	10.256	9.689	9.306	8.694	8.454
The literature [25]	10.574	10.211	9.690	9.311	8.692	8.419
This paper	8.697	8.592	8.490	8.361	8.224	8.128

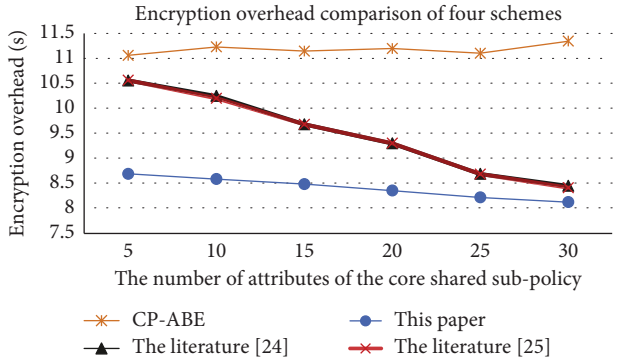

Figure 15: The relationship between the encryption computational cost and the number of attributes in the core shared subpolicy.

less than or equal to the number of attributes in the shared subpolicies. That is, when the number of attributes in the core shared subpolicy reaches the maximum, the computational overhead in the literature [24] and the literature [25] is at most equal to that in this paper. In other words, the computational overhead in this paper is the lowest among the four.

8. Conclusion

In the scenario where data owners share different data and the corresponding access control policies have multiple shared subpolicies, this study proposed an optimized CP-ABE scheme via the flexible integration of access trees with multiple shared subpolicies. Our integration method includes the modification of access trees, first round integration of access trees with the same root nodes, selection of the cross-node, and a last round of integration of access trees. Thus, we achieve an optimally integrated access tree from the viewpoint of encryption-related computation. Furthermore, security analysis showed that the proposed scheme is IND-CPA secure.

In our work, we did not consider optimally categorizing all access trees of a data owner for integration. Another challenge is integrating access trees, where the number of shared subpolicies between every two policies differs. Therefore, our future work will focus on how to globally categorize access trees of a data owner for optimal integration. Moreover, we will consider integrating access trees with a different number of shared subpolicies between every two policies.

Data Availability

The experimental data used to support the findings of this study are available from the first author upon reasonable request.

Conflicts of Interest

The authors declare that there are no conflicts of interest.

Acknowledgments

This research work was supported by National Natural Science Foundation of China (grant numbers: 61862059 and 61562085).

References

[1] D. Feng, M. Zhang, Y. Zhang, and Z. Xu, "Study on cloud computing security," *Journal of Software*, vol. 22, no. 1, pp. 71–83, 2011.

[2] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology - EUROCRYPT 2005, EUROCRYPT 2005*, R. Cramer, Ed., vol. 3494, Springer, Berlin, Germany, 2005.

[3] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in Cryptology. CRYPTO 1984*, G. R. Blakley and D. Chaum, Eds., vol. 196, Springer Berlin, Germany, 1984.

[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 ACM Conference on Computer and Communications Security*, pp. 89–98, ACM, Alexandria VA USA, October, 2006.

[5] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 321–334, IEEE Computer Society, Berkeley, CA, USA, May, 2007.

[6] M. Green, S. Hohenberger, and B. Waters, "Outsourcing the decryption of ABE ciphertexts," in *Proceedings of the 20th USENIX Conference on Security*, San Francisco CA, USA, August, 2011.

[7] Z. Zhou and D. Huang, "Efficient and secure data storage operations for mobile cloud computing," in *Proceedings of the 2012 8th International Conference on Network and Service Management(CNSM)*, pp. 37–45, IEEE, Las Vegas, NV, USA, October, 2012.

[8] M. Asim, M. Petkovic, and T. Ignatenko, "Attribute-based encryption with encryption and decryption outsourcing conference on innovations in clouds," in *12th Australian Information Security Management Conference*, Research Online, Perth, Australia, pp. 21–28, December 2014.

[9] S. Hohenberger and B. Waters, "Online/offline attribute based encryption," *International Conference on Practice and Theory in Public-Key Cryptography*, pp. 293–310, Springer, Berlin, Germany, 2014.

[10] S. Li and H. Zhang, "Online/offline attribute-based encryption with multi-authority access control," in *Proceedings of the 2021 18th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pp. 426–433, IEEE, Chengdu, China, December, 2021.

[11] L. Cao, Y. Liu, X. Dong, and X. Guo, "User privacy-preserving cloud storage scheme on CP-ABE," *Journal of Tsinghua University*, vol. 58, no. 2, pp. 150–156, 2018.

[12] L. Li, Z. Wang, and N. Li, "Access control scheme supporting computing outsourcing in fog computing," *Computer Engineering and Applications*, vol. 57, no. 6, pp. 81–87, 2021.

[13] Q. Leng and W. Luo, "Attribute-based encryption with outsourced encryption," *Communications Technology*, vol. 54, no. 9, pp. 2242–2246, 2021.

[14] W. Luo, C. Feng, L. Zou et al., "Attribute-based encryption scheme with fast encryption," *Journal of Software*, vol. 31, no. 12, pp. 3923–3936, 2020.

[15] S. Liu and Y. Guo, "Multi-authority based CP-ABE proxy re-encryption scheme for cloud computing," *Chinese Journal of Network and Information Security*, vol. 8, no. 3, pp. 176–188, 2022.

[16] Y. W. Hwang and I. Y. Lee, "A study on CP-ABE based data sharing system that provides signature-based verifiable outsourcing," in *Proceedings of the 2021 International Conference on Advanced Enterprise Information System (AEIS)*, pp. 1–5, IEEE, St. Petersburg, Russia, June, 2021.

[17] K. Emura, A. Miyaji, K. Omote, A. Nomura, and M. Soshi, "A ciphertext-policy attribute-based encryption scheme with constant ciphertext length," *International Journal of Applied Cryptography*, vol. 2, no. 1, pp. 46–59, 2010.

[18] J. Herranz, F. Laguillaumie, and C. Ràfols, "Constant size ciphertexts in threshold attribute-based encryption," *International Conference on Practice and Theory in Public Key Cryptography*, pp. 19–34, Springer, Berlin, Germarny, 2010.

[19] V. Odelu, A. K. Das, M. Khurram Khan, K. R. Choo, and M. Jo, "Expressive CP-ABE scheme for mobile devices in IoT satisfying constant-size keys and ciphertexts," *IEEE Access*, vol. 5, pp. 3273–3283, 2017.

[20] G. S. Tamizharasi, B. Balamurugan, and H. A. Gaffar, "Privacy preserving ciphertext policy attribute based encryption scheme with efficient and constant ciphertextsize," in *Proceedings of the International Conference on Inventive Computation Technologies*, pp. 1–5, IEEE, Coimbatore, India, August, 2016.

[21] W. Yang, R. Wang, Z. Guan, L. Wu, X. Du, and M. Guizani, "A lightweight attribute based encryption scheme with constant size ciphertext for Internet of Things," in *Proceedings of the ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, Dublin, Ireland, June, 2020.

[22] S. Wang, J. Zhou, J. Liu, J. Yu, J. Chen, and W. Xie, "An efficient file hierarchy attribute-based encryption scheme in cloud computing," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1265–1277, 2016.

[23] J. Li, N. Chen, and Y. Zhang, "Extended file hierarchy access control scheme with attribute-based encryption in cloud computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 2, pp. 983–993, 2021.

[24] W. Li, *Research of Attribute-Based Encryption Schemes Based on Shared Subpolicy in Cloud Computing*, Beijing University of Posts and Telecommunications, Beijing, China, 2019.

[25] J. Zhao, "Research on key technology of attribute-based encryption for cloud storage," *Information Engineering University*, 2022.

[26] K. Xue, N. Gai, J. Hong, D. Wei, P. Hong, and N. Yu, "Efficient and secure attribute-based access control with identical subpolicies frequently used in cloud storage," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 635–646, 2022.

[27] S. Fugkeaw and H. Sato, "Enabling dynamic and efficient data access control in cloud computing based on attribute certificate management and CP-ABE," in *Proceedings of the Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pp. 454–461, IEEE, Cambridge, UK, March, 2018.

[28] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.