Dynamic Policy Update for Ciphertext-Policy Attribute-Based Encryption

Wei Yuan *

Abstract. Ciphertext-policy attribute-based encryption (CP-ABE) is a promising access control technique for cloud storage. However, due to the absence of the update function, CP-ABE has not been widely accepted as a complete access control tool. In this paper, we add the update function for CP-ABE such that data access policy can be dynamically updated after the ciphertext is generated. First, we present a new linear secret sharing (LSS) matrix update algorithm based on existing LSS matrix generation algorithm. Then we summarize the common structure of some typical CP-ABE schemes and abstract a basic CP-ABE scheme from them. Next, based on the matrix update algorithm, we implement the policy update algorithm with the encryption algorithm of the basic CP-ABE scheme. In our scheme, data access policy can be directly changed without key update. If a user, whose attributes satisfy the old data access policy, does not decrypt old ciphertext before the policy update, he cannot obtain the data after the ciphertext is updated. As a result, the long-term problem "user can refuse to the update on his secret key if the policy update reduces his privilege" that hinders CP-ABE from being a practical network access control tool is overcome. Meanwhile, private channels to transmit update keys for non-revoked users are eliminated. The communication, computation, and storage costs for an update no longer depend on the number of users, but are relative to the number of attributes in the access policy.

Keywords: Access control, attribute based encryption, policy update

1 Introduction

Our Motivation. In cloud storage systems, data files can be stored on different platforms and geographic locations, and the management and usage for data are very convenient. This brings numerous advantages for commercial applications and prompts more and more organizations to migrate their valuable data from local storage systems to cloud storage systems. As a result, many security equipments to protect dispersive local servers are saved and the costs for organizations are greatly reduced. Meanwhile, since valuable data are stored on remote cloud servers, access control should be achieved by network and cloud servers become high value attacking targets.

* Wei Yuan is with State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China.

An intuitive solution to control data access privilege in cloud storage systems is to give full control privilege of data to cloud servers and increase security facilities, such as firewall and intrusion detection system, to protect them. However, this straightforward approach may incur uncontrollable risks for organizations: A hacker may intrude into the cloud storage system and bypasses the security facilities or the authentication mechanism utilizing system vulnerabilities. In addition, cloud servers may grant access privilege of organizations' data to unauthorized users or use these data by themselves. This threat cannot be defended with technique measures in advance. Thus, most organizations do not wish to grant full control privilege of their data to cloud service providers although data files are stored on them.

A promising access control technique to solve this problem is cryptographically enforced access control, CP-ABE [1]. Organizations may encrypt their data and set access polices for encrypted data with CP-ABE. Only the users with appropriate authorization can decrypt the encrypted data. However, a complete access control scheme should provide 4 functions: create, read, update, and delete. In CP-ABE, the encryption algorithm provides the function of create; the decryption algorithm provides the function of read; removing the ciphertext equals the function of delete; but no algorithm provides the function of update. That is to say, CP-ABE can only set an initial access policy for data files. When a user is dismissed or promoted, his access privileges on existing data files cannot be changed dynamically. For this reason, CP-ABE has not been regarded as a complete access control tool to control access policy of massive data. This problem motivates the studies of dynamic policy update on CP-ABE, which includes the functions of elevating or reducing the access privileges of certain kinds of users on certain data files.

Seemingly, run the encryption algorithm again under a new access policy is able to circumvent the problem of policy update. However, the encryption algorithm needs the original data as an input. If the re-encryption is executed by the cloud server, it surely knows the plaintext of the data. That is equivalent to give full control privilege to cloud servers and thus the cryptographically enforced access control is meaningless. Otherwise, if the re-encryption is executed by the data owner, the cloud server that stores the data files should transmit the encrypted files back to him. Then the data owner should decrypt and re-encrypt data files. It means that many valuable data will repeatedly appear in the owners' devices, and thus data owners also become high value attacking targets as cloud servers and each data owner should be protected with security equipments. This violates the original target of cloud storage in security area. Thus, transmitting data back to data owner is also not reasonable. As a result, re-encrypting the original data cannot circumvent the problem of policy update.

In local storage systems, access control can be enforced on each user. However, in cloud storage, there is not an enforcement mechanism to oblige users to accept operations on their private keys. This is the main difference between local access control and network access control. In cloud storage, if the policy update reduces/cancels a user's access privilege, he can refuse the update on his

secret key and continues to decrypt the encrypted files that he once had ability to decrypt. Thus, a policy update algorithm on CP-ABE cannot simply update the secret keys of the revoked users when it modifies data access policy.

To fit for the network environment of cloud storage, existing policy update methods on CP-ABE have no choice but to produce update keys for all the non-revoked users. This brings an inexpectant result: To change data access policy for once, the number of secret keys, which should be updated, depends on the number of non-revoked users in the cloud system but not the number of changed attributes of the access policy. We know that modern cloud storage systems store huge amount of data and serve millions of users at the same time while the number of attributes to control data access policy is very limited (Usually dozens of attributes are enough). This inherent problem greatly reduces the efficiency of existing policy update methods such that organizations are reason to doubt the feasibility of CP-ABE in practical systems.

In this paper, we propose a dynamic policy update method on CP-ABE to overcome the long-term problem that CP-ABE cannot dynamically change data access policy such that cryptographically enforced access control can be accepted as a complete access control tool.

Our Updating Principle. First of all, the principle of our policy update is different from previous methods. In existing policy update schemes, a CP-ABE scheme is viewed as a scale. The left tray places a decryption key and the right tray places a ciphertext. The balance state of the scale represents the decryption key matches the ciphertext. Suppose the scale is balance before an update. If the data owner only changes the ciphertext, the balance state is sure to be broken. To recover the balance state after the update on the old ciphertext, updating decryption keys is unavoidable. This is the philosophic essence of current policy update methods. However, in our observation, a CP-ABE scheme also can be regarded as a steelyard. The left side is a decryption key and the right side is a ciphertext. The balance state represents the decryption key matches the ciphertext. Regard the access policy as the tick mark on the weigh beam and each ciphertext component, which is viewed as a weight, is placed at the right position according to the access policy. When the data access policy changes, we may take off, reset, or recombine parts of ciphertext components according to the differences between new access policy and the original one. Then the steelyard will recover the balance state. This will bring an exciting result that the decryption key on the left side does not need to change at all. As a result, the owners of the associated decryption keys do not have the right to refuse the update and thus other users do not need to update their decryption keys for unrelated updates.

Our Technique. To implement the policy update on CP-ABE under above idea, we make following innovations.

First, we propose a matrix update algorithm, which is compatible with the existing matrix generation algorithm, to transform an old LSS [2] matrix to a

new one. By comparing the two matrices, we can find the attributes, whose corresponding vectors in the LSS matrix are changed when the access policy updates. We observe a fact that only the ciphertext components corresponding to these attributes need to be updated for changing data access policy.

Then, we summarize the common structure of some typical CP-ABE schemes and abstract a basic CP-ABE scheme from them. We propose a new policy update algorithm based on the matrix update algorithm and implement it with the encryption algorithm of the basic CP-ABE scheme. The policy update algorithm does not need original data or old ciphertext as an input. Data owner is able to execute it independently to output a group of update components in off-line manner and then cloud server can generate ciphertext corresponding to the new access policy with the update components and the old ciphertext. The distribution of the new ciphertext is the same as the distribution of the ciphertext generated by the encryption algorithm under same access policy. Moreover, the update components can be transmitted from the data owner to the cloud server *publicly*.

Eventually, we construct a CP-ABE scheme with the function of dynamic *policy update*, called PU-CP-ABE, and prove that PU-CP-ABE is secure if the basic CP-ABE scheme is secure. Performance analysis on PU-CP-ABE shows: Given the update components, the ciphertext update time is even less than the encryption time.

Paper Organization. The remainder of our paper is structured as follows. In Section 2, we discuss some related works. In Section 3, we give the algorithm model and security model of PU-CP-ABE. In Section 4, the LSS technique and LSS matrix generation method is introduced and our matrix update method is proposed. We then describe the details of our construction and and prove its security in Section 5 and 6. In Section 7, we give an example implemented on reference [3]. In Section 8, the performance analysis of our scheme is given. Finally, we make a conclusion in Section 9.

2 Related Works

Controlling user access privileges is considered to be an important application of public key encryption (PKE) [4], such as PKI [5] and IBE [6]. Traditional PKE schemes usually restrict user privileges with time or user ID list. ABE [7] was proposed to extend the function of IBE. In ABE schemes [1, 8], users are described with a group of attributes instead of ID. Logically, we should control user privileges with attributes as well. However, earlier ABE schemes also used time or ID list to control user privileges.

Pirretti, Traynor, Mcdaniel and Waters [9] suggested that each attribute include an expiry time, and the system should periodically update associated user secret keys. When certain attribute needs updating, the authority will stop issuing new key components for that attribute. Bethencourt, Sahai and Waters [1] believed that the expiry time should be an independent attribute such that

different expiry time could be given to different users. Since time changes automatically, the privileges corresponding to specific attributes change as well. Attrapadung and Imai [10] added user ID list into access policy such that some users can be manually excluded from authorized set even though their attributes satisfy the access policy. Lewko, Sahai and Waters [11] further constructed a revocation system with user ID list.

However, both the time based control method and the ID list based control method have potential problems: If the system controls user privileges with time, the expiry time should be defined when the user secret key is generated. When the credentials of a user change before the expiry time, the system cannot make any reflection on the changes. If the system controls privileges of individual user by adding ID list, the list should be determined when the ciphertext is generated. To change the ID list embedded into the ciphertext, a new ciphertext has to be re-generated. Regard the ID list as parts of access policy, this method is equivalent to re-generate a new ciphertext with original plaintext.

To fit for dynamic changes of user credentials, some policy update schemes try to directly change old ciphertext and update associated user secret keys. Although the detailed methods and techniques of these schemes are different, their principles are similar.

Sahai, Seyalioglu, and Waters [12] proposed a stateful user revocation scheme for ABE. This scheme expresses expiry time as an LSS matrix [2] and ensures that the distribution of updated ciphertext is the same as the distribution of the original one with the piecewise generation technique. Meanwhile, a revoked user cannot decrypt updated ciphertexts. Similarly, Yu, Wang, and Ren [13] proposed an attribute revocation scheme for CP-ABE with the proxy re-encryption technique. This scheme is based on Cheung and Newport's CP-ABE scheme [14], and only supports access policies consisting of AND gate. To update an attribute from the system, the authority changes public parameters and generates proxy re-keys to redefine involved components of the public key and master key. Then a proxy server updates ciphertexts and secret keys of non-revoked users utilizing the proxy re-keys. A common problem of these schemes is updating efficiency. The system authority should update secret keys for all the non-revoked users for an update.

Yang et al. [15] proposed an attribute update scheme based on a multi-authority CP-ABE scheme [16] and further constructed an ABE-based access control system [17]. Their schemes are able to add new attributes besides removing old attributes. This extends the function of the scheme in [12]. The data owner generates update keys and then submits the update keys to the cloud server. The cloud server is able to change data access policy with these keys. Each time the cloud server updates a ciphertext, a group of new update keys are needed. A problem of these schemes is that the distribution of the updated ciphertext is different from the distribution of the original one. After receiving a ciphertext, data recipients should try different parameters [17] or use different decryption algorithms [18] to decrypt the ciphertext. The decryption time is relative to the number of update times. This restricts policy update times

tremendously. An unfortunate matter for these schemes is that the security of the reference [15] is not strong enough. Hong, Xue, and Li [19] pointed out that the scheme in [15] cannot resistant collusion attacks. A revoked user can still decrypt subsequently encrypted data as well.

Generally speaking, above policy update methods suffer from some inherent problems: To change decryption privilege of encrypted data, user secret keys have to be updated. The update keys should be transmitted to cloud server or data recipients secretly while the channel is public. These contradictions are difficult to solve in practical applications. Furthermore, the number of secret update keys depends on the number of users but not the number of attributes. This is contrary to the original intention of ABE: Describe users with fuzzy attributes instead of exact user ID. As a result, an efficient policy update method to promote CP-ABE from theory to practice is an urgent demand for crypto community.

3 Policy Update for CP-ABE

3.1 Algorithms

A PU-CP-ABE scheme consists of 6 polynomial time algorithms: Setup, KeyGen, Encrypt, Decrypt, PolicyUpdate, and CiphertextUpdate. Setup and KeyGen are executed by the trusted authority of the system. Encrypt and PolicyUpdate are executed by data owners. CiphertextUpdate can be executed by a cloud server. Decrypt is executed by data receivers. These algorithms are defined as follows:

• **Setup**$(1^\lambda) \rightarrow$(PK,MK). Input a security parameter λ, the setup algorithm outputs a public key PK and a master key MK.

• **KeyGen**(PK,MK,$\mathcal{S}$)$\rightarrow$SK. Input PK, MK, and a set of attributes $\mathcal{S}$, the key generation algorithm outputs a user private key SK corresponding to $\mathcal{S}$.

• **Encrypt**(PK,M,SK_{owner},$\mathbb{A}$)$\rightarrow$CT. Input PK, plaintext M, the private key of the data owner, and an access policy $\mathbb{A}$, the encryption algorithm outputs a ciphertext CT corresponding to $\mathbb{A}$.

• **Decrypt**(CT,SK)$\rightarrow$M. Input PK, a ciphertext CT that contains an access policy $\mathbb{A}$, and a private key SK that contains a set of attributes $\mathcal{S}$, the decryption algorithm outputs a plaintext M if $\mathcal{S}$ satisfies $\mathbb{A}$. Otherwise, it outputs a $\perp$.

• **PolicyUpdate**(PK,SK_{owner},$\mathbb{A}$,$\mathbb{A}'$)$\rightarrow$UC. Input PK, the private key of the data owner, an old access policy $\mathbb{A}$ and a new access policy $\mathbb{A}'$, the policy update algorithm produces a group of update components UC corresponding to $\mathbb{A}'$.

• **CiphertextUpdate**(CT,UC)$\rightarrow$CT$'$. Input a ciphertext CT corresponding to the old policy, and the update components UC corresponding to a new access policy $\mathbb{A}'$, the ciphertext update algorithm outputs new ciphertext CT$'$ corresponding to $\mathbb{A}'$.

Correctness. Suppose the security parameter λ is large enough. For all (PK,MK)$\leftarrow$Setup(1^λ), plaintext M, $\mathrm{SK}_{owner} \leftarrow$KeyGen(PK,MK,$\mathcal{S}_{owner}$), and $\mathrm{SK}_{receiver} \leftarrow$KeyGen(PK,MK,$\mathcal{S}_{receiver}$), PU-CP-ABE satisfies the following two correctness conditions:

1.If CT$\leftarrow$Encrypt(PK,M,SK_{owner},$\mathbb{A}$) and $\mathcal{S}_{receiver}$ satisfies $\mathbb{A}$, Decrypt(CT, $\mathrm{SK}_{receiver}$) outputs M.

2.If CT′ ←CiphertextUpdate(CT,UC) where CT corresponds to $\mathbb{A}$, UC ← PolicyUpdate(PK, SK_{owner}, $\mathbb{A}$, $\mathbb{A}'$), and $\mathcal{S}_{receiver}$ satisfies $\mathbb{A}'$, Decrypt(CT′, $\mathrm{SK}_{receiver}$) outputs M.

3.2 Selective Security for PU-CP-ABE

The security game for PU-CP-ABE is described between a simulator and an adversary as follows:

• **Init**. The adversary declares the challenge access policy $\mathbb{A}^*$ and sends it to the simulator.

• **Setup**. The simulator sends the public key PK to the adversary.

• **Query phase 1**. The adversary adaptively asks for secret keys with attribute set $\mathcal{S}_1, \cdots, \mathcal{S}_{u_1}$. For each attribute set, the simulator returns corresponding secret key to the adversary. The restriction is that none of the queried sets satisfy $\mathbb{A}^*$.

• **Challenge**. The adversary submits two equal-length plaintexts (M_0, M_1) and an access policy $\mathbb{A}_0$ that is not satisfied by the attribute set queried in query phase 1 to the simulator. The simulator creates a private key that is not queried before, flips a random coin $b \in \{0, 1\}$, generates $\mathrm{CT}_0^{(b)}$ corresponding to M_b and $\mathbb{A}_0$ with Encrypt, and returns $\mathrm{CT}_0^{(b)}$ to the adversary. If $\mathbb{A}_0 = \mathbb{A}^*$, challenge ciphertext has been returned. Otherwise, the adversary continues to submit a series of access policies $\mathbb{A}_1, \cdots, \mathbb{A}_n$ to the simulator that some $\mathbb{A}_j = \mathbb{A}^*$. The restriction is that each access policy cannot be satisfied by the attribute set queried before. For each access policy $\mathbb{A}_i$, the simulator returns UC_{i-1} generated by PolicyUpdate and let the adversary compute $\mathrm{CT}_i^{(b)}$ with CiphertextUpdate according to UC_{i-1} and $\mathrm{CT}_{i-1}^{(b)}$. Finally, the adversary obtains ($\mathrm{CT}_0^{(b)}, \cdots, \mathrm{CT}_n^{(b)}$).

• **Query phase 2**. The adversary asks for more secret keys with attribute set $\mathcal{S}_{u_1+1} \cdots, \mathcal{S}_u$. The restriction is that none of the queried sets satisfy the access policies submitted in challenge phase.

• **Guess**. The adversary outputs a guess b' for b.

Definition 1. Selective CPA Security. A PU-CP-ABE scheme is selectively secure against chosen-plaintext attacks if all polynomial time adversaries have at most a negligible advantage in the above game. The advantage of an adversary is defined as $\Pr[b' = b]\text{-}\frac{1}{2}$.

4 Matrix Update from Linear Secret Sharing

In this section, we recall the properties of LSS and how to generate an LSS matrix from an access policy. Moreover, our new matrix update algorithm is proposed.

4.1 Linear Secret Sharing

Definition 2. Linear Secret Sharing. A linear secret sharing scheme Π over a set of parties $\mathcal{P}$ is linear over $\mathbb{Z}_p$ if

1. The shares for each party form a vector over $\mathbb{Z}_p$.

2. There exists a share-generating matrix W with l rows and n columns. A function ρ labels each row W_i of W to a party. Considering a vector $\overrightarrow{v} = (s, y_2, y_3, \cdots, y_n)$, where $s \in \mathbb{Z}_p$ is the secret to be shared and $y_2, y_3, \cdots, y_n \in \mathbb{Z}_p$ are chosen randomly, then $\overrightarrow{v}W$ is the vector of l shares of the secret s according to Π. Here, $(\overrightarrow{v}W)_i$ belongs to party $\rho(i)$.

It has been shown in [2] that an LSS scheme defined as above has the linear reconstruction property: Suppose that Π is a linear secret sharing scheme for an access structure $\mathbb{A}$. Let $\mathcal{S}$ be an authorized set, and $I \subseteq \{1, 2, \cdots, l\}$ be defined as $I = \{i : \rho(i) \in \mathcal{S}\}$. Then, there exist constants $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ such that, if $\{\lambda_i = \overrightarrow{v}\overrightarrow{W}_i\}_{i \in I}$ are valid shares of any secret s according to W, $\sum_{i \in I} \omega_i \lambda_i = s$. Furthermore, these constants can be found in polynomial time in the size of the matrix W.

4.2 Matrix Generation

An access policy can be expressed in the form of access structure [12] in a direct way, e.g. (A *or* B) *and* C. This form of access policy also can be phased as a binary tree. The leaf nodes are attributes and non-leaf nodes are connectors "*and*" and "*or*".

Based on the definition of linear secret sharing, if $\mathcal{S}$ is an authorized set, and $I \subseteq \{1, 2, \cdots, l\}$ is defined as $I = \{i : \rho(i) \in \mathcal{S}\}$, we can find a group of constants $\{\omega_i\}_{i \in I}$ in polynomial time such that $\sum_{i \in I} \omega_i \overrightarrow{W}_i = (1, 0, \cdots, 0)$. Then we have $\sum_{i \in I} \omega_i \lambda_i = s$ since $\{\lambda_i = \overrightarrow{v}\overrightarrow{W}_i\}_{i \in I}$. As a result, $(1, 0, \cdots, 0)$ should be the sharing vector corresponding to the root of the binary tree. An LSS matrix is generated from the binary tree with the following algorithm [12, 16]:

The algorithm begins by labeling the root node of the tree with the vector $\overrightarrow{v}$=(1). The vectors of the other nodes are determined by their parent nodes. Go down the levels of the tree.

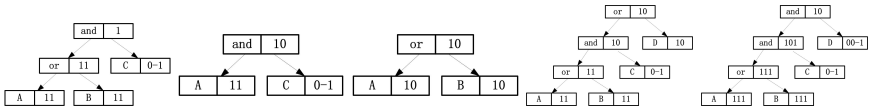
Suppose the current node is labeled with a vector $\overrightarrow{v}$. If the node is "*or*", we label its two children with $\overrightarrow{v}$. If the node is "*and*", we label its left child with $\overrightarrow{v}|1$ and its right child with $(0, \cdots, 0|\text{-}1)$ such that the two vectors sum to $\overrightarrow{v}|0$. | denotes concatenation. Once all the leaf nodes are labeled with vectors, the algorithm terminates. The vectors corresponding to the leaf nodes consist of an LSS matrix. The length of the matrix is determined by the longest vector. Meanwhile, other shorter vectors are padded with 0s.

Let $(W_{l \times n}, \rho)$ be an LSS matrix generated by above algorithm and $\{\omega_i\}_{i \in I}$ are a group of constants for an authorized set $\mathcal{S}$ that I is defined as above. Obviously, $\{W_{i,j}\}_{i \in [1,l], j \in [1,n]} \in \{1, 0, -1\}$ and $\omega_i \in \{0, 1\}$. $\omega_i = 0$ represents attribute $\rho(i) \in \mathcal{S}$ is unused while $\omega_i = 1$ represents attribute $\rho(i) \in \mathcal{S}$ is used for satisfying the LSS matrix.

For example, a policy $\mathbb{A}$=((A *or* B) *and* C) can be described in the form of the binary tree as shown in Fig.1(a). The LSS matrix W associated with the attributes A, B, and C can be obtained as follows:

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & -1 \end{bmatrix} \begin{matrix} \cdots A \\ \cdots B \\ \cdots C \end{matrix}$$

Since an attribute set $\mathcal{S} \supseteq \{A, C\}$ (or $\mathcal{S} \supseteq \{B, C\}$) satisfies $\mathbb{A}$, we can find a vector $\overrightarrow{\omega} = (1, 0, 1)$ (or $\overrightarrow{\omega} = (0, 1, 1)$) such that $\sum_{i \in I} \overrightarrow{\omega} \overrightarrow{W}_i$ =(1,0,0), where $I = \{i : \rho(i) \in \mathcal{S}\}$.



(a) $\mathbb{A}$=(A *or* B) *and* C (b) $\mathbb{A}_2$=A *and* C (c) $\mathbb{A}_3$=A *or* B (d) $\mathbb{A}_4$ = ((A *and* B) *and* C) *or* D (e) $\mathbb{A}_5$ = ((A *and* B) *and* C) *and* D

Fig. 1. Access Policies and Corresponding Binary Trees

A fact should be noticed: The vectors corresponding to the non-leaf nodes "*and*" and "*or*" are not included in the LSS matrix. However, these vectors can be recovered by the vectors of the leaf nodes. The recursive recovery process is as follows:

If the connector is "*and*", the vector associated with it equals the sum of its two children. If the connector is "*or*", the vector associated with it equals one of its child.

4.3 Matrix Update

In this section, we present a matrix update method to directly change an LSS matrix to a new one. The goal of the update is that the LSS matrix outputted by the matrix update method also shares the vector (1,$\cdots$,0) as the matrix outputted by the matrix generation method. There are 4 possible operations for updating a policy:

1. Delete an attribute connected by "*or*", e.g.(A *or* B)$\rightarrow$ A.
2. Delete an attribute connected by "*and*", e.g.(A *and* B)$\rightarrow$ A.
3. Add an attribute connected by "*or*", e.g.A $\rightarrow$ (A *or* B).
4. Add an attribute connected by "*and*", e.g.A $\rightarrow$ (A *and* B).

Since a binary tree is generated from top to bottom in the matrix generation algorithm, none of the vectors associated with the higher level nodes will be affected when we change an attribute from a policy. Only the vectors associated with the sibling and the children of the sibling of the changed attribute may be affected. Thus, our matrix update only needs to deal with the vectors associated with these nodes.

As the first step, we update the vector associated with the sibling of the changed attribute.

For case 1), since "*or*" grants the same vectors to its two children in the matrix generation algorithm, deleting an attribute connected by "*or*" does not affect the other one. We are able to directly remove the vector associated with that attribute when it is connected by "*or*".

For case 2), when we delete an attribute connected by "*and*", the vector associated with that attribute should be added to that of its sibling besides removing the vector associated with the deleted attribute.

For case 3), to add an attribute connected by "*or*", the vector associated with its sibling is added to the vector associated with the new attribute. The sibling of the new attribute may be a connector. Its associated vector can be recovered with the vectors of its children as in Section 4.2.

For case 4), to add an attribute connected by "*and*", the vector associated with the new attribute is set to (0,$\cdots$,0|-1). Then (0,$\cdots$,0|1) is added to its sibling. The length of 0s equals the longest vector.

As the second step, we continue to update the vectors associated with the children of the sibling of the changed attribute. The operations of this step are relevant to the operations on the vector associated with the sibling of the changed attribute. If the vector associated with the sibling is not changed, the vectors associated with the children of the sibling do not change as well. Otherwise, if a vector $\overrightarrow{v}$ is added on the vector associated with the sibling, we recursively deal with the vectors associated with the children of the sibling as follows:

Set the sibling as the current node at the beginning. If the current node is "*and*", adding $\overrightarrow{v}$ to one of its child. If the current node is "*or*", adding $\overrightarrow{v}$ to its two children. If the current node is an attribute, adding $\overrightarrow{v}$ to it.

Finally, the new LSS matrix associated with the new policy is generated. We can see clearly from the matrix: Which vectors are changed when we add/delete an attribute to/from the current policy. The attributes corresponding to these vectors can be recorded.

Let $(\mathbb{A} = W_{l\times n}, \rho)$ be an LSS matrix generated by the matrix generation algorithm and $(\mathbb{A}' = W_{l'\times n'}, \rho')$ be the LSS matrix outputted by the matrix update algorithm. Fig.1(b)-(e) show the new binary trees corresponding to the 4 cases of changing from $\mathbb{A}$, and the following LSS matrices show the relationship between the new policies and $\mathbb{A}$.

$$\mathbb{A}_2 = \begin{bmatrix} W'_A = W_A \\ W'_B \ removed \\ W'_C = W_C \end{bmatrix} \quad \mathbb{A}_3 = \begin{bmatrix} W'_A = W_A + W_C \\ W'_B = W_B + W_C \\ W'_C \quad removed \end{bmatrix}$$

$$\mathbb{A}_4 = \begin{bmatrix} W'_A = W_A \\ W'_B = W_B \\ W'_C = W_C \\ W'_D = W_{and} \end{bmatrix} \quad \mathbb{A}_5 = \begin{bmatrix} W'_A = W_A - W_D \\ W'_B = W_B - W_D \\ W'_C = W_C \\ W'_D = W_D \end{bmatrix}$$

Let a set $N(\mathbb{A}, C)$ denote the attributes, whose corresponding vectors are changed when deleting an attribute C from $\mathbb{A}$ and let a set $N(\mathbb{A}, D)$ denote the

attributes, whose corresponding vectors are changed when adding an attribute D to $\mathbb{A}$. Viewing from $\mathbb{A}_3$ and $\mathbb{A}_5$, $N(\mathbb{A}, C) = (A, B)$ and $N(\mathbb{A}, D) = (A, B)$.

5 Constructions of PU-CP-ABE

In this section, we first introduce a basic CP-ABE scheme abstracted from some existing schemes and then construct PU-CP-ABE based on the basic CP-ABE scheme.

5.1 Basic CP-ABE Scheme

When we update the access policy of a ciphertext, we should generate a ciphertext at first. Thus, the policy update algorithm should base on an encryption scheme. Logically, we should propose a new CP-ABE scheme or use an existing CP-ABE scheme as the base of our policy update.

To maximize the application range of our policy update, we first recall some typical CP-ABE schemes [1, 3, 20] and then abstract a basic encryption scheme from them. As a result, the policy update on the basic scheme can be applied on such existing schemes. In addition, many other CP-ABE schemes [10, 16, 21–24] have same structure. That is to say, our policy update is suitable for all these schemes.

The CP-ABE schemes of RW13 [3], Waters11 [20].scheme1, and BSW07 [1] are list as follows. We describe them with unified symbols. All these schemes include 4 algorithms: Setup, KeyGen, Encrypt, and Decrypt.

RW13 [3]:
Setup(1^λ) $\rightarrow$Generate PK=$(\mathbb{G}, \mathbb{G}_T, e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T, g, u, , w, v, e(g,g)^\alpha)$ and MK=(α).
Encrypt(PK,M,$\mathbb{A} = (W_{l\times n}, \rho)$)$\rightarrow$Select l random numbers $t_1, \cdots, t_l \in \mathbb{Z}_p$ and a random vector $\overrightarrow{v} = (s, y_2, y_3, \cdots, y_n) \in \mathbb{Z}_p^n$, compute $\lambda_i = \overrightarrow{v}\overrightarrow{W}_i$, and embeds it into $C_{1,i}$ for $i \in [1, l]$. Finally, CT=$(\mathbb{A}, C = Me(g,g)^{\alpha s}, C_0 = g^s, i \in [1,l], \{C_{1,i} = w^{\lambda_i} v^{t_i}, C_{2,i} = (u^{\rho(i)}h)^{-t_i}, C_{3,i} = g^{t_i}\})$.
KeyGen(PK,MK,$\mathcal{S}$)$\rightarrow$Select a random number $r \in \mathbb{Z}_p$ and random number $r_j \in \mathbb{Z}_p$ for each attribute $j \in \mathcal{S}$ and compute Key=$(\mathcal{S}, K_0 = g^\alpha w^r, K_1 = g^r, j \in \mathcal{S}, \{K_{2,j} = g^{r_j}, K_{3,j} = (u^j h)^{r_j} v^{-r}\})$.
Decrypt(CT,SK): Define $I \subseteq \{1, 2, \cdots, l\}$ and $I = \{i : \rho(i) \in \mathcal{S}\}$. There exists constants $\{\omega_i\}_{i\in I}$ such that $\sum_{i\in I} \omega_i \overrightarrow{W}_i = (1, 0, \cdots, 0)$. Compute $\mathbf{B_i} = \mathbf{e(C_{1,i}, K_1)e(C_{2,i}, K_{2,\rho(i)})e(C_{3,i}, K_{3,\rho(i)})} = e(g,w)^{r\lambda_i}$, $\mathbf{B} = \prod_{\mathbf{i}\in\mathbf{I}}(\mathbf{B_i})^{\omega_\mathbf{i}} = e(g,r)^{rs}$, $M = BC/e(C_0, K_0)$.

Water11 [20].scheme1:
Setup(1^λ) $\rightarrow$Generate PK=($\mathbb{G}, \mathbb{G}_T, e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T, g, g^a, e(g,g)^\alpha, h_1, \cdots, h_U$) and MK=($g^\alpha$).
Encrypt(PK,M,$\mathbb{A} = (W_{l\times n}, \rho)$)$\rightarrow$Select Select l random numbers $t_1, \cdots, t_l \in \mathbb{Z}_p$ and a random vector $\overrightarrow{v} = (s, y_2, y_3, \cdots, y_n)$, compute $\lambda_i = \overrightarrow{v}\overrightarrow{W}_i$, and embeds it into $C_{1,i}$ for $i\in[1,l]$. Finally, CT=($\mathbb{A}$, $C = Me(g,g)^{\alpha s}$, $C_0 = g^s$, $i\in[1,l]$, $\{C_{1,i} = g^{a\lambda_i} h_{\rho(i)}^{-t_i}$, $C_{2,i} = g^{t_i}\}$).
KeyGen(PK,MK,$\mathcal{S}$)$\rightarrow$Select a random number $r\in\mathbb{Z}_p$ and compute Key=($\mathcal{S}$, $K_0 = g^\alpha g^{ar}$, $K_1 = g^r$, $j\in\mathcal{S}$, $\{K_{2,j} = h_j^r\}$).
Decrypt(CT,SK): Define $I \subseteq \{1, 2, \cdots, l\}$ and $I = \{i : \rho(i) \in \mathcal{S}\}$. There exists constants $\{\omega_i\}_{i\in I}$ such that $\sum_{i\in I} \omega_i \overrightarrow{W}_i = (1, 0, \cdots, 0)$. Compute $\mathbf{B_i} = \mathbf{e(C_{1,i}, K_1)e(C_{2,i}, K_{2,\rho(i)})} = e(g,g)^{ar\lambda_i}$, $\mathbf{B} = \prod_{\mathbf{i}\in\mathbf{I}}(\mathbf{B_i})^{\omega_\mathbf{i}} = e(g,g)^{ars}$, $M = BC/e(C_0, K_0)$.

BSW07 [1](The parameter β associated to key delegation is ignored):
Setup(1^λ) $\rightarrow$Generate PK=($\mathbb{G}, \mathbb{G}_T, e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T, g, e(g,g)^\alpha, H : \{0,1\}^* \rightarrow \mathbb{G}$) and MK=($g^\alpha$).
Encrypt(PK,M,$\mathbb{A} = (W_{l\times n}, \rho)$)$\rightarrow$Select a random vector $\overrightarrow{v} = (s, y_2, y_3, \cdots, y_n)$, compute $\lambda_i = \overrightarrow{v}\overrightarrow{W}_i$, and embeds it into $C_{1,i}$ for $i\in[1,l]$. Finally,CT=($\mathbb{A}$, $C = Me(g,g)^{\alpha s}$, $C_0 = g^s$, $i\in[1,l]$, $\{C_{1,i} = g^{\lambda_i}$, $C_{2,i} = H(i)^{\lambda_i}\}$).
Key(PK,MK,$\mathcal{S}$)$\rightarrow$Select a random number $r \in \mathbb{Z}_p$ and random number $r_j \in \mathbb{Z}_p$ for each attribute $j\in\mathcal{S}$ and compute Key=($\mathcal{S}$, $K_0 = g^{(\alpha+r)}$, $j\in\mathcal{S}$, $\{K_{1,i} = g^r H(j)^{-r_j}$, $K_{2,j} = g^{r_j}\}$).
Decrypt(CT,SK): Define $I \subseteq \{1, 2, \cdots, l\}$ and $I = \{i : \rho(i) \in \mathcal{S}\}$. There exists constants $\{\omega_i\}_{i\in I}$ such that $\sum_{i\in I} \omega_i \overrightarrow{W}_i = (1, 0, \cdots, 0)$. Compute $\mathbf{B_i} = \mathbf{e(C_{1,i}, K_{1,\rho(i)})e(C_{2,i}, K_{2,\rho(i)})} = e(g,g)^{r\lambda_i}$, $\mathbf{B} = \prod_{\mathbf{i}\in\mathbf{I}}(\mathbf{B_i})^{\omega_\mathbf{i}} = e(g,g)^{rs}$, $M = BC/e(C_0, K_0)$.

Summarized from above schemes, in the encryption algorithms, the components of a ciphertext can be divided into two kinds:

1. Common ciphertext components, C and C_0, corresponding to the plaintext.

2. Ciphertext components, $C_{1,i}, \cdots, C_{m,i}$, corresponding to an attribute $\rho(i)$. m=3 in [3] and m=2 in [1] and [20].scheme1.

The common ciphertext components mask the plaintext with a secret s, and s is divided into a series of shares $\lambda_i = \overrightarrow{v}\overrightarrow{W}_i$. Then given the share λ_i associated to the attribute $\rho(i)$, a group of ciphertext components, $C_{1,i}, \cdots, C_{m,i}$, can be computed.

In the decryption algorithms, B_i is defined as $e(C_{1,i}, K_{1,\rho(i)}) \cdot \cdots \cdot e(C_{m,i}, K_{m,\rho(i)})$ and B is defined as $\prod_{i\in I}(B_i)^{\omega_i}$. Ignoring imperceptible distinctions[1] of these schemes, we have $B_i = A^{\lambda_i}$ and then $B = A^{\overrightarrow{v}\sum_{i\in I}\overrightarrow{W}_i\omega_i} = A^s$. As a result, $M = BC/e(C_0, K_0)$ can be obtained. Finally, we can abstract the basic CP-ABE scheme from [1, 3, 20] as below.

[1] A equals $e(g,g)^r$, $e(g,w)^r$ or $e(g,g)^{ar}$ respectively in [1, 3, 20].

Our basic CP-ABE from [1, 3, 20]:
Setup(1^λ) →Generate PK and MK.
Encrypt(PK, M, $\mathbb{A} = (W_{l\times n}, \rho)$) → Select a random vector $\overrightarrow{v} = (s, y_2, y_3, \cdots, y_n)$, compute $\lambda_i = \overrightarrow{v}\overrightarrow{W_i}$, and embeds it into $C_{1,i}$ for $i\in[1, l]$. Finally, CT=($\mathbb{A}$, C, C_0, $i\in[1, l]$, $\{C_{1,i}, \cdots, C_{m,i}\}$).
Key(PK,MK,$\mathcal{S}$) → Select random numbers and compute Key=($\mathcal{S}$, K_0, $j\in\mathcal{S}$, $\{K_{1,j}, \cdots, K_{m,j}\}$).
Decrypt(CT,SK): Define $I \subseteq \{1, 2, \cdots, l\}$ and $I = \{i : \rho(i) \in \mathcal{S}\}$. There exists constants $\{\omega_i\}_{i\in I}$ such that $\sum_{i\in I}\omega_i\overrightarrow{W_i} = (1, 0, \cdots, 0)$. Compute $B_i = e(C_{1,i}, K_{1,\rho(i)})\cdots e(C_{m,i}, K_{m,\rho(i)}) = A^{\lambda_i}$, $B = \prod_{i\in I}(B_i)^{\omega_i} = A^s$, $M = BC/e(C_0, K_0)$.

5.2 Construction of PU-CP-ABE

We call the basic CP-ABE scheme BASE (Setup, KeyGen, Encrypt, Decrypt), and the PU-CP-ABE scheme is constructed as follows.

Setup. First run BASE.Setup to generate PK and MK. Then add a pseudorandom function (PRF) generator $\mathcal{G}$ into PK.

KeyGen. Given PK, MK, and an attribute set $\mathcal{S}$, first run BASE.KeyGen to generate K_0 and $\{K_{1,j}, \cdots, K_{m,j}\}$ for attribute $j\in\mathcal{S}$. Then randomly select a pseudorandom function f from $\mathcal{G}$. SK=($\mathcal{S}$, f, K_0, $j\in\mathcal{S}$, $\{K_{1,j}, \cdots, K_{m,j}\}$).

Encrypt. Given PK, M, SK_{owner}, and an access policy $\mathbb{A} = (W_{l\times n}, \rho)$, run BASE.Encrypt and output CT. The difference is that $s, y_2, y_3, \cdots, y_n$ are not disorderly generated by pseudorandom generator but orderly selected from $f(x)$. x is a special attribute, e.g. ID, in $\mathcal{S}_{owner}$ and can be connected by a message index for multiply plaintexts of a user.

Decrypt. Given CT and SK, run BASE.Decrypt to recover M.

PolicyUpdate. Given PK, SK_{owner}, an old access policy $\mathbb{A}$, and a new access policy $\mathbb{A}'$, this algorithm produces update components UC corresponding to $\mathbb{A}'$.

First, compute $\overrightarrow{v} = (s, y_2, y_3, \cdots, y_n)$ with f_{owner} and $\mathcal{S}_{owner}$, and then recover $\lambda_i = \overrightarrow{v}\overrightarrow{W_i}$ for $i\in[1, l]$ with $\overrightarrow{v}$. Next, decompose the process of transforming $\mathbb{A}$ to $\mathbb{A}'$ to a series of atomic updates. Each atomic update only changes one attribute. The operations for an atomic update are as follows:

Case 1): If $\mathbb{A}$=($\mathbb{A}'$ *or* A), to delete A from $\mathbb{A}$, ciphertext components ($C_{1,A}$, $\cdots$, $C_{m,A}$) are removed. Correspondingly, an operation label "*delete*" is attached to the attribute A and λ_A is set to *null*.

Case 2): If $\mathbb{A}$=($\mathbb{A}'$ *and* A), to delete A from $\mathbb{A}$, ciphertext components $(C_{1,A}, \cdots, C_{m,A})$ are removed.

Let $N(\mathbb{A}, A)$ denote the attributes, whose associated vectors need to be changed when deleting attribute A from $\mathbb{A}$. $N(\mathbb{A}, A)$ can be obtained by the matrix update method in Section 4.3. For each attribute $i\in N(\mathbb{A}, A)$, λ_i is set to $\lambda_i + \lambda_A$ and corresponding ciphertext components $(C_{1,i}, \cdots, C_{m,i})$ are updated as below:

1. Call BASE.Encrypt to generate new $(C'_{1,i}, \cdots, C'_{m,i})$. The difference is that λ_i is replaced by λ_A.

2. For $j \in [1, m]$, compute $C_{j,i} = C'_{j,i} \times C_{j,i}$.

Correspondingly, an operation label "*multiply*" is attached to each attribute i. At last, an operation label "*delete*" is attached to A and λ_A is set to *null*.

Case 3): If $\mathbb{A}'$=($\mathbb{A}$ *or* A), to add A to $\mathbb{A}$, $\lambda_{\mathbb{A}}$ can be computed with the children of $\mathbb{A}$. The computing method is in the bottom of Section.4.2. Define $\lambda_A = \lambda_{\mathbb{A}}$. New ciphertext components $(C_{1,A}, \cdots, C_{m,A})$ are generated with BASE.Encrypt. Correspondingly, an operation label "*add*" is attached to the attribute A.

Case 4): If $\mathbb{A}'$=($\mathbb{A}$ *and* A), to add A to $\mathbb{A}$, a random number $y_A \in \mathbb{Z}_p$ is obtained orderly with f_{owner} and $\mathcal{S}_{owner}$. Set $\lambda_A = -y_A$. New ciphertext components $(C_{1,A}, \cdots, C_{m,A})$ are generated with BASE.Encrypt according to λ_A. Correspondingly, an operation label "*add*" is attached to the attribute A.

Let $N(\mathbb{A}, A)$ denote the attributes, whose associated vectors need to be changed when adding attribute A to $\mathbb{A}$. $N(\mathbb{A}, A)$ can be obtained by the matrix update method in Section 4.3. For each attribute $i \in N(\mathbb{A}, A)$, λ_i is set to $\lambda_i + y_A$ and corresponding ciphertext components $(C_{1,i}, \cdots, C_{m,i})$ are updated as below:

1. Call BASE.Encrypt to generate new $(C'_{1,i}, \cdots, C'_{m,i})$. The difference is that λ_i is replaced by y_A.

2. For $j \in [1, m]$, compute $C_{j,i} = C'_{j,i} \times C_{j,i}$.

Correspondingly, an operation label "*multiply*" is attached to each attribute i.

Since a group of ciphertext components may be changed continually in multiply atomic updates, we define a state transition diagram to deal with corresponding labels of the atomic updates:

First, we already have 3 operation labels: "*delete*", "*add*", and "*multiply*". Then we define 7 states: *Start*, *Delete*, *Add*, *Multiply*, *Add*$'$, *Multiply*$'$, and *Replace*. The state transition diagram is shown in Fig.2.

The attributes corresponding to all the groups of ciphertext components are in the *Start* state at the beginning. If an attribute is attached to an operation "*delete*", "*multiply*", or "*add*", its state is transferred to corresponding state. For following atomic updates, the state of that attribute transfers along the diagram. If the final state of that attribute is *Add*$'$ or *Multiply*$'$ after all the atomic updates are executed, its state transfers to *Replace* automatically.

After all the atomic updates are handled, the update components should be re-randomized such that the shared secret is fresh. The re-randomization processes are as follows:

Run BASE.Encrypt to generate a new ciphertext corresponding to new access policy $\mathbb{A}'$. The input plaintext is the identity element instead of M. Then multiply each ciphertext component of UC by corresponding component of the new ciphertext.

CiphertextUpdate. Given the ciphertext CT corresponding to the old access policy $\mathbb{A}$, and the update components UC corresponding to the new access policy $\mathbb{A}'$, the ciphertext update algorithm outputs new ciphertext CT$'$ corresponding to $\mathbb{A}'$.

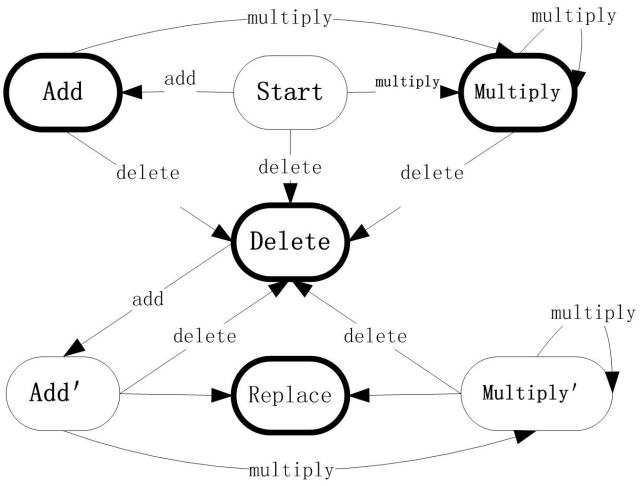


Fig. 2. State Transition Diagram

First, for the common ciphertext components, such as C and C_0, the ciphertext update algorithm multiplies them by corresponding common components of UC.

Second, for the ciphertext components $(C_{1,i}, \cdots, C_{m,i})$, which are associated to an attribute i, the ciphertext update algorithm deals with them as follows:

1. If the state of the attribute i in UC is $Multiply$, new ciphertext components equal that $C_{j,i}$ multiplies the corresponding element of the old ciphertext, where $j = 1, \cdots, m$.

2. If the state of the attribute i in UC is *Replace*, replacing the corresponding components of the old ciphertex with $(C_{1,i}, \cdots, C_{m,i})$.

3. If the state of the attribute i in UC is *Delete*, deleting all the corresponding components of the old ciphertext.

4. If the state of the attribute i in UC is *Add*, adding $(C_{1,i}, \cdots, C_{m,i})$ to corresponding position.

At last, the new ciphertext is generated.

Correctness.

1.Given CT that includes $\mathbb{A} = (W, \rho)$, which is generated by the encryption algorithm, and SK that includes $\mathcal{S}$, if $\mathcal{S}$ does not satisfy $\mathbb{A}$, the decryption fails. Otherwise, if $\mathcal{S}$ satisfies $\mathbb{A}$, Let $I \subseteq \{1, \cdots, l\}$ be defined as $I = \{i : \rho(i) \in \mathcal{S}\}$. Find a set of constants $\{\omega_i\}_{i \in I}$, such that $\sum_{i \in I} \omega_i \overrightarrow{W}_i = (1, 0, \cdots, 0)$. Compute $B_i = e(C_{1,i}, K_{1,\rho(i)}) \cdots e(C_{m,i}, K_{m,\rho(i)}) = A^{\lambda_i}$, $B = \prod_{i \in I}(B_i)^{\omega_i} = A^{\sum_{i \in I} \lambda_i \omega_i} = A^{\overrightarrow{v} \sum_{i \in I} \overrightarrow{W}_i \omega_i} = A^s$, and $M = BC/e(C_0, K_0)$.

2.Given CT′ according to $\mathbb{A}' = (W', \rho)$, which is generated by the policy update algorithm, and SK according to $\mathcal{S}$, if $\mathcal{S}$ does not satisfy $\mathbb{A}'$, the decryption fails. Otherwise, if $\mathcal{S}$ satisfies $\mathbb{A}'$, Let $I' \subseteq \{1, \cdots, l'\}$ be defined as $I' = \{i : \rho(i) \in \mathcal{S}\}$. Based on the update times of the ciphertext components, I'

can be regarded as the union of subsets $(I_0, I_1, \cdots, I_{max})$. A subset I_q includes the attributes, whose associated ciphertext components are updated for q times and I_{max} is the maximum one of all these subset in I'. Note, I_{max} is determined after CT$'$ is generated. For each attribute $i_q \in I_q$, $(C^{(q)}_{1,i_q}, \cdots, C^{(q)}_{m,i_q})$ are the corresponding ciphertext components, and $\lambda^{(q)}_{i_q}$ and $\overrightarrow{W}^{(q)}_{i_q}$ are the corresponding share and vector. We have

$$
\begin{aligned}
B' &= \prod_{i_0 \in I_0} (B_{i_0})^{\omega_{i_0}} \cdot \prod_{i_1 \in I_1} (B^{(1)}_{i_1})^{\omega_{i_1}} \cdot \cdots \cdot \prod_{i_{max} \in I_{max}} (B^{(max)}_{i_{max}})^{\omega_{i_{max}}} \\
&= \prod_{i_0 \in I_0} (e(C_{1,i_0}, K_{1,\rho(i_0)}) \cdots e(C_{m,i_0}, K_{m,\rho(i_0)}))^{\omega_{i_0}} \cdot \prod_{i_1 \in I_1} (e(C^{(1)}_{1,i_1}, K_{1,\rho(i_1)}) \cdots e(C^{(1)}_{m,i_1}, \\
&\quad K_{m,\rho(i_1)}))^{\omega_{i_1}} \cdot \cdots \cdot \prod_{i_{max} \in I_{max}} (e(C^{(max)}_{1,i_{max}}, K_{1,\rho(i_{max})}) \cdots e(C^{(max)}_{m,i_{max}}, K_{m,\rho(i_{max})}))^{\omega_{i_{max}}} \\
&= A^{\sum_{i_0 \in I_0} \lambda_{i_0} \omega_{i_0} + \sum_{i_1 \in I_1} \lambda^{(1)}_{i_1} \omega_{i_1} + \cdots + \sum_{i_{max} \in I_{max}} \lambda^{(max)}_{i_{max}} \omega_{i_{max}}} \\
&= A^{\overrightarrow{v} (\sum_{i_0 \in I_0} \overrightarrow{W}_{i_0} \omega_{i_0} + \sum_{i_1 \in I_1} \overrightarrow{W}^{(1)}_{i_1} \omega_{i_1} + \cdots + \sum_{i_{max} \in I_{max}} \overrightarrow{W}^{(max)}_{i_{max}} \omega_{i_{max}})}
\end{aligned}
$$

For each attribute $i_q \in I_q$, we may assume its associated ciphertext components are updated for max times. For the latter $max - q$ updates, policy update does not change any ciphertext components. Then we have $B' = A^{\overrightarrow{v} \sum_{i_{max} \in I_{max}} \overrightarrow{W}^{(max)}_{i_{max}} \omega_{i_{max}}}$, where $I_{max} = I'$.

Based on the goal of the matrix update algorithm, the new matrix also shares the vector $(1, \cdots, 0)$. Thus, we have $B' = A^s$ as well.

If we do not consider the influence of re-randomization, the intermediate result B, which is recovered from the new ciphertext outputted by the policy update algorithm, is always the same as the initial one, which is recovered from the old ciphertext outputted by the encryption algorithm. In re-randomization phase, C_0 will be updated synchronously with C. Thus, $BC/e(C_0, K_0)$ also outputs correct M.

6 Security Analysis

In this section, we prove that a PU-CP-ABE scheme is secure if the basic CP-ABE scheme is secure.

Theorem 1. If the security of the basic CP-ABE scheme holds, all polynomial time adversaries have at most a negligible advantage in breaking the PU-CP-ABE scheme.

Proof. To prove the theorem, we will show that if an adversary $\mathcal{A}$ win the security game again PU-CP-ABE with a non-negligible advantage, a simulator $\mathcal{B}$ is able to break the security of the basic CP-ABE scheme.

Init. $\mathcal{B}$ receives the challenge access policy $\mathbb{A}^* = (W_{l^* \times n^*}, \rho^*)$.

Setup. $\mathcal{B}$ selects parameters and constructs the public key PK and the master secret key MK. Then it computes its private key $\mathrm{SK}_{\mathcal{B}}$. PK is sent to $\mathcal{A}$.

Query phase 1. For each secret key query from $\mathcal{A}$, $\mathcal{B}$ responds corresponding key to $\mathcal{A}$.

Challenge. After receiving two plaintexts (M_0, M_1) and an access policy $\mathbb{A}_0 = (W_{l_0 \times n_0}, \rho)$, $\mathcal{B}$ flips a random coin $b \in \{0,1\}$ and simulates the ciphertext $CT_0^{(b)}$ = ($\mathbb{A}_0$, C, C_0, $\forall i \in [1, l_0]$, $\{C_{1,i}, \cdots, C_{m,i}\}$) with BASE.Encrypt(PK, M_b, $SK_{\mathcal{B}}$, $\mathbb{A}_0$). C, C_0 represent the common ciphertext components. $C_{1,i}, \cdots, C_{m,i}$ are associated with row $i \in [1, l_0]$. To simulate these ciphertext components, $\mathcal{B}$ selects the vector $\overrightarrow{v} = (s, y_2, y_3, \cdots, y_n)$ with PRF f and a special attribute $x \in \mathcal{S}_{\mathcal{B}}$ at first, and then computes $\lambda_i = \overrightarrow{v}\overrightarrow{W}_i$.

If $\mathbb{A}_0 = \mathbb{A}^*$, the challenge ciphertext has been simulated. Otherwise, $\mathcal{B}$ continues to accept policy update queries $\mathbb{A}_i$. The simulation for PolicyUpdate can be decomposed to a series of atomic updates and a re-randomization. For an atomic update that an attribute A is added into or deleted from the current policy, $\mathcal{B}$ simulates the 4 kinds of update components as follows.

For Case 1), $\{C_{1,A}, \cdots, C_{m,A}\}$ are removed.

For Case 2), $\{C_{1,A}, \cdots, C_{m,A}\}$ are removed. Find the set of attribute $N(\mathbb{A}_{i-1}, A)$ at first. Then, for each attribute $\rho(i) \in N(\mathbb{A}_{i-1}, A)$, simulate associated ciphertext components $\{C_{1,i}, \cdots, C_{m,i}\}$ with BASE.Encrypt. The difference is that λ_A instead of λ_i is embedded into $C_{1,i}$.

For case 3), First, compute $\lambda_{\mathbb{A}_{i-1}}$ and let $\lambda_A = \lambda_{\mathbb{A}_{i-1}}$. Then, simulate associated ciphertext components $\{C_{1,A}, \cdots, C_{m,A}\}$ with BASE.Encrypt.

For case 4), First, select y_A with PRF f and a special attribute $x \in \mathcal{S}_{\mathcal{B}}$ and set $\lambda_A = -y_A$. Then, $\{C_{1,A}, \cdots, C_{m,A}\}$ can be simulated with BASE.Encrypt. Next, find the set of attribute $N(\mathbb{A}_{i-1}, A)$. For each attribute $\rho(i) \in N(\mathbb{A}_{i-1}, A)$, call BASE.Encrypt to simulate associated ciphertext components $\{C_{1,i}, \cdots, C_{m,i}\}$ with y_A instead of λ_i.

For the re-randomization, call BASE.Encrypt(PK, 1, $SK_{\mathcal{B}}$, $\mathbb{A}_i$) to simulate a new ciphertext. Then multiply each ciphertext component by the corresponding component of the new ciphertext. UC_{i-1} is generated. Finally, $\mathcal{A}$ is able to obtain $CT_i^{(b)}$ with CipertextUpdate(UC_{i-1}, $CT_{i-1}^{(b)}$).

Query phase 2. $\mathcal{B}$ proceeds as in query phase 1.

Guess. Finally, $\mathcal{A}$ outputs its guess $b'_{\mathcal{A}}$. If $b' = 0$, $\mathcal{B}$ output $b'_{\mathcal{B}}$. Otherwise, $\mathcal{B}$ output 1-$b'_{\mathcal{B}}$. The distribution for $\mathcal{A}$ is perfect.

The responses that $\mathcal{B}$ returns to $\mathcal{A}$ are distributed identically as in the game defined in Section 3.2. If $\mathcal{A}$ wins this security game with a non-negligible advantage, $\mathcal{B}$ has same advantage in breaking the security assumption of the based CP-ABE scheme.

7 An Example of PU-CP-ABE

In this section, we give a detailed example to update access policy on a typical large attribute universe CP-ABE scheme [3]. The example shows how to use PU-CP-ABE intuitively. The details are as follows.

A system administer first publishes PK=($\mathbb{G}, \mathbb{G}_T, e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T, g, u, h, w, v, e(g,g)^{\alpha}, \mathcal{G}$) with **Setup**, and keeps MK=(α) as a secret.

For two sets of attributes S_1 = (A, C), S_2 = (A, B, D), the system administer calls **KeyGen** to generate private keys for them. For S_1, it selects a group of random numbers r, r_A, r_C and a PRF f_1, and then computes SK_1 as follows:

$$S_1, f_1, K_0 = g^{\alpha} w^r, K_1 = g^r, K_{2,A} = g^{r_A}, K_{3,A} = (u^A h)^{r_A} v^{-r},$$
$$K_{2,C} = g^{r_C}, K_{3,C} = (u^C h)^{r_C} v^{-r}$$

For S_2, it selects another group of random numbers r, r_A, r_B, r_D and a PRF f_2, and then computes SK_2 as follows:

$$S_2, f_2, K_0 = g^{\alpha} w^r, K_1 = g^r, K_{2,A} = g^{r_A}, K_{3,A} = (u^A h)^{r_A} v^{-r},$$
$$K_{2,B} = g^{r_B}, K_{3,B} = (u^B h)^{r_B} v^{-r},$$
$$K_{2,D} = g^{r_D}, K_{3,D} = (u^D h)^{r_D} v^{-r}$$

Define policy $\mathbb{A}$ = ((A *or* B) *and* C). A data owner encrypts a document M under $\mathbb{A}$ with his private key. The ciphertext is generated as follows:

Phase $\mathbb{A}$ to matrix $(W_{2\times 3}, \rho) = (\overrightarrow{W}_1 = (1,1), \overrightarrow{W}_2 = (1,1), \overrightarrow{W}_3 = (0,-1), \rho(1) = A, \rho(2) = B, \rho(3) = C)$. Select a vector $\overrightarrow{v} = (s, y_2)$ from $f_{owner}(ID_{owner})$, and 3 random numbers t_1, t_2, and t_3, compute $\lambda_1 = \overrightarrow{v}\overrightarrow{W}_1 = s + y_2$, $\lambda_2 = \overrightarrow{v}\overrightarrow{W}_2 = s + y_2$, $\lambda_3 = \overrightarrow{v}\overrightarrow{W}_3 = -y_2$.

$$CT = (\mathbb{A} = ((A \ or \ B) \ and \ C), C = Me(g,g)^{\alpha s}, C_0 = g^s,$$
$$C_{1,1} = w^{(\lambda_1 = s+y_2)} v^{t_1}, C_{2,1} = (u^{\rho(1)} h)^{-t_1}, C_{3,1} = g^{t_1},$$
$$C_{1,2} = w^{(\lambda_2 = s+y_2)} v^{t_2}, C_{2,2} = (u^{\rho(2)} h)^{-t_2}, C_{3,2} = g^{t_2},$$
$$C_{1,3} = w^{(\lambda_3 = -y_2)} v^{t_3}, C_{2,3} = (u^{\rho(3)} h)^{-t_3}, C_{3,3} = g^{t_3})$$

CT is uploaded on a cloud server.

At the moment, the users holding SK1 are able to download and decrypt CT. The decryption process is as follows.

Given CT, for users with SK_1, Let $I = \{A, C\}$, and $\omega = (1, 0, 1, 0)$

$$\begin{aligned}
B_1 &= e(C_{1,1}, K_1) e(C_{2,1}, K_{2,\rho(1)}) e(C_{3,1}, K_{3,\rho(1)}) \\
&= e(w^{\lambda_1} v^{t_1}, g^r) e((u^{\rho(1)} h)^{-t_1}, g^{r_{\rho(1)}}) e(g^{t_1}, (u^{\rho(1)} h)^{r_{\rho(1)}} v^{-r}) \\
&= e(w, g)^{r\lambda_1} \\
B_3 &= e(C_{1,3}, K_1) e(C_{2,3}, K_{2,\rho(3)}) e(C_{3,3}, K_{3,\rho(3)}) \\
&= e(w^{\lambda_3} v^{t_3}, g^r) e((u^{\rho(3)} h)^{-t_3}, g^{r_{\rho(3)}}) e(g^{t_3}, (u^{\rho(3)} h)^{r_{\rho(3)}} v^{-r}) \\
&= e(w, g)^{r\lambda_3} \\
B &= (B_1 B_3) = e(w, g)^{r(\lambda_1 + \lambda_3)} = e(w, g)^{rs} \\
&\frac{C \cdot B}{e(C_0, K_0)} = \frac{M e(g,g)^{\alpha s} e(w,g)^{rs}}{e(g^s, g^{\alpha} w^r)} \Rightarrow M
\end{aligned}$$

Next, the data owner changes the access policy to $\mathbb{A}'$= ((A *and* B) *or* D) that deletes an old attribute C and adds a newly issued attribute D comparing

with $\mathbb{A}$. Obviously, he is able to directly call the encryption algorithm with $\mathbb{A}'$ and the plaintext M. The ciphertext is as follows:

$$\begin{aligned} CT' = (\mathbb{A} = ((A\ and\ B)\ or\ D), C = Me(g,g)^{\alpha s}, C_0 = g^s, \\ C_{1,1} = w^{(\lambda_1 = s+y_2)} v^{t_1}, C_{2,1} = (u^{\rho(1)}h)^{-t_1}, C_{3,1} = g^{t_1}, \\ C_{1,2} = w^{(\lambda_2 = -y_2)} v^{t_2}, C_{2,2} = (u^{\rho(2)}h)^{-t_2}, C_{3,2} = g^{t_2}, \\ C_{1,4} = w^{(\lambda_4 = s)} v^{t_4}\ , C_{2,4} = (u^{\rho(4)}h)^{-t_4}, C_{3,4} = g^{t_4}) \end{aligned}$$

On the other hand, he is able to generate update components UC for $\mathbb{A}'$ as well. First, select $\overrightarrow{v} = (s, y_2)$ from $f_{owner}(ID_{owner})$, and then recover λ_1, λ_2, and λ_3. The processes of the policy update are decomposed to following 4 successive atomic updates:

1.((A *or* B) *and* C)→(A *or* B)

2.(A *or* B)→A

3. A→(A *and* B)

4.(A *and* B)→((A *and* B) *or* D)

The associated operations are as follows:

1. Atomic update 1 satisfies Case 2). $(C_{1,3}, C_{2,3}, C_{3,3})$ are removed and *delete* is attached to attribute C. Both A and B are affected when deleting C. We have $N(((A$ or $B)$ and $C)$, $C)$ $=(A, B)$. Correspondingly, $\lambda_1 = \lambda_1 + \lambda_3 = s$, $\lambda_2 = \lambda_2 + \lambda_3 = s$, and λ_3 is set to *null*. For attributes A, B, select random numbers $t_1^{(C)}$ and $t_2^{(C)}$ from $\mathbb{Z}_p$, and compute associated ciphertext components. We have

$$\begin{aligned} UC_1 = (\mathbb{A}_1 = (A\ or\ B), \\ Multiply, C_{1,1} = w^{(\lambda_3 = -y_2)} v^{t_1^{(C)}}, C_{2,1} = (u^{\rho(1)}h)^{-t_1^{(C)}}, C_{3,1} = g^{t_1^{(C)}}, \\ Multiply, C_{1,2} = w^{(\lambda_3 = -y_2)} v^{t_2^{(C)}}, C_{2,2} = (u^{\rho(2)}h)^{-t_2^{(C)}}, C_{3,2} = g^{t_2^{(C)}}, \\ Delete) \end{aligned}$$

2. Atomic update 2 satisfies Case 1). $(C_{1,2}, C_{2,2}, C_{3,2})$ are removed and *delete* is attached to attribute B. Correspondingly, λ_2 is set to *null*. We have

$$\begin{aligned} UC_2 = (\mathbb{A}_2 = (A), \\ Multiply, C_{1,1} = w^{(\lambda_3 = -y_2)} v^{t_1^{(C)}}, C_{2,1} = (u^{\rho(1)}h)^{-t_1^{(C)}}, C_{3,1} = g^{t_1^{(C)}}, \\ Delete \\ Delete) \end{aligned}$$

3. Atomic update 3 satisfies Case 4). Select y_2' from $f_{owner}(ID_{owner})$ and set $\lambda_2 = -y_2'$. Attribute A is affected when adding B. We have $N((A), B)$=(A). Correspondingly, λ_1 is set to $\lambda_1 + y_2'$. Select a random number $t_1^{(B)} \in \mathbb{Z}_p$ for

attribute A, and compute

$$C_{1,1} = w^{(\lambda_3 = -y_2)} v^{t_1^{(C)}} w^{y_2'} v^{t_1^{(B)}}, C_{2,1} = (u^{\rho(1)} h)^{-t_1^{(C)}} (u^{\rho(1)} h)^{-t_1^{(B)}}, C_{3,1} = g^{t_1^{(C)}} g^{t_1^{(B)}}$$

Next, compute new ciphertext components

$$C_{1,2} = w^{(\lambda_2 = -y_2')} v^{t_2^{(B)}}, C_{2,2} = (u^{\rho(2)} h)^{-t_2^{(B)}}, C_{3,2} = g^{t_2^{(B)}}$$

We have

$$\begin{aligned}
&UC_3 = (\mathbb{A}_3 = (A \ and \ B),\\
&Multiply, C_{1,1} = w^{(\lambda_3 = -y_2)} v^{t_1^{(C)}} w^{y_2'} v^{t_1^{(B)}}, C_{2,1} = (u^{\rho(1)} h)^{-t_1^{(C)}} (u^{\rho(1)} h)^{-t_1^{(B)}},\\
&\qquad C_{3,1} = g^{t_1^{(C)}} g^{t_1^{(B)}},\\
&Add', \qquad C_{1,2} = w^{(\lambda_2 = -y_2')} v^{t_2^{(B)}}, C_{2,2} = (u^{\rho(2)} h)^{-t_2^{(B)}}, C_{3,2} = g^{t_2^{(B)}}\\
&Delete)
\end{aligned}$$

4. Atomic update 4 satisfies Case 3). First compute $\lambda_{(A \ and \ B)} = \lambda_1 + \lambda_2 = s$, set $\lambda_4 = \lambda_{(A \ and \ B)} = s$. Then, select a random number $t_4^{(D)} \in \mathbb{Z}_p$ and compute new ciphertext components

$$C_{1,4} = w^{(\lambda_4 = s)} v^{t_4^{(D)}}, C_{2,4} = (u^{\rho(4)} h)^{-t_4^{(D)}}, C_{3,4} = g^{t_4^{(D)}}$$

We have

$$\begin{aligned}
&UC_4 = (\mathbb{A}_4 = ((A \ and \ B) \ or \ D),\\
&Multiply, C_{1,1} = w^{(\lambda_3 = -y_2)} v^{t_1^{(C)}} w^{y_2'} v^{t_1^{(B)}}, C_{2,1} = (u^{\rho(1)} h)^{-t_1^{(C)}} (u^{\rho(1)} h)^{-t_1^{(B)}},\\
&\qquad C_{3,1} = g^{t_1^{(C)}} g^{t_1^{(B)}},\\
&Add', \qquad C_{1,2} = w^{(\lambda_2 = -y_2')} v^{t_2^{(B)}}, C_{2,2} = (u^{\rho(2)} h)^{-t_2^{(B)}}, C_{3,2} = g^{t_2^{(B)}},\\
&Delete\\
&Add, \qquad C_{1,4} = w^{(\lambda_4 = s)} v^{t_4^{(D)}}, C_{2,4} = (u^{\rho(4)} h)^{-t_4^{(D)}}, C_{3,4} = g^{t_4^{(D)}})
\end{aligned}$$

After these atomic updates, the update components should be re-randomized. First, generate a new ciphertext CT^* under $\mathbb{A}'$ with identity element.

$$\begin{aligned}
CT^* = (\mathbb{A}' = ((A \ and \ B) \ or \ D), &C^* = e(g,g)^{\alpha s^*}, C_0^* = g^{s^*},\\
&C_{1,1}^* = w^{(\lambda_1 = s^* + y_2^*)} v^{t_1^*}, C_{2,1}^* = (u^{\rho(1)} h)^{-t_1^*}, C_{3,1}^* = g^{t_1^*},\\
&C_{1,2}^* = w^{(\lambda_2 = -y_2^*)} v^{t_2^*}, C_{2,2}^* = (u^{\rho(2)} h)^{-t_2^*}, C_{3,2}^* = g^{t_2^*},\\
&C_{1,4}^* = w^{(\lambda_4 = s^*)} v^{t_4^*}, C_{2,4^*} = (u^{\rho(4)} h)^{-t_4^*}, C_{3,4}^* = g^{t_4^*})
\end{aligned}$$

Then, multiply each component of UC_4 by corresponding component of CT^*. UC is as follows.

$$UC = (\mathbb{A}' = ((A \ and \ B) \ or \ D), C' = e(g,g)^{\alpha s^*}, C_0' = g^{s^*},$$

$$
\begin{aligned}
&Multiply, C_{1,1} = w^{(s^*-y_2+y_2'+y_2^*)} v^{(t_1^{(B)}+t_1^{(C)}+t_1^*)}, C_{2,1} = (u^{\rho(1)}h)^{-(t_1^{(B)}+t_1^{(C)}+t_1^*)},\\
&\qquad C_{3,1} = g^{(t_1^{(B)}+t_1^{(C)}+t_1^*)},\\
&Replace,\ \ C_{1,2} = w^{-(y_2'+y_2^*)} v^{(t_2^{(B)}+t_2^*)}, C_{2,2} = (u^{\rho(2)}h)^{-(t_2^{(B)}+t_2^*)}, C_{3,2} = g^{(t_2^{(B)}+t_2^*)},\\
&Delete\\
&Add,\qquad C_{1,4} = w^{(s+s^*)} v^{(t_4^{(D)}+t_4^*)}, C_{2,4} = (u^{\rho(4)}h)^{-(t_4^{(D)}+t_4^*)}, C_{3,4} = g^{(t_4^{(D)}+t_4^*)})
\end{aligned}
$$

After receiving UC, the cloud server generates CT$'$ with CT and UC with **CiphertextUpdate** as follows.

$$
\begin{aligned}
&CT' = (\mathbb{A} = ((A\ and\ B)\ or\ D), C = Me(g,g)^{\alpha(s+s^*)}, C_0 = g^{(s+s^*)},\\
&C_{1,1} = w^{(s+s^*+y_2'+y_2^*)} v^{(t_1+t_1^{(B)}+t_1^{(C)}+t_1^*)}, C_{2,1} = (u^{\rho(1)}h)^{-(t_1+t_1^{(B)}+t_1^{(C)}+t_1^*)},\\
&C_{3,1} = g^{(t_1+t_1^{(B)}+t_1^{(C)}+t_1^*)},\\
&C_{1,2} = w^{-(y_2'+y_2^*)} v^{(t_2^{(B)}+t_2^*)}, C_{2,2} = (u^{\rho(2)}h)^{-(t_2^{(B)}+t_2^*)}, C_{3,2} = g^{(t_2^{(B)}+t_2^*)},\\
&C_{1,4} = w^{(s+s^*)} v^{(t_4^{(D)}+t_4^*)}, C_{2,4} = (u^{\rho(4)}h)^{-(t_4^{(D)}+t_4^*)}, C_{3,4} = g^{(t_4^{(D)}+t_4^*)})
\end{aligned}
$$

We can see that *the distribution of CT$'$ outputted by the ciphertext update algorithm is the same as the distribution of CT$'$ outputted by the encryption algorithm.*

As a result, the new ciphertext CT$'$ cannot be decrypted by the users holding SK_1. In addition, these users cannot obtain new data encrypted under $\mathbb{A}'$. Meanwhile, the users holding SK_2 are able to obtain M since S_2 = (A, B, D) satisfies $\mathbb{A}'$= ((A *and* B) *or* D). The decryption process is as follows.

Given CT$'$, for users with SK_2, Let $I = \{A, B, D\}$, and $\omega = (1, 1, 0, 0)$.

$$
\begin{aligned}
B_1 &= e(C_{1,1}, K_1)e(C_{2,1}, K_{2,\rho(1)})e(C_{3,1}, K_{3,\rho(1)})\\
&= e(w^{(s+s^*+y_2'+y_2^*)} v^{(t_1+t_1^{(B)}+t_1^{(C)}+t_1^*)}, g^r)e((u^{\rho(1)}h)^{-(t_1+t_1^{(B)}+t_1^{(C)}+t_1^*)}, g^{r_{\rho(1)}})\\
&\times e(g^{(t_1+t_1^{(B)}+t_1^{(C)}+t_1^*)}, (u^{\rho(1)}h)^{r_{\rho(1)}} v^{-r})\\
&= e(w,g)^{r(s+s^*+y_2'+y_2^*)}\\
B_2 &= e(C_{1,2}, K_1)e(C_{2,2}, K_{2,\rho(2)})e(C_{3,2}, K_{3,\rho(2)})\\
&= e(w^{-(y_2'+y_2^*)} v^{(t_2^{(B)}+t_2^*)}, g^r)e((u^{\rho(2)}h)^{-(t_2^{(B)}+t_2^*)}, g^{r_{\rho(2)}})e(g^{(t_2^{(B)}+t_2^*)}, (u^B h)^{r_{\rho(2)}} v^{-r})\\
&= e(w,g)^{-r(y_2'+y_2^*)}\\
B &= (B_1 B_2) = e(w,g)^{r(s+s^*+y_2'+y_2^*-y_2'-y_2^*)} = e(w,g)^{r(s+s^*)}\\
&\frac{C \cdot B}{e(C_0, K_0)} = \frac{Me(g,g)^{\alpha(s+s^*)} e(w,g)^{r(s+s^*)}}{e(g^{(s+s^*)}, g^\alpha w^r)} \Rightarrow M
\end{aligned}
$$

Eventually, the access policy of the document is flexibly changed by the data owner. Readers is able to apply our policy update on the other CP-ABE schemes [1, 10, 16, 20–24].

8 Performance Analysis

In this section, we compare the efficiency of a typical policy update scheme [12] and PU-CP-ABE, and test executing time of PU-CP-ABE.

Let u denote the number of users and a denote the number of attributes in an access policy or an attribute set. Define that generating or transmitting a group of ciphertext/key components is a basic operation. We omit the operations of generating the common components. Then generating or transmitting a ciphertext/key needs a operations and storing space for a ciphertext/key is a.

In scheme of [12], data owner sends a ciphertext. After receiving the ciphertext, a key update server sends an update key for each user. The communication operations and the computation operations of the server are au. The new storage space to store the state information is a.

In PU-CP-ABE, data owner generates and sends the ciphertext update components. The communication operations are a. To estimate the computation operations of PU-CP-ABE, we should count out the 4 cases of atomic updates respectively.

For case 1), 0 attribute will be affected. For case 2), if all the other connectors are "*or*", a-1 attributes will be affected. Otherwise, if all the other connectors are "*and*", 1 attribute will be affected. Averagely, the expected number of affected attributes are $\frac{a}{2}$. For case 3), only the new attribute is added. No other attributes will be affected. For case 4), the new attribute is added. The number of affected attributes is the same as in the case 2).

Suppose the ratio of the 4 cases are equal. The expected computation costs of an atomic update are $\frac{a+2}{4}$. If a policy update only contains 1 atomic update, the minimum computation costs are $\frac{a+2}{4}$. Otherwise, if a policy update contains a atomic updates, the maximum computation costs are $\frac{a^2+2a}{4}$. Thus, the expected computation costs of a policy update is $\frac{a^2+3a+2}{8}$.

Theoretically, when the number of different attributes between the two access policies is less than 5, the expected computation costs of the policy update algorithm is less than those of the encryption algorithm. However, we also need to consider the costs of re-randomization. Its computation costs are the same as encryption. Thus, the expected computation costs of a policy update is $\frac{a^2+11a+2}{8}$. This number is more than that of the encryption algorithm.

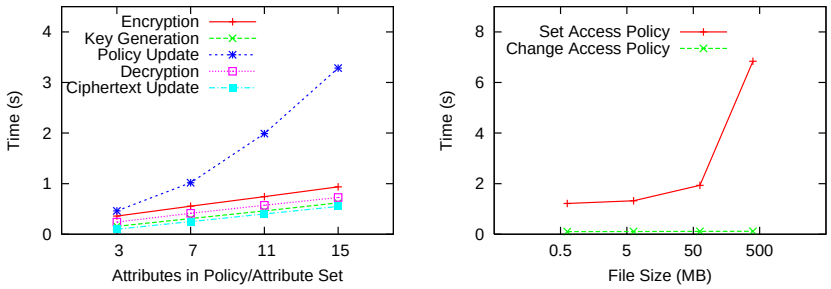
A fact should be notice that since the policy update does not need ciphertexts as input, the update components can be pre-computed in off-line manner. The comparison result is list in Table.1.

Table 1. Efficiency and Conditions Comparison

	Communication		Computation		Storage		Periodic	Private
	Server	Owner	Server	Owner	Server	Owner	Update	Channel
Scheme of [12]	au	a	au	a	0	a	Y	Y
PU-CP-ABE	a	a	a	$\frac{a^2+11a+2}{8}$	0	0	N	N

Comparing with the scheme of [12], PU-CP-ABE does not need private channel to transmit secret keys and ciphertexts do not need periodically updating. Most importantly, the communication and computation costs no longer depend on the number of users.

Next, we implement PU-CP-ABE based on a well known CP-ABE scheme, Scheme1 of [20]. The codes are programmed with the C language on Microsoft VS 2005. The bilinear pairing is based on PBC library v0.5.14 [25]. The pairing groups are implemented on an asymmetric elliptic curve, MNT, over 224-bit finite field. Namely, there are three groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$. The pairing e is a map from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$. The test machine is a ThinkPad X1 laptop with Intel Core i5-3337U 1.8GHz and 64-bit Microsoft Windows 7 operating system.



(a) Scheme1 of [20] with Policy Update (b) Hybird Encryption with AES 256

Fig. 3. Executing Time of PU-CP-ABE

We execute the encryption algorithm, the key generation algorithm, the policy update algorithm, the decryption algorithm and the ciphertext update algorithm under the scenarios that 3, 7, 11, 15 attributes are in the input access policy/attribute set, respectively. Fig.3(a) shows the result. The number of attributes in the policy update algorithm and the ciphertext update algorithm are the different attributes between the old access policy and the new one. In practical applications, this number will far less than the total number of attributes in the access policy of the encryption algorithm, because updating 1 attribute is able to change the decryption privilege of a ciphertext.

Moreover, we test the time of generating encrypted files by hybrid encryption and the time to change access policy of the files by PU-CP-ABE. We encrypt 4 files of different sizes (a 0.607MB .txt file, a 7.45MB .pdf file, a 55.1MB .mpg file, and a 428MB .rmvb file) with AES 256. The 256-bit encryption key is extracted from a group element in $\mathbb{G}_T$ with the base64 coding. Then the element is encrypted by PU-CP-ABE. 15 attributes are in the input access policy of the encryption algorithm and 3 of them are changed in the policy update algorithm to modify access policy of the encrypted files. Eventually, the test result is shown in Fig.3(b).

Viewing from the Fig.3(a), the policy update time is more than the encryption time and the ciphertext update time is less than the encryption time. Since policy update can be executed by a data owner in off-line manner, the ciphertext update even needs less time than generate a new ciphertext for cloud server.

Viewing from the Fig.3(b), the time of changing access policy of a file is less than the time of setting initial access policy for a new file. Moreover, with the increase of file size, the setting access policy time is increased linearly while the changing access policy time is nearly not increased. It is a practical function to manage data access policy in secure cloud storage systems.

9 Conclusions

In this paper, we propose a policy update method for ciphertext-policy attribute-based encryption. Our method allows data owners to dynamically change data access policy after the ciphertext has been generated. The new method adds the update function for CP-ABE such that cryptographically enforced access control becomes a complete access control tool.

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-Policy Attribute-Based Encryption, IEEE S&P 2007, pp.321-334, 2007.
2. Beimel, A.: Secure Schemes for Secret Sharing and Key Distribution, PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1996.
3. Rouselakis, Y., Waters, B.: Practical constructions and new proof methods for large universe attribute-based encryption, ACM CCS 2013, pp.463-474, 2013.
4. Diffie, W., Hellman, M.: New directions in cryptography, IEEE Transactions on Information Theory, vol.22, no.6, pp.644-654, 1976.
5. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signa-tures and public-key cryptosystems, Communications of ACM vol.21, no.2, pp.120-126, 1978.
6. Boldyreva, A., Goyal, V., Kumar, V.: Identity-based Encryption with Efficient Revocation, ACM CCS 2008, pp.417-426, 2008.
7. Sahai, A., Waters, B.: Fuzzy identity based encryption, EUROCRYPT 2005, LNCS 3494, pp.457-473, Springer, Heidelberg, 2005.
8. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-Based encryption for Fine-Grained Access Control of Encrypted Data, ACM CCS 2006, pp.89-98, Virginia, 2006.
9. Pirretti, M., Traynor, P., Mcdaniel, P., Waters, B.: Secure attribute-based systems, ACM CCS 2006, pp.99-112, 2006.
10. Attrapadung, N., Imai, H.: Conjunctive Broadcast and Attribute-Based Encryption, Pairing 2009, LNCS 5671, pp.248-265, Springer, Heidelberg, 2009.
11. Lewko, A., Sahai, A., Waters, B.: Revocation Systems with Very Small Private Keys, IEEE S&P 2010, pp.273-285, 2010.
12. Sahai, A., Seyalioglu, H., Waters, B.: Dynamic Credentials and Ciphertext Delegation for Attribute-based Encryption, CRYPTO 2012, LNCS 7417, pp.199-217, 2012.

13. Yu, S., Wang, C., Ren, K.: Attribute Based Data Sharing with Attribute Revocation, ACM ASIACCS 2010,pp.261-270, 2010.
14. Cheung, L., Newport, C.: Provably Secure Ciphertext Policy ABE, ACM CCS 2007, pp.456-465, 2007.
15. Yang, K., Jia, X.H., Ren, K., Zhang, B., Xie, R.T.: Dac-macs: Effective data access control for Multiauthority cloud storage systems, IEEE Transactions on Information Forensics and Security, vol.8, no.11, pp.1790-1801, 2013.
16. Lewko A., Waters, B.: Decentralizing Attribute-Based Encryption, EUROCRYPT 2011, LNCS 6632, pp.568-588, Springer, Heidelberg, 2011.
17. Yang, K., Jia, X.H., Ren, K., Xie, R.T., Huang, L.S.: Enabling Efficient Access Control with Dynamic Policy Updating for Big Data in the Cloud, INFOCOM 2014, pp.2013-2021, 2014.
18. Liang, K.T., Au, H. M., Liu, K. J., et al.: A secure and efficient Ciphertext-Policy Attribute-Based Proxy Re-Encryption for cloud data sharing, Future Generation Computer Systems, vol.52, pp.95-108, 2016.
19. Hong, J., Xue, K.P, Li, W.: Comments on DAC-MACS: Effective Data Access Control for Multiauthority Cloud Storage Systems, IEEE Transactions on Information Forensics and Security, vol.10, no.6, pp.1315-1317, 2015.
20. Waters, B.: Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization, PKC 2011, LNCS 6571, Springer, Heidelberg, pp.53-70, 2011.
21. Lewko, A., Waters, B.: New proof methods for attribute based encryption: Achieving full security through selective techniques, CRYPTO 2012, pp.180-198, 2012.
22. Hohenberger, S., Waters, B.: Online/Offline Attribute-Based Encryption, PKC 2014, LNCS 8383, Springer, Heidelberg, pp.293-310, 2014.
23. Hohenberger, S., Waters, B.: Attribute-Based Encryption with Fast Decryption, PKC 2013, LNCS 7778, Springer, Heidelberg, pp.162-179, 2013.
24. Ibraimi, L., Tang, Q., Hartel, P., Jonker, W.: Efficient and Provable Secure Ciphertext-Policy Attribute-Based Encryption Schemes, ISPEC 2009, LNCS 5451, pp.1-12, 2009.
25. Lynn, B.: The Stanford pairing based crypto library, http:// crypto. stanford. edu/pbc.