Dynamic Policy Update for Ciphertext-Policy Attribute-Based Encryption

Wei Yuan

State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China. Email: yuanwei@iie.ac.cn

Abstract. Ciphertext-policy attribute-based encryption (CP-ABE) is a promising access control tool in theory. However, there are some unsolved problems in terms of attributes/policy update of CP-ABE. Current revocation/update methods need to update many secret keys for legal users to avoid revoked users refusing cooperation when the key update reduces their privileges. Furthermore, transmitting the update keys needs many private channels between the trusted authority and the key owners. Consequently, the overheads of the update are very large.

In this paper, we abstract a generic CP-ABE scheme from many linear secret sharing (LSS) based CP-ABE schemes. Then we propose a matrix update algorithm to distinguish the attributes, whose corresponding vectors in the LSS matrix are changed when the access policy updates. Combined with the new matrix update algorithm and the abstract CP-ABE scheme, we construct a generic CP-ABE scheme with the function of dynamic policy update. The new scheme is able to directly change data access policy and user secret keys do not need to be updated. If the attributes of a user satisfy the old access policy but he/she does not decrypt the old ciphertext before the policy update, he/she cannot obtain the plaintext again after the ciphertext is updated. As a result, user cannot refuse to cooperate when the policy update reduces his/her privilege. Moreover, private channels to transmit update keys are eliminated. The efficiency of the new update method is higher than previous approaches as well.

Keywords: Access Control; Attribute-Based Encryption; Policy Update

1 Introduction

The public key encryption (PKE) technique [1, 2] has been widely applied into modern secure communication and storage systems. Many standards, e.g. PKCS, are proposed to support flexible credential update for PKE. With the expansion of the network scale and the data size, traditional PKE, a typical one-to-one encryption, already cannot satisfy the demands of practical applications. For example, an institute has 1000 staff and the administrator wants to grant 300 of them the privilege to access an electronic document. The document has to be encrypted for 300 times using different public keys and each ciphertext is different from others. The administrator needs to carefully send each ciphertext to right recipient. If many documents should be distributed, handling so many ciphertexts will become a bottleneck.

To cope with the new demands of confidentiality accompanied by the expansion of the data size, attribute based encryption (ABE) technique, which allows a data owner to encrypt data with the common attributes but not public keys of a group of recipients, was proposed [3]. According to the methods how the decryption can be done, namely, whether the ciphertext corresponds to a policy and the secret keys corresponds to a set of attributes, or vice versa, an ABE scheme is called ciphertext-policy (CP) ABE [4], or key-policy (KP) ABE [5]. The policy and the attributes are chosen independently. Only when the attributes satisfy the policy, the decryption succeeds. Thus, ABE is a one-to-many encryption. Continuing from the above example, the administrator only needs to generate the ciphertext of the document with the common attributes of the 300 staff. Then each one is able to decrypt the ciphertext with his/her own secret key. Compared with the tradition PKE, ABE has obvious advantages in terms of efficiency and is fit for large scale network environment, such as cloud systems.

In the real world, the attributes of an entity often changes with time. However, there is not a standard to support flexible privilege control with ABE. Since the emergence of the first ABE scheme [3], many schemes have been proposed to solve the problem. We classify the existing methods to update attributes or access policy into four kinds.

The first kind of method is downloading ciphertexts from cloud servers and decrypting original data, and then generating new ciphertext according to the decrypted data. This approach is trivial. Usually, a lot of data are stored on cloud servers. If any modification for data access policy results in ciphertexts regeneration, large computation resources and network bandwidth will be wasted. Obviously, a practical system cannot adopt this method.

The second kind of method [4,6] is to set an expiry date for user attributes. Ciphertexts will be updated after the expiry date and the secret key components corresponding to the unexpired attributes will be regenerated to fit for new ciphertexts. As a result, the secret key components corresponding to the expired attributes will be invalid after the expiry date. However, if the real state of a user changes before the expiry date, his/her privilege cannot be changed. That is to say, the expiry date is determined when the secret key is generated. Therefore, this method cannot satisfy the demand of dynamic attribute update.

The third kind of method [7, 8] imitates the revocation list in traditional PKE. A legal/illegal ID list is embedded into input parameters of the encryption algorithm. The condition of successful decryption is that the user attributes satisfy the data access policy as well as the ID of that user is in/not in the ID list. That is to say, the ID list is determined when the ciphertext is generated. If we want to add or delete a user, the ciphertext should be regenerated. Actually, we can regard the ID list as parts of access policy. This method is nearly same as the first one. So, the problems are still unsolved.

The fourth kind of method [9,10] is to let the data owner directly update the ciphertexts such that all the secret keys are invalid temporarily. Accordingly, update keys are generated for some of users to recover their decryption privileges. As a result, the decryption privilege of the users can be directly controlled. This method is similar to the second one. The main difference in details is that the forth method uses random numbers to update ciphertexts and secret keys while the second method mainly relies on timestamp. Unfortunately, the workload of the fourth method is very large since the trusted authority should generate an update key for each legal user. Let's recall the above example. If we want to cancel the decryption privilege of 50 of the 300 staff, we need to update secret keys for all the other 950 staff. Then the update for the ciphertext can come into effect. Furthermore, if a staff has different decryption privileges for different documents, managing his/her secret key will become a complex problem. Hence, although this approach satisfies the demand of dynamic attribute update, the actual efficiency may be not better than that of the tradition PKE.

One may wonder why not directly update the secret keys of the users to be revoked to avoid such a large number of key updates. That is because a user has already owned a secret key before the key update. If the key update reduces his/her privilege, he/she is able to refuse to cooperate. Then the update key is useless. Consequently, the efficiency problem seems to be an unbridgeable gap that prevents ABE from being applied into practical applications. In addition, another difficult problem, which has not been taken into serious consideration, also restricts the prospect of ABE: The update keys cannot be transmitted directly via public network channels. Private channels should be set up between the trusted authority and the key owners before the key updates. Even though we do not consider the high overheads to set up private channels to so many users, these channels cannot always be established when we need them.

Summarized from the above analysis, the essential reason for the efficiency problem and the private channel problem is that secret keys must be updated for changing the decryption privilege of users in current policy/attributes update methods. Private channels are to transmit update keys. Updating secret keys of legal users is to avoid revoked users refusing to update their secret keys. Therefore, an effective solution for the problems is to find a new method to directly change user decryption privilege and the user secret key does not need to be changed at the same time.

Our Contribution. To solve the problems of attributes/policy update of ABE, we summarize the common structure of many linear secret sharing (LSS) [11] based CP-ABE schemes and abstract a generic scheme from these schemes. Then, we propose a matrix update algorithm, which is compatible with the existing matrix generation algorithm, to transform an old LSS matrix to a new one according to the differences between the old access policy and the 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. Only the ciphertext components corresponding to these attributes need to be updated for changing data access policy. Next, we find a reasonable way to combine the new matrix update algorithm with the abstract CP-ABE scheme such that a new policy update algorithm can be constructed with the encryption algorithm of the abstract scheme. Moreover, the distribution of the ciphertext generated by the policy update algorithm is the same as the distribution of the ciphertext directly generated from the encryption algorithm. Eventually, we construct an abstract CP-ABE scheme with the function of dynamic *policy update*, called PU-CP-ABE. [1] The new scheme has following two features:

First, only a policy update algorithm is added into the scheme besides the basic four algorithms (Setup, Key Generation, Encryption, and Decryption). There is no key update as in previous schemes [9,10,12,13]. After the data access policy updates, user secret keys do not need to be changed. If the attributes corresponding to a user secret key satisfy the new access policy, that user is able to successfully decrypt the new ciphertext. Otherwise, if the attributes of a user satisfy the old access policy but he/she does not decrypt the old ciphertext before the policy update, he/she cannot obtain the plaintext again after the ciphertext is updated. If subsequent data are all encrypted with the new policy, that user is excluded from obtaining these data. As a result, a data recipient does not have the right to refuse to cooperate even if the policy update reduces his/her privilege. Moreover, we do not need to worry about transmitting secret keys as well because no secrets need to be transmitted in policy update process.

Second, the structures of the original algorithms in CP-ABE do not need to be changed for adding the policy update algorithm. The setup algorithm, the key generation algorithm, the encryption algorithm and the decryption algorithm do not need extra input parameters, such as timestamp and ID list in [4,6–8], and also does not produce extra public parameters as in [12]. Only some internal numbers to generate the ciphertext in the original encryption algorithm are reserved for policy update as in [9]. Let's recall the above example. The data owner only needs to run the policy update algorithm for once to update the old ciphertext to a new one. Thus, the new scheme avoids the overheads of generating ciphertexts for different recipients or updating secret keys for legal recipients. As a result, our method overcomes the problems in the fourth approach and provides the dynamic privilege control function with high efficiency.

Organization. The remainder of our paper is structured as follows. In Section 2, we discuss some related works. In Section 3, we give some background knowledge. In Section 4, the LSS technique is introduced and our update method based on LSS is presented. We then describe the details of the policy update algorithm and give an example to show its usage in Section 5 and 6. In Section 7 and 8, the security analysis and performance analysis of our scheme is given. Finally, we conclude in Section 9.

2 Related Works

ABE was first introduced by Sahai and Waters [3] to tolerant fuzzy biometric characteristics. Goyal, Pandey, Sahai, and Water [5] later formalized two flavors of ABE: KP-ABE and CP-ABE, and implemented first KP-ABE scheme. Then, Bethencourt, Sahai and Wa-

[1] The principle to apply this method on key-policy attribute-based encryption is the same as the ciphertext-policy case. We do not discuss the key-policy case in this paper.

ters [4] implemented first CP-ABE scheme. In KP-ABE, policy is used to generate secret keys, while policy in CP-ABE is used to build ciphertext.

Subsequently, many expanded ABE branches were well studied as in traditional PKE: New proof method to achieve full secure [14, 15], new key structures to extend attribute universe [16, 17], new ciphertext generation pattern to decrease encryption or decryption time [18–22], new secure model to enhance security and reliability in practical systems [23, 24], new paradigms to hide access policy [25, 26], and new functions to prevent illegal key sharing among colluding users [27]. However, the problem of flexible access control has not been completely solved.

Earlier approach [4, 6] used expiry date to postpone this problem. Pirretti, Traynor, Mcdaniel and Waters [6] suggested that each attribute include an expiry date, 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 [4] presented that the expiry date should be an independent attribute so that different expiry date could be given to different users. Since time changes automatically, the privileges corresponding to specific attributes change as well. A shortage of this approach is that the expiry date must be predefined and the system cannot deal with unexpected changes on user attributes. As a result, the method of adding user ID list into access policy was created as a supplementary approach [7, 8]. This approach does not change access policy but excludes some users although their attributes satisfy the access policy. The shortage of this approach is that its computation and communication costs nearly equal regenerating ciphertexts under a new policy.

Yu, Wang, and Ren [12] proposed an attribute revocation scheme with the proxy re-encryption technique. The authority generates proxy re-keys, and then a proxy server updates secret keys except the users with revoked attributes utilizing the proxy re-keys. Then, data will be re-encrypted as well. Similarly, Sahai, Seyalioglu, and Waters [9] further expressed expiry date as an LSS matrix [11] and constructed new revocable storage systems to exclude users. But the costs of this method are very large. For example, to exclude a user from the storage system, the authority needs to update secret keys for all the other users besides periodically updating ciphertexts. Since transmitting new secret keys to their owners needs many private channels, the costs for the update are larger than generating individual ciphertexts for users.

Yang et al. [10] proposed a multi-authority CP-ABE scheme with attribute revocation function based on the reference [23] and further constructed an ABE-based access control system [13]. Their schemes extend the function of the reference [9]. In [10, 13], data owner also generates update keys. The update keys are transmitted to the cloud server such that the cloud server is able to change data access policy with these keys. A problem is that each time the cloud server updates a ciphertext, new update keys are needed. Transmitting these keys also needs private channels. Thus, the dilemma exists in the reference [9] is still not completely solved. A more unfortunate matter is that the security of the reference [10] is not strong enough. Hong, Xue, and Li [28] pointed out that the scheme in [10] cannot resistant collusion attacks. A revoked user can still decrypt subsequently encrypted data as well.

To summarize, the current access policy/attributes update methods need to generate update keys. Transmitting keys needs additional operations that will increase extra overheads. Thus, an efficient policy update algorithm that does not need to change user privileges by generating update keys is needed for practical applications.

3 Background

In this section, we first describe PU-CP-ABE and discuss the differences and relationship between PU-CP-ABE and CP-ABE. Then, we define the security model for PU-CP-ABE. At last, we introduce the background information on bilinear maps.

3.1 PU-CP-ABE

A CP-ABE scheme consists of four basic algorithms: Setup, KeyGen, Encrypt, and Decrypt. Besides these algorithms, a PU-CP-ABE scheme includes an independent algorithm PolicyUpdate. Setup and KeyGen are executed by the trusted authority of the system. Encrypt and PolicyUpdate are executed by data owners. Decrypt is executed by data recipients. These algorithms are defined as follows:

• **Setup**(1^λ) $\rightarrow$(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 $\mathcal{S}$ that describe the key, the key generation algorithm outputs a private key SK that contains $\mathcal{S}$.

• **Encrypt**(PK,$\mathbb{A}$,M)$\rightarrow$(CT,**IP**). Input PK, a plaintext M, and an access policy $\mathbb{A}$, the encryption algorithm outputs a ciphertext CT that contains $\mathbb{A}$. In addition, some internal parameters IP used in generating CT are recorded.

• **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 $\perp$.

• **PolicyUpdate**(PK,IP,CT,$\mathbb{A}'$)$\rightarrow$(CT$'$,IP$'$). Input PK, IP, a ciphertext CT, and a new access policy $\mathbb{A}'$, the policy update algorithm outputs a new ciphertext CT$'$ corresponding to $\mathbb{A}'$ and adjusts internal parameters IP for future updates.

The inputs of Setup, KeyGen, Encrypt, Decrypt in PU-CP-ABE are the same as in CP-ABE. Only the outputs of Encrypt are a bit different. Some internal parameters IP should be outputted besides a ciphertext in PU-CP-ABE. IP is the only secret input in the policy update algorithm. It includes the state information of the ciphertext. The data owner only needs to create a single file or a database table as in Linux operating system to reserve the IPs corresponding to the ciphertexts he/she may update in future. CP-ABE schemes with the above inputs can be easily upgraded to PU-CP-ABE without any modification. Thus, PU-CP-ABE is compatible with the expanded ABE branches, such as large universe ABE and traceable ABE. Moreover, the correctness conditions of PU-CP-ABE are stricter than CP-ABE as below. In CP-ABE, only the first condition should be satisfied.

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

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

2.If CT$'$ $\leftarrow$PolicyUpdate(PK,IP,CT,$\mathbb{A}'$), and $\mathcal{S}$ satisfies $\mathbb{A}'$, Decrypt(CT$'$,SK) 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 responds corresponding secret key to the adversary. The restriction is that none of the queried sets satisfies $\mathbb{A}^*$.

• **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 flips a random coin $b \in \{0, 1\}$ and returns a ciphertext $\mathrm{CT}_0^{(b)}$ corresponding to M$_b$ and $\mathbb{A}_0$ 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 in query phase 1. For each access policy $\mathbb{A}_i$, the simulator returns corresponding ciphertexts $\mathrm{CT}_i^{(b)}$ to the adversary. 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 satisfies 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 a-gainst 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[$b' = b$]-$\frac{1}{2}$.

3.3 Bilinear Maps

Bilinear map is an important tool to construct CP-ABE scheme. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two cyclic groups of prime order p and g be a generator of $\mathbb{G}$. Define e be a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ with following properties:

1. Bilinearity. For $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy. $e(g, g) \neq 1$.
3. Computability. There is an efficient algorithm to compute $e(u, v)$ for $u, v \in \mathbb{G}$.

4 Policy Update from Linear Secret Sharing

In this section, we first recall the properties of LSS and how to generate an LSS matrix from an access policy. Then we summarize the LSS-based CP-ABE and explain how to embed the LSS matrix into CP-ABE. Finally, our new matrix update algorithm is given.

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 [11] 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 $\{\lambda_i = \overrightarrow{v}W_i\}_{i \in I}$ 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 [9] in 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*". Then, an LSS matrix can be generated from the binary tree with the following algorithm [9, 23]:

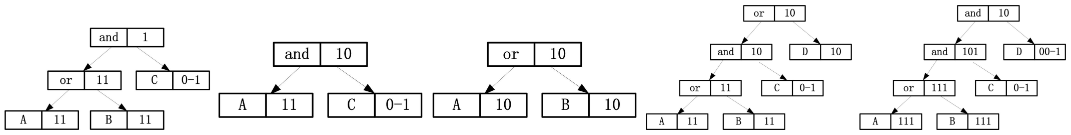
First, $(1, 0, \cdots, 0)$ is used as the sharing vector for the LSS matrix based on the definition of LSS. The matrix generation process begins by labeling the root node of the tree with the vector $\overrightarrow{v}$=(1). The vectors labeled to the other nodes are determined by their parent nodes. Go down the levels of the tree.

Suppose the current node is labeled by a vector $\overrightarrow{v}$. If the node is "*or*", we label its two children by $\overrightarrow{v}$. If the node is "*and*", we label its left child with $\overrightarrow{v}$|1 and its right child with (0,$\cdots$,0|-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 equals that of the longest vector. The other shorter vectors are padded with 0s.

For example, a policy $\mathbb{A}$=((A *or* B) *and* C) can be described in the form of the binary tree as 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 $I = \{i : \rho(i) \in \mathcal{S}\}$ for $\mathcal{S} \supseteq \{A, C\}$ (or $\mathcal{S} \supseteq \{B, C\}$).



(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 LSS Based CP-ABE Schemes

A CP-ABE scheme includes four algorithms: Setup, KeyGen, Encrypt, and Decrypt. The common characteristic of the LSS based CP-ABE schemes [4, 7, 15, 16, 18, 19, 23, 27, 29, 30] is that the ciphertext is generated according to an access policy. Our policy update deals with the case that access policy includes attributes and connectors "*and*" and "*or*". Some earlier schemes [31] do not use the LSS technique can be regarded as a special case that the access policy only includes attributes and connectors "*and*".

Although the ciphertext structures of the above LSS based CP-ABE schemes are different, the ciphertext in all these schemes can be divided into two parts: Common ciphertext components corresponding to the plaintext and ciphertext components corresponding to an attribute. The number of common ciphertext components is irrelevant to the number of attributes in the access policy. Since the number of elements contained in ciphertext components corresponding to an attribute are different in diverse schemes, we use an abstract constant m to represent this number. For example, m=1 in [7], m=2 in [15, 27, 30] and scheme1 of [29], m=3 in [4, 16, 18, 23], and m=n_{max} in scheme3 of [29]. The abstract implementation of the four algorithms are as follows.

Setup. The setup algorithm generates public key PK and master secret key MK for other algorithms.

KeyGen. Given PK, MK, and an attribute set $\mathcal{S}$, the key generation algorithm selects a series of random numbers, computes some common key components, remarked as K_0, K_1. For each attribute $i{\in}\mathcal{S}$, it constructs key components $(K_{2,i}, \cdots, K_{m,i})$. Finally, it outputs SK=$(\mathcal{S}, K_0, K_1, \forall i{\in}\mathcal{S} : \{K_{2,i}, \cdots, K_{m,i}\})$.

Encrypt. Given PK, a message M and an access policy $\mathbb{A} = (W_{l\times n}, \rho)$, the encryption algorithm selects a random vector $\overrightarrow{v} = (s, y_2, y_3, \cdots, y_n)$, and computes $\lambda_i = \overrightarrow{v}\overrightarrow{W}_i$ for each row $i{\in}[1, l]$. It sets common ciphertext components, remarked as C, C_0, and constructs ciphertext components $(C_{1,i}, \cdots, C_{m,i})$ for each row $i{\in}[1, l]$. λ_i should be embedded into some $C_{j,i}$ to share the secret s. Finally, it outputs CT = $(\mathbb{A}, C, C_0, \forall i{\in}[1, l], \{C_{1,i}, \cdots, C_{m,i}\})$.

Decrypt. Given PK, CT, and SK, let $I \subseteq \{1, 2, \cdots, l\}$ be defined as $I = \{i : \rho(i) \in \mathcal{S}\}$. To keep the correctness of the scheme, the decryption algorithm should find a set of constants $\{\omega_i \in Z_p\}_{i\in I}$ such that $\sum_{i\in I} \omega_i\overrightarrow{W}_i = (1, 0, \cdots, 0)$, and computes

$$\begin{aligned}
B_i &= e(C_{1,i}, K_1){\times}e(C_{2,i}, K_{2,\rho(i)}){\times}\cdots{\times}e(C_{m,i}, K_{m,\rho(i)}) \\
&= (e(g, g)^{x\lambda_i} e(g, g)^{x_1}){\times}(e(g, g)^{-x_1} e(g, g)^{x_2}){\times}(e(g, g)^{-x_2} \\
&\quad e(g, g)^{x_3}){\times}\cdots{\times}(e(g, g)^{-x_{m-1}} e(g, g)^{x_m}){\times}(e(g, g)^{-x_m}) \\
&= e(g, g)^{x\lambda_i} \\
B &= \prod_{i\in I}(B_i)^{\omega_i} = e(g, g)^{x\overrightarrow{v}\sum_{i\in I}\omega_i W_i} = e(g, g)^{xs}
\end{aligned}$$

Finally, it outputs M with B and the common components C, C_0, K_0. Here, $x, x_1, \cdots, x_m$ are some existing but unknown numbers. All these numbers are mutual cancellation in decryption algorithm.

Our policy update does not rely on the details of the encryption algorithm. But $\lambda_1, \cdots, \lambda_l$ should be embedded into ciphertext and can be correctly recovered in decryption algorithm such that the correctness condition can be satisfied.

4.4 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. This goal is to ensure the distribution of the shared secret embedded into the ciphertext outputted by the policy update algorithm is same as that of the secret directly selected in the encryption algorithm. There are 7 possible operations we can image to update 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)

5. Change a connector "*and*" to "*or*", e.g.(A *and* B)→ (A *or* B).
6. Change a connector "*or*" to "*and*", e.g.(A *or* B)→ (A *and* B).
7. Change an attribute to another, e.g.(A *or* B)→ (A *or* C), and (A *and* B)→ (A *and* C).

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 $(0,\cdots,0|\text{-}1)$ and $(0,\cdots,0|1)$ is added to its sibling. The length of 0s equals the longest vector.

For case 5), to change a connector "*or*" to "*and*", we first delete an attribute connected by "*or*" as in case 1) and then add this attribute as in case 4).

For case 6), to change a connector "*and*" to "*or*", we first delete an attribute connected by "*and*" as in case 2) and then add this attribute as in case 3).

For case 7), to change an attribute to another, we first delete the attribute as in case 1) or case 2) and then add new attribute as in case 3) or case 4).

Apparently, case 1) to case 4) have covered all the possible operations.

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 or delete an attribute from the current policy. The attributes corresponding to these vectors can be recorded.

Continuing from the above example, Fig.1(b)-(e) show the new binary trees corresponding to the four cases of changing from $\mathbb{A}$, and the following LSS matrices show the relationship between the four new policies and $\mathbb{A}$.

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

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

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 Construction

In this section, we construct our PU-CP-ABE scheme with the abstract CP-ABE scheme in Section 4.3 and the matrix update method in Section 4.4. We call the abstract CP-ABE scheme BASE (Setup, KeyGen, Encrypt, Decrypt). Our construction is as follows.
Setup. Run BASE.Setup and output PK and MK.
KeyGen. Given PK, MK, and an attribute set $\mathcal{S}$, run BASE.KeyGen and output SK.
Encrypt. Given PK, M and an access policy $\mathbb{A} = (W_{l\times n}, \rho)$, run BASE.Encrypt and output CT. In addition, $(\lambda_1, \cdots, \lambda_l)$ are recorded as internal parameters IP.
Decrypt. Given PK, CT, and SK, run BASE.Decrypt to obtain M.
PolicyUpdate. Given PK, IP, CT, and a new access policy $\mathbb{A}'$, this algorithm updates CT to CT$'$ according to $\mathbb{A}'$ and updates IP.

First, it decomposes the process of transforming $\mathbb{A}$ to $\mathbb{A}'$ to a series of atomic updates. Each atomic update only changes one attribute. E.g. Let $\mathbb{A}$=((A *and* B) *and* C) and $\mathbb{A}'$=((A *or* B) *and* D), the transformation from $\mathbb{A}$ to $\mathbb{A}'$ includes following 4 atomic updates:

Atomic update 1.((A *and* B) *and* C)→(A *and* B).

Atomic update 2.(A *and* B)→A.

Atomic update 3.A$\rightarrow$(A *or* B).

Atomic update 4.(A *or* B)$\rightarrow$((A *or* B) *and* D).

Then, the operations for each atomic update are as follows:

Case 1): If $\mathbb{A}$=($\mathbb{A}'$ *or* A), to delete A from $\mathbb{A}$, removing corresponding ciphertext components $(C_{1,A}, \cdots, C_{m,A})$ from CT and deleting λ_A from IP.

Case 2): If $\mathbb{A}$=($\mathbb{A}'$ *and* A), to delete A from $\mathbb{A}$, removing corresponding ciphertext components $(C_{1,A}, \cdots, C_{m,A})$ from CT and deleting λ_A from IP at first.

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 4.4. For each attribute $i \in N(\mathbb{A}, A)$, 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 $C_{j,i} = C'_{j,i} \times C_{j,i}$.
3. Replace λ_i with $\lambda_i + \lambda_A$ in IP.

Case 3): If $\mathbb{A}'$=($\mathbb{A}$ *or* A), to add A to $\mathbb{A}$, computing $\lambda_{\mathbb{A}}$ 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. Finally, λ_A is added into IP.

Case 4): If $\mathbb{A}'$=($\mathbb{A}$ *and* A), to add A to $\mathbb{A}$, selecting random number $y_A \in \mathbb{Z}_p$ and defining $\lambda_A = -y_A$.

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 4.4. For each attribute $i \in N(\mathbb{A}, A)$, 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 $C_{j,i} = C'_{j,i} \times C_{j,i}$.
3. Replace λ_i with $\lambda_i + y_A$ in IP.

In addition, new ciphertext components $(C_{1,A}, \cdots, C_{m,A})$ are generated with BASE.Encrypt and λ_A is added into IP.

Other possible cases, such as case 5), case 6), and case 7) in Section 4.4, can be covered by the above cases. Finally new ciphertext CT$'$ according to $\mathbb{A}'$ 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, running BASE.decrypt will obtain M.

2.Given CT$'$ that includes $\mathbb{A}' = (W', \rho)$, which is generated by the policy update 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 $\{\omega_i \in \mathbb{Z}_p\}_{i \in I'}$, such that $\sum_{i \in I'} \omega_i \overrightarrow{W}'_i = (1, 0, \cdots, 0)$.

Based on the update times for the ciphertext, I' can be regarded as many subsets $(I_1, I_2, \cdots)$. A subset I_q includes the attributes, whose associated ciphertext components are updated for q times. For each attribute $i \in I_q$, $(C_{1,i}, \cdots, C_{m,i})$ are the corresponding ciphertext components and $\overrightarrow{W}_i$ is the corresponding vector. For each update $j \in [1, q]$ on attribute i, a vector $\overrightarrow{W}_{A_j}$ is added on $\overrightarrow{W}_i$. Thus, we have $\overrightarrow{W}'_i = \overrightarrow{W}_i + \overrightarrow{W}_{A_1} + \cdots + \overrightarrow{W}_{A_q}$.

$$\begin{aligned}
B_i &= e(C_{1,i}, K_1) \times e(C_{2,i}, K_{2,\rho(i)}) \times \cdots \times e(C_{m,i}, K_{m,\rho(i)}) \\
&= (e(g,g)^{x\lambda_i} e(g,g)^{x_1}) \times (e(g,g)^{-x_1} e(g,g)^{x_2}) \times (e(g,g)^{-x_2} \\
&\quad e(g,g)^{x_3}) \times \cdots \times (e(g,g)^{-x_{m-1}} e(g,g)^{x_m}) \times (e(g,g)^{-x_m}) \\
&\times (e(g,g)^{x\lambda_{A_1}} e(g,g)^{x_1^{(A_1)}}) \times (e(g,g)^{-x_1^{(A_1)}} e(g,g)^{x_2^{(A_1)}}) \times \\
&\quad (e(g,g)^{-x_2^{(A_1)}} e(g,g)^{x_3^{(A_1)}}) \times \cdots \times (e(g,g)^{-x_m^{(A_1)}}) \times \cdots \\
&\times (e(g,g)^{x\lambda_{A_q}} e(g,g)^{x_1^{(A_q)}}) \times (e(g,g)^{-x_1^{(A_q)}} e(g,g)^{x_2^{(A_q)}}) \times \\
&\quad (e(g,g)^{-x_2^{(A_q)}} e(g,g)^{x_3^{(A_q)}}) \times \cdots \times (e(g,g)^{-x_m^{(A_q)}}) \\
&= e(g,g)^{x(\lambda_i + \lambda_{A_1} + \cdots + \lambda_{A_q})} \\
&= e(g,g)^{xv(\overrightarrow{W}_i + \overrightarrow{W}_{A_1} + \cdots + \overrightarrow{W}_{A_q})} \\
&= e(g,g)^{xv\overrightarrow{W}'_i} \\
&= e(g,g)^{x\lambda'_i}
\end{aligned}$$

Thus, the distribution of the secret embedded into the ciphertext outputted by the policy update algorithm is same as that of the secret directly selected in the encryption algorithm. Then

$$B = \prod_{i \in I} (B_i)^{\omega_i} = e(g,g)^{x\overrightarrow{v} \sum_{i \in I'} \omega_i W'_i} = e(g,g)^{xs}$$

Finally, M can be obtained with B and other common ciphertext and key components.

6 Example and Usage 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 [16]. The example shows how to use PU-CP-ABE intuitively. The details are as follows.

A system administer first publishes PK=$(g, u, h, w, v,\ e(g,g)^\alpha)$ with **Setup**, and keeps MK=(g^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 then computes SK1 as follows:

$$S1, K_0 = g^\alpha w^r, K_1 = g^r,$$

$$K_{2,A} = g^{r_A}, K_{3,A} = (u^A h)^{r_A} v^{-r},$$
$$K_{2,C} = g^{r_C}, K_{3,C} = (u^C h)^{r_C} v^{-r}$$

For S_2, it selects another group of random numbers r, r_A, r_B, r_D, and then computes SK2 as follows:

$$\begin{aligned}
&S2, K_0 = g^{\alpha} w^r, K_1 = g^r,\\
&K_{2,A} = g^{r_A}, K_{3,A} = (u^A h)^{r_A} v^{-r},\\
&K_{2,B} = g^{r_B}, K_{3,B} = (u^B h)^{r_B} v^{-r},\\
&K_{2,D} = g^{r_D}, K_{3,D} = (u^D h)^{r_D} v^{-r}
\end{aligned}$$

Finally, SK1 and SK2 are distributed to associated users.

Define policy $\mathbb{A}$ = ((A *or* B) *and* C). A data owner first encrypts a document M under $\mathbb{A}$. 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 random vector $\overrightarrow{v} = (s, y_2)$, and 3 random numbers t_1, t_2, and t_3, compute $\lambda_1 = \overrightarrow{v}\overrightarrow{W}_1 = s + y_2$, $\lambda_2 = \overrightarrow{v}\overrightarrow{W}_2 = s + y_2$, $\lambda_3 = \overrightarrow{v}\overrightarrow{W}_3 = -y_2$.

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

CT is uploaded on a cloud server. Then, the data owner (or the cloud administer) broadcasts a new data generation notification for CT. 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 $\omega = (1, 0, 1, 0)$

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

Next, the data owner wants to change 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 $\mathbb{A}'$. The ciphertext is generated as follows:

Phase $\mathbb{A}$ to matrix $(W_{2\times 3}, \rho) = (\overrightarrow{W}_1 = (1,1), \overrightarrow{W}_2 = (0,-1), \overrightarrow{W}_4 = (1,0), \rho(1) = A, \rho(2) = B, \rho(4) = D)$. Select a random vector $\overrightarrow{v} = (s, y_2)$, and 3 random numbers t_1, t_2, and t_4, compute $\lambda_1 = \overrightarrow{v}\overrightarrow{W}_1 = s + y_2$, $\lambda_2 = \overrightarrow{v}\overrightarrow{W}_2 = -y_2$, $\lambda_4 = \overrightarrow{v}\overrightarrow{W}_4 = s$.

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

On the other hand, he is able to update CT to fit for $\mathbb{A}'$ as well. The processes of the 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). Given CT and IP, $(C_{1,3}, C_{2,3}, C_{3,3})$ are removed and λ_3 is deleted. Both A and B are affected when deleting C. We have $N(((A$ or $B)$ and $C)$, $C)$ =(A, B). For attributes A, B, select random numbers $t_1^{(C)}$ and $t_2^{(C)}$, and compute

$$\begin{aligned}
&C_{1,1} = w^{(\lambda_1 = s+y_2)}v^{t_1}w^{(\lambda_3 = -y_2)}v^{t_1^{(C)}},\\
&C_{2,1} = (u^{\rho(1)}h)^{-t_1}(u^{\rho(1)}h)^{-t_1^{(C)}}, C_{3,1} = g^{t_1}g^{t_1^{(C)}},\\
&C_{1,2} = w^{(\lambda_2 = s+y_2)}v^{t_2}w^{(\lambda_3 = -y_2)}v^{t_2^{(C)}},\\
&C_{2,2} = (u^{\rho(2)}h)^{-t_2}(u^{\rho(2)}h)^{-t_2^{(C)}}, C_{3,2} = g^{t_2}g^{t_2^{(C)}}
\end{aligned}$$

Finally, we have

$$\begin{aligned}
CT_1 = (&\mathbb{A}_1 = (A\ or\ B), C = Me(g,g)^{\alpha s}, C_0 = g^s,\\
&C_{1,1} = w^{(\lambda_1 = s+y_2)}v^{t_1}w^{(\lambda_3 = -y_2)}v^{t_1^{(C)}},\\
&C_{2,1} = (u^{\rho(1)}h)^{-t_1}(u^{\rho(1)}h)^{-t_1^{(C)}}, C_{3,1} = g^{t_1}g^{t_1^{(C)}},\\
&C_{1,2} = w^{(\lambda_2 = s+y_2)}v^{t_2}w^{(\lambda_3 = -y_2)}v^{t_2^{(C)}},\\
&C_{2,2} = (u^{\rho(2)}h)^{-t_2}(u^{\rho(2)}h)^{-t_2^{(C)}}, C_{3,2} = g^{t_2}g^{t_2^{(C)}})
\end{aligned}$$

$$IP_1 = (\lambda_1 = \lambda_1 + \lambda_3 = s, \lambda_2 = \lambda_2 + \lambda_3 = s)$$

2. Atomic update 2 satisfies case 1). Given CT$_1$ and IP$_1$, $(C_{1,2}, C_{2,2}, C_{3,2})$ are removed and λ_2 is deleted. Finally, we have

$$\begin{aligned}
CT_2 = (&\mathbb{A}_2 = (A), C = Me(g,g)^{\alpha s}, C_0 = g^s,\\
&C_{1,1} = w^{(\lambda_1 = s+y_2)} v^{t_1} w^{(\lambda_3 = -y_2)} v^{t_1^{(C)}},\\
&C_{2,1} = (u^{\rho(1)}h)^{-t_1} (u^{\rho(1)}h)^{-t_1^{(C)}}, C_{3,1} = g^{t_1} g^{t_1^{(C)}})\\
IP_2 = (&\lambda_1 = s)
\end{aligned}$$

3. Atomic update 3 satisfies case 4). Given CT$_2$ and IP$_2$, select a random number $y_2' \in \mathbb{Z}_p$ and set $\lambda_2 = -y_2'$. $N((A), B)$=(A). Select a random number $t_1^{(B)}$ for attribute A, and compute

$$\begin{aligned}
&C_{1,1} = w^{(\lambda_1 = s+y_2)} v^{t_1} w^{(\lambda_3 = -y_2)} v^{t_1^{(C)}} w^{y_2'} v^{t_1^{(B)}},\\
&C_{2,1} = (u^{\rho(1)}h)^{-t_1} (u^{\rho(1)}h)^{-t_1^{(C)}} (u^{\rho(1)}h)^{-t_1^{(B)}},\\
&C_{3,1} = g^{t_1} g^{t_1^{(C)}} g^{t_1^{(B)}}
\end{aligned}$$

Next, compute new ciphertext components

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

Finally, we have

$$\begin{aligned}
CT_3 = (&\mathbb{A}_3 = (A \text{ } and \text{ } B), C = Me(g,g)^{\alpha s}, C_0 = g^s,\\
&C_{1,1} = w^{(\lambda_1 = s+y_2)} v^{t_1} w^{(\lambda_3 = -y_2)} v^{t_1^{(C)}} w^{y_2'} v^{t_1^{(B)}},\\
&C_{2,1} = (u^{\rho(1)}h)^{-t_1} (u^{\rho(1)}h)^{-t_1^{(C)}} (u^{\rho(1)}h)^{-t_1^{(B)}},\\
&C_{3,1} = g^{t_1} g^{t_1^{(C)}} g^{t_1^{(B)}},\\
&C_{1,2} = w^{(\lambda_2 = -y_2')} v^{t_2^{(B)}}, C_{2,2} = (u^{\rho(2)}h)^{-t_2^{(B)}}, C_{3,2} = g^{t_2^{(B)}})\\
IP_3 = (&\lambda_1 = s + y_2', \lambda_2 = -y_2')
\end{aligned}$$

4. Atomic update 4 satisfies case 3). Given CT$_3$ and IP$_3$, compute $\lambda_{(A \text{ } and \text{ } B)} = \lambda_1 + \lambda_2 = s$, set $\lambda_4 = \lambda_{(A \text{ } and \text{ } B)} = s$. Then, select $t_4^{(D)} \in \mathbb{Z}_p$ and compute new ciphertext components

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

Finally, we have

$$CT_4 = (\mathbb{A}_4 = ((A \text{ } and \text{ } B) \text{ } or \text{ } D), C = Me(g,g)^{\alpha s}, C_0 = g^s,$$

$$\begin{aligned}
&C_{1,1} = w^{(\lambda_1 = s + y_2)} v^{t_1} w^{(\lambda_3 = -y_2)} v^{t_1^{(C)}} w^{y_2'} v^{t_1^{(B)}}, \\
&C_{2,1} = (u^{\rho(1)} h)^{-t_1} (u^{\rho(1)} h)^{-t_1^{(C)}} (u^{\rho(1)} h)^{-t_1^{(B)}}, \\
&C_{3,1} = g^{t_1} g^{t_1^{(C)}} g^{t_1^{(B)}}, \\
&C_{1,2} = w^{(\lambda_2 = -y_2')} v^{t_2^{(B)}}, C_{2,2} = (u^{\rho(2)} h)^{-t_2^{(B)}}, C_{3,2} = g^{t_2^{(B)}}, \\
&C_{1,4} = w^{(\lambda_4 = s)} v^{t_4^{(D)}}, C_{2,4} = (u^{\rho(2)} h)^{-t_4^{(D)}}, C_{3,4} = g^{t_4^{(D)}}) \\
IP_4 &= (\lambda_1 = s + y_2', \lambda_2 = -y_2', \lambda_4 = s)
\end{aligned}$$

At last, CT′=CT$_4$ is outputted to replace CT and IP′=IP$_4$ is kept for future updates. CT′ can be normalized as below form.

$$\begin{aligned}
CT' &= (\mathbb{A}' = ((A\ and\ B)\ or\ D), C = Me(g,g)^{\alpha s}, C_0 = g^s, \\
&C_{1,1} = w^{s + y_2'} v^{(t_1 + t_1^{(B)} + t_1^{(C)})}, C_{2,1} = (u^{\rho(1)} h)^{-(t_1 + t_1^{(B)} + t_1^{(C)})}, \\
&C_{3,1} = g^{(t_1 + t_1^{(B)} + t_1^{(C)})}, \\
&C_{1,2} = w^{-y_2'} v^{t_2^{(B)}}, C_{2,2} = (u^{\rho(2)} h)^{-t_2^{(B)}}, C_{3,2} = g^{t_2^{(B)}}, \\
&C_{1,4} = w^s v^{t_4^{(D)}}, C_{2,4} = (u^{\rho(2)} h)^{-t_4^{(D)}}, C_{3,4} = g^{t_4^{(D)}}) \\
IP_4 &= (\lambda_1 = s + y_2', \lambda_2 = -y_2', \lambda_4 = s)
\end{aligned}$$

We can regard $t_1 + t_1^{(B)} + t_1^{(C)}$ as a random number t_1. *The distribution of CT' (=CT_4) outputted by the policy 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 SK1. It means that the users with attributes S_1 = (A, C) had lost the privilege to obtain data after the update. Meanwhile, the users holding SK2 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 $\omega = (1, 1, 0, 0)$.

$$\begin{aligned}
B_1 &= e(C_{1,1}, K_1) e(C_{2,1}, K_{2,\rho(1)}) e(C_{3,1}, K_{3,\rho(1)}) \\
&= e(w^{(\lambda_1 = s + y_2)} v^{t_1} w^{(\lambda_3 = -y_2)} v^{t_1^{(C)}} w^{y_2'} v^{t_1^{(B)}}, g^r) \\
&\times\ e((u^{\rho(1)} h)^{-t_1} (u^{\rho(1)} h)^{-t_1^{(C)}} (u^{\rho(1)} h)^{-t_1^{(B)}}, g^{r_{\rho(1)}}) \\
&\times\ e(g^{t_1} g^{t_1^{(C)}} g^{t_1^{(B)}}, (u^{\rho(1)} h)^{r_{\rho(1)}} v^{-r}) \\
&= e(w,g)^{r(s + y_2')} e(v^{t_1} v^{t_1^{(C)}} v^{t_1^{(B)}}, g^r) e((u^A h)^{-t_1} (u^A h)^{-t_1^{(C)}} \\
&\times\ (u^A h)^{-t_1^{(B)}}, g^{r_A}) e(g^{t_1} g^{t_1^{(C)}} g^{t_1^{(B)}}, (u^A h)^{r_A} v^{-r}) \\
&= e(w,g)^{r(s + y_2')}
\end{aligned}$$

$$
\begin{aligned}
B_2 &= e(C_{1,2}, K_1)e(C_{2,2}, K_{2,\rho(2)})e(C_{3,2}, K_{3,\rho(2)}) \\
&= e(w^{(\lambda_2=-y'_2)}v^{t_2^{(B)}}, g^r)e((u^{\rho(2)}h)^{-t_2^{(B)}}, g^{r_{\rho(2)}}) \\
&\times\ e(g^{t_2^{(B)}}, (u^B h)^{r_{\rho(2)}}v^{-r}) \\
&= e(w,g)^{-y'_2}e(v^{t_2^{(B)}}, g^r)e((u^B h)^{-t_2^{(B)}}, g^{r_B}) \\
&\times\ e(g^{t_2^{(B)}}, (u^B h)^{r_B}v^{-r}) \\
&= e(w,g)^{-ry'_2} \\
B &= (B_1B_2) = e(w,g)^{r(s+y'_2-y'_2)} = e(w,g)^{rs} \\
\frac{C{\cdot}B}{e(C_0,K_0)} &= \frac{Me(g,g)^{\alpha s}e(w,g)^{rs}}{e(g^s, g^{\alpha}w^r)} \Rightarrow M
\end{aligned}
$$

Eventually, the access policy of the document is flexibly changed by the data owner. The updating time does not need to be predefined in encryption algorithm or key generation algorithm.

7 Security Analysis of PU-CP-ABE

In this section, we discuss the security of PU-CP-ABE. The example in Section 6 intuitively shows that PolicyUpdate can be constructed by Encrypt and the distribution of the ciphertext generated by PolicyUpdate is the same as the distribution of the ciphertext generated by Encrypt. Actually, we can obtain the same conclusion in the formal construction of Section 5.

PolicyUpdate deals with 4 cases of attributes. The case 1) only removes some components of the ciphertext, the rest part does not change. The case 2), case 3), and case 4) call Encrypt to generate the ciphertext components $\mathrm{C}'_{j,i}$s for $\mathrm{C}_{j,i}$s that need updating, and accumulate these new components to original ciphertext. Since $\mathrm{C}_{j,i}$s are also generated by Encrypt, the structures of $\mathrm{C}'_{j,i}$s and $\mathrm{C}_{j,i}$s are same. For example, let $\mathrm{C}_{j,i}{=}g^a$, $\mathrm{C}'_{j,i}{=}g^b$. We have $\mathrm{C}_{j,i} \times \mathrm{C}'_{j,i} = g^{a+b}$. Obviously, the distributions of g^a and g^{a+b} are same. Note, an element $g_1^{a_1}g_2^{a_2}\cdots g_x^{a_x}$ can be abstracted as $(g^{c_1})^{a_1}(g^{c_2})^{a_2}\cdots(g^{c_x})^{a_x}$. Define $a_1{\times}c_1{+}\cdots{+}a_x{\times}c_x$ as an known number a, any group element has the form of g^a.

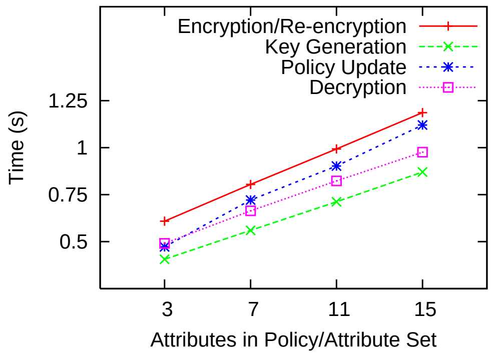
Thus, if a challenger is able to simulate the encryption algorithm to obtain a ciphertext, he/she can also simulate the policy update algorithm to generate a ciphertext with the same distribution. Intuitively, PU-CP-ABE is secure if the basic CP-ABE is secure. According to the usages of the basic CP-ABE in the security game, there are two kinds of proof approaches.

First, the basic CP-ABE is regarded as an independent scheme. For this case, there are three participators in the security game: An adversary $\mathcal{A}$, a simulator $\mathcal{B}$, and a challenger $\mathcal{C}$. The basic CP-ABE scheme is simulated by $\mathcal{C}$. $\mathcal{B}$ passes the queries from $\mathcal{A}$ on to $\mathcal{C}$ and the ciphertext is actually simulated by $\mathcal{C}$ and no internal parameters are returned to

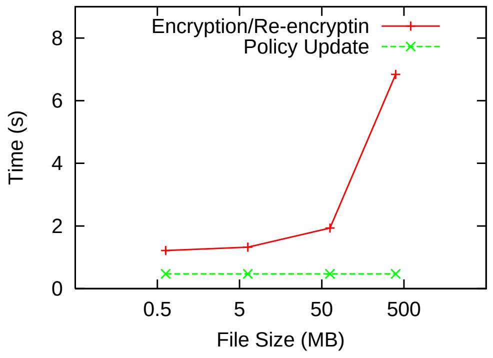
$\mathcal{B}$. As a result, $\mathcal{B}$ cannot simulate the ciphertext of the policy update algorithm without IP. However, $\mathcal{B}$ is able to directly forward the query access policy to $\mathcal{C}$ and pass the result ciphertext from $\mathcal{C}$ on to $\mathcal{A}$ as the response. That is because the distribution of the ciphertexts generated by the policy update algorithm of PU-CP-ABE and the encryption algorithm of the basic CP-ABE are same. $\mathcal{A}$ cannot distinguish them. Thus, if $\mathcal{A}$ breaks PU-CP-ABE, $\mathcal{B}$ is able to break the basic CP-ABE. Then if $\mathcal{B}$ breaks the basic CP-ABE, $\mathcal{C}$ will break the security assumption of the basic CP-ABE.

Second, the basic CP-ABE is regarded as parts of PU-CP-ABE. For this case, there are two participators in the security game: An adversary $\mathcal{A}$ and a challenger $\mathcal{B}$. After receiving queries from $\mathcal{A}$, $\mathcal{B}$ calls the encryption algorithm of the basic CP-ABE to simulate the ciphertext and records the internal parameters IP. Then, for the four cases of updates, $\mathcal{B}$ is able to call the encryption algorithm of the basic CP-ABE with new access policies and accumulates the ciphertext components to corresponding components of the last ciphertext. As a result, $\mathcal{B}$ simulates the ciphertext of the policy update algorithm. If $\mathcal{A}$ breaks PU-CP-ABE, $\mathcal{B}$ will break the security assumption of the basic CP-ABE.

The essence of the two proof approaches is same. Theorem 1 and Theorem 2 will describe the details of the two proof approaches respectively.



(a) Scheme1 of [29] with Policy Update



(b) Hybird Encryption with AES 256

Fig. 2. Executing Time of PU-CP-ABE

Theorem 1. If the basic CP-ABE scheme is secure with respect to Definition 1, all polynomial time adversaries have at most a negligible advantage in breaking PU-CP-ABE.

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

Init. $\mathcal{B}$ receives the challenge access policy $\mathbb{A}^* = (W_{l^* \times n^*}, \rho^*)$ and sends it to the challenger $\mathcal{C}$ of the basic CP-ABE scheme BASE.

Setup. $\mathcal{B}$ receives the public key PK from $\mathcal{C}$ and sends it to $\mathcal{A}$.

Query phase 1. For each secret key query from $\mathcal{A}$, $\mathcal{B}$ passes it on to $\mathcal{C}$ and returns the key constructed by $\mathcal{C}$ to $\mathcal{A}$.
Challenge. After receiving two plaintexts (M_0, M_1) and an access policy $\mathbb{A}_0 = (W_{l_0 \times n_0}, \rho)$ from $\mathcal{A}$, $\mathcal{B}$ sends them to $\mathcal{C}$. $\mathcal{C}$ flips a random coin $b \in \{0,1\}$ and simulates a ciphertext $\mathrm{CT}_0^{(b)} = (\mathbb{A}_0, C, C_0, \forall j \in [1, l_0], \{C_{1,j}, \cdots, C_{m,j}\})$ with M_b. $\mathrm{CT}_0^{(b)}$ is returned to $\mathcal{B}$ and then is forwarded to $\mathcal{A}$. If $\mathbb{A}_0 = \mathbb{A}^*$, the challenge ciphertext has been simulated. Otherwise, $\mathcal{B}$ continues to forwards $\mathbb{A}_i$ to $\mathcal{C}$. $\mathcal{C}$ simulates corresponding ciphertext $\mathrm{CT}_i^{(b)} = (\mathbb{A}_i, C, C_0, \forall j \in [1, l_i], \{C_{1,j}, \cdots, C_{m,j}\})$. $\mathrm{CT}_i^{(b)}$ is returned back to $\mathcal{B}$ and then is passed on to $\mathcal{A}$. Finally, $\mathcal{A}$ obtains $\mathrm{CT}_0^{(b)} \cdots \mathrm{CT}_n^{(b)}$. Here, we require that the common components, C, C_0, in all the ciphertexts are same. This constraint is easy to satisfy because these components are simulated utilizing designate numbers produced in the setup phase and given terms of the security assumption, such as in [16] and scheme1 of [29].
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 the security game with a non-negligible advantage, $\mathcal{B}$ has same advantage in breaking the basic CP-ABE scheme.

Theorem 2. If the security assumption of the basic CP-ABE scheme holds, all polynomial time adversaries have at most a negligible advantage in breaking PU-CP-ABE.

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 assumption 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 SK. 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 $\mathbb{A} = (W_{l \times n}, \rho)$, $\mathcal{B}$ first selects a random vector $\overrightarrow{v} = (s, y_2, y_3, \cdots, y_n)$, computes $\lambda_i = \overrightarrow{v}\overrightarrow{W}_i$ for each row $i \in [1, l]$. IP_0 is $(\lambda_1, \lambda_2, \cdots, \lambda_l)$. Then $\mathcal{B}$ calls BASE.Encrypt with the inputs M_b, s, IP_0 and obtains the simulation result $\mathrm{CT}_0^{(b)} = (\mathbb{A}_0, C, C_0, \forall j \in [1, l], \{C_{1,j}, \cdots, C_{m,j}\})$. $\mathrm{CT}_0^{(b)}$ is returned to $\mathcal{A}$. If $\mathbb{A}_0 = \mathbb{A}^*$, the challenge ciphertext has been simulated. Otherwise, $\mathcal{B}$ continues to receive policy update queries. The simulation for PolicyUpdate can be decomposed to a series of atomic updates.

For an atomic update that an attribute A is added into or deleted from the current policy, $\mathcal{B}$ simulates the four cases of PolicyUpdate(PK,IP_{i-1},$\mathrm{CT}_{i-1}^{(b)}$,$\mathbb{A}_i$). The common components, such as C, C_0, do not need to simulate again. Thus, M_b and s do not need providing. The ciphertext components that are the same as in $\mathrm{CT}_{i-1}^{(b)}$ do not need to simulate as well. $\mathcal{B}$ only needs to tell BASE which ciphertext components need updating and what the corresponding λs are.

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 $N(\mathbb{A}, A)$. Call BASE.Encrypt with λ_A and $N(\mathbb{A}, A)$. $\{C_{1,j}, \cdots, C_{m,j}\}$ for $j \in N(\mathbb{A}, A)$ are simulated and accumulated on corresponding components of $\mathrm{CT}_{i-1}^{(b)}$.

For case 3), compute $\lambda_{\mathcal{A}}$. Call BASE.Encrypt with $\lambda_A = \lambda_{\mathcal{A}}$ and receive $\{C_{1,A}, \cdots, C_{m,A}\}$.

For case 4), select random number $y_A \in \mathbb{Z}_p$, set $\lambda_A = -y_A$, and find the set of attribute $N(\mathbb{A}, A)$. Call BASE.Encrypt with y_A, λ_A and $N(\mathbb{A}, A)$. $\{C_{1,A}, \cdots, C_{m,A}\}$, and $\{C_{1,j}, \cdots, C_{m,j}\}$ for $j \in N(\mathbb{A}, A)$ are simulated and accumulated on corresponding components of $\mathrm{CT}_{i-1}^{(b)}$.

As a result, $\mathrm{CT}_i^{(b)}$ is simulated by $\mathcal{B}$ and is returned to $\mathcal{A}$. We also require that the common components, C, C_0, in all the ciphertexts are same as in Theorem 1.

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 basic CP-ABE scheme.

8 Performance Analysis

In this section, we implement PU-CP-ABE and test its efficiency. We apply the policy update algorithm on a well known CP-ABE scheme, Scheme1 of [29]. The codes are programmed with the C language on Microsoft VS 2005 and uses PBC library v0.5.14 [32]. PBC provides basic pairing functions. We implement the LSS matrix generation and update algorithms and construct PU-CP-ABE with them. 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 $\mathbb{G}_T$. Since the operations in $\mathbb{G}_1$ are faster than the operations in $\mathbb{G}_2$, more operations are executed in $\mathbb{G}_1$. In addition, a 256-bit session key is extracted from an element in $\mathbb{G}_T$ with the base64 coding to encrypt files under AES algorithm. The test machine is a ThinkPad X1 laptop with Intel Core i5-3337U 1.8GHz and 64-bit Microsoft Windows 7 operating system.

First, we test the time of the algorithms in PU-CP-ABE. We execute the encryption algorithm, the key generation algorithm, the policy update algorithm, and the decryption algorithm under the scenarios that 3, 7, 11, 15 attributes are in the input access policy/attribute set, respectively. Fig.2(a) shows the result. The time of the encryption algorithm, the key generation algorithm and the policy update algorithm includes the time of serializing ciphertext or secret key to text files. The time of the decryption algorithm includes the times of deserializing ciphertext and secret key to memory. The number of attributes in the policy 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 number of attributes in the access policy of the encryption algorithm. That is because updating ciphertext components corresponding to 1 attribute of the old policy is able to change the decryption privilege of the ciphertext but setting the initial decryption privilege needs generating the whole ciphertext. Viewing from Fig.2(a), the policy update time is still small than the encryption time.

Next, 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 encryption key is extracted from the plaintext of PU-CP-ABE with the base64 coding. Then the plaintext 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.2(b). Viewing from the figure, the time of updating files access policy is small than the time of generating/regenerating files and setting their access policy. Moreover, with the increase of file size, the encryption time is increased linearly while the update time is nearly not increased. Thus, PU-CP-ABE is very efficient. We believe that PU-CP-ABE has been a practical access control tool.

9 Conclusions

In this paper, we propose a policy update method for ciphertext-policy attribute-based encryption. Our method allows data owners to change the access policy after the ciphertext has been generated. The update does not rely on particular schemes and can be proved secure. Our scheme is efficient as well.

References

1. W. Diffie, M. Hellman, New directions in cryptography, IEEE Transactions on Information Theory, vol.22, no.6, pp.644-654, 1976.
2. R. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signa-tures and public-key cryptosystems, Communications of ACM vol.21, no.2, pp.120-126, 1978.
3. A. Sahai and B. Waters, Fuzzy identity based encryption, EUROCRYPT 2005, LNCS 3494, pp.457-473, Springer, Heidelberg, 2005.
4. J. Bethencourt, A. Sahai and B. Waters, Ciphertext-Policy Attribute-Based Encryption, IEEE S&P 2007, pp.321-334, 2007.
5. V. Goyal, O. Pandey, A. Sahai, and B. Waters, Attribute-Based encryption for Fine-Grained Access Control of Encrypted Data, ACM CCS 2006, pp.89-98, Virginia, 2006.
6. M. Pirretti, P. Traynor, P. Mcdaniel and B. Waters, Secure attribute-based systems, ACM CCS 2006, pp.99-112, 2006.
7. N. Attrapadung and H. Imai, Conjunctive Broadcast and Attribute-Based Encryption, Pairing 2009, LNCS 5671, pp.248-265, Springer, Heidelberg, 2009.

8. A. Lewko, A. Sahai and B. Waters, Revocation Systems with Very Small Private Keys, IEEE S&P 2010, pp.273-285, 2010.
9. A. Sahai, H. Seyalioglu and B. Waters, Dynamic Credentials and Ciphertext Delegation for Attribute-based Encryption, CRYPTO 2012, LNCS 7417, pp.199-217, 2012.
10. K. Yang, X.H. Jia, K. Ren, B. Zhang, R.T. Xie, 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.
11. A. Beimel, Secure Schemes for Secret Sharing and Key Distribution, PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1996.
12. S. Yu, C. Wang, K. Ren, Attribute Based Data Sharing with Attribute Revocation, ACM ASIACCS 2010,pp.261-270, 2010.
13. K. Yang, X.H. Jia, K. Ren, R.T. Xie and L.S. Huang, Enabling Efficient Access Control with Dynamic Policy Updating for Big Data in the Cloud, INFOCOM 2014, pp.2013-2021, 2014.
14. A. Lewko and B. Waters, New techniques for dual system encryption and fully secure hibe with short ciphertexts, TCC 2010, pp.455-479, 2010.
15. A. Lewko and B. Waters, New proof methods for attribute based encryption: Achieving full security through selective techniques, CRYPTO 2012, pp.180-198, 2012.
16. Y. Rouselakis and B. Waters, Practical constructions and new proof methods for large universe attribute-based encryption, ACM CCS 2013, pp.463-474, 2013.
17. R. Ostrovsky, A. Sahai, and B. Waters, Attribute Based Encryption with Non-Monotonic Access Structures, ACM CCS 2007, pp.195-203, 2007.
18. S. Hohenberger and B. Waters, Online/Offline Attribute-Based Encryption, PKC 2014, LNCS 8383, Springer, Heidelberg, pp.293-310, 2014.
19. S. Hohenberger and B. Waters, Attribute-Based Encryption with Fast Decryption, PKC 2013, LNCS 7778, Springer, Heidelberg, pp.162-179, 2013.
20. F. C. Guo, Y. Mu, W. Susilo, D. S. Wong and V. Varadharajan, CP-ABE with Constant-Size Keys for Lightweight Devices, IEEE Transactions on Information Forensics and Security, vol.9, no.5, pp.763-771, 2014.
21. N. Attrapadung, J. Herranz, F. Laguillaumie, B. Libert, E. D. Panafieue and C. Ràfols, Attribute-based encryption schemes with constant-Size ciphertexts, Theoretical Computer Science, vol.422, pp.15-38, 2012.
22. E. Zavattoni, L.J.Perez Dominguez, S. Mitsunari, A.H. Sanchez-Ramrez, T. Teruya, F.Rodriguez-Henriquez, Software Implementation of an Attribute-Based Encryption Scheme, IEEE Transactions on Computers, vol.64, no.5, pp.1429-1441, 2015.
23. A. Lewko and B. Waters, Decentralizing Attribute-Based Encryption, EUROCRYPT 2011, LNCS 6632, pp.568-588, Springer, Heidelberg, 2011.
24. M. Chase, Multi-authority attribute based encryption, TCC 2007, pp.515-534, 2007.
25. Jonathan Katz, Amit Sahai, Brent Waters, Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products, Journal of Cryptology, 2013, vol.26, no.2, pp.191-224.
26. Takashi Nishide, Kazuki Yoneyama, and Kazuo Ohta, Attribute-Based Encryption with Partially Hidden Encryptor-Specified Access Structures, ACNS 2008, LNCS 5037, pp.111-129, 2008.
27. Z. Liu, Z.F. Cao, D. S. Wong, Blackbox Traceable CP-ABE: How to Catch People Leaking Their Keys by Selling Decryption Devices on eBay, ACM CCS 2013, pp.475-486, 2013.
28. J. Hong, K.P Xue, W. Li, 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.
29. B. Waters, Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization, PKC 2011, LNCS 6571, Springer, Heidelberg, pp.53-70, 2011.
30. L. Ibraimi, Q. Tang, P. hartel, W. Jonker, Efficient and Provable Secure Ciphertext-Policy Attribute-Based Encryption Schemes, ISPEC 2009, LNCS 5451, pp.1-12, 2009.
31. L. Cheung, C. Newport, provably secure ciphertext policy ABE, ACM CCS 2007, pp.456-465, 2007.
32. B. Lynn, The Stanford pairing based crypto library, http://crypto.stanford.edu/pbc.