Dynamic Policy Update for Ciphertext-Policy Attribute-Based Encryption

Abstract. Ciphertext-policy attribute-based encryption (CP-ABE) is regarded as a promising cryptographic tool for encrypted access control in public cloud storage systems. However, a problem for CP-ABE schemes is that there is no way to change access policy on ciphertext once it is generated. This shortcoming makes us cannot conveniently use CP-ABE as traditional 1-to-1 public key encryption when the access policy needs to be changed. In this paper, we propose a dynamic policy update algorithm for CP-ABE. The policy update algorithm not only has the ability to remove attributes from an access policy but also is able to add newly issued attributes to an access policy. When the access policy of a ciphertext changes, user private key will always be fixed and thus private channels to update user keys are eliminated. Moreover, our policy update algorithm does not rely on predefined attributes, such as timestamp and user ID, and does not produce new public parameters as well. The update algorithm can be independently executed by the creator of a ciphertext and the update times for the ciphertext are unlimited. We construct such a scheme and show its usage in a practical scenario. The performance analysis shows an excellent result: The communication, computation, and storage costs of our policy update are only relevant to the number of attributes in access policy.

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

1 Introduction

Attribute based encryption (ABE) is an expressive encryption method that allows users to encrypt sensitive data based on user attributes. In many scenarios, ABE, particularly CP-ABE, is able to control data access policy by users' roles but not identifiers. For example, to grant access privilege of a document to a group of users with same role, a data owner only needs to encrypt the document with their common attributes for once and uploads the ciphertext to a public cloud server. Each one in the group is able to fetch the ciphertext and obtains the document with own key. Comparing with traditional public key encryption (PKE) which only allows a data owner to encrypt document to a particular user, ABE has more advantages on both performance and reliability. However, ABE is still insufficient for practical applications lack of the functionality of changing access policy.

Consider a scenario that Alice uploads some pictures on her social network pages and no longer keeps these pictures on her local computer again. In default setting of the website, all her friends are able to access these pictures. To implement this function with CP-ABE, Alice is able to encrypt these pictures with the common attributes of her friends. However, if Alice wants to change the access policy such that only some particular friends, e.g. her family members, are able to see these pictures, a trivial way with CP-ABE is to

download the ciphertext, decrypt it, re-encrypt the pictures with the common attributes of the particular friends and upload the new ciphertext again. The public key infrastructure (PKI) technique is a possible choice to change user access privileges, e.g., Alice is able to change the access policy of the pictures once for an individual friend with PKI. However, PKI cannot support privilege changes in 1-to-n encryption. The original ciphertext must be generated with traditional 1-to-1 PKE scheme.

For a social network service (SNS) provider, millions of users' data are stored on cloud servers and access policy of many kinds of data that belongs to different users may be changed for thousands of times a day. No matter the policy update for user data is implemented by previous CP-ABE schemes or PKI, huge computation resources and network bandwidth will be wasted on changing access policy for the SNS provider. Thus, a reasonable choice for a SNS provider is to discard the cryptographic tools for data access control even though the data may be sensitive. On this case, the SNS provider has no choice but to manage user data on its own private cloud storage systems in plaintext form and try its best to protect the private system from outside invasion. If the private storage system is invaded, user data is easy to be leaked. [1]

There have been some attempts to attract application providers back to CP-ABE. One approach is to add expiry time into user attributes in advanced. Those attributes will be invalid after the expiry time. The new ciphertexts generated after the expiry time cannot be decrypted. Actually, when the expiry time is predefined, the privilege changing processes are implicitly defined. If an attribute changes before the expiry time, the system cannot deal with the situation. Another approach is to add a user ID list into the access policy. To add/remove a user to/from a league set, the ciphertext will be re-generated with a new policy that includes/excludes his ID. However, the computation and communication costs of this approach nearly equal re-generating the ciphertext.

A natural idea to solve the policy update problem is to directly modify the ciphertext to fit for new policy. The communication costs for downloading and up-uploading ciphertexts can be avoided. However, previous revocation/update CP-ABE schemes have to update user private key when modifying ciphertext. That is because a private key that can decrypt a ciphertext should match with that ciphertext. We say the matched private key and the ciphertext is in a balance state. If the ciphertext changes, the balance state is broken. In previous revocation/update schemes, all the matched private keys must be changed to achieve new balance state so that decryption successes. It means that a policy update for a ciphertext has to be accompanied by a key update (as in [1]). However, transmitting secrets between authority and users needs private channels. Setting up a private channel between two entities must execute an identity authentication protocol and a session key agreement protocol. In some cases, e.g., the user number is large and data scale is small, the communication and computation costs for transmitting secrets are greater than re-

[1] Many Hollywood stars' private pictures stored on iCloud are leaked in 2014

generating ciphertexts. So, the problem of updating access policy has been a long standing bottleneck in ABE [2] and similar encryption schemes [3, 4].

Our result. An important goal for a network access control tool is to flexibly change user privilege as in a local computer and the additional costs for security do not affect normal applications at the same time. However, previous methods either needs too much communication bandwidth to transmit ciphertexts or needs a large number of private channels to transmit private keys. The aim of this work is to provide a new policy update method for CP-ABE such that the communication and computation costs for the update significantly decreases. In addition, the usage of the new method is compatible with or more convenience than old methods as far as possible. It is a very challenge target. We know most CP-ABE schemes are based on secret sharing technique. The new update method starts from this technique as well. More precisely, our method is based on linear secret sharing (LSS) technique [5] and extends the application of LSS. First, we develop an effective matrix update algorithm to transform an old LSS matrix to a new one. As the result of the algorithm, we can distinguish the attributes, whose corresponding vectors are changed in the new LSS matrix when adding or deleting attributes from old policy. Obviously, only the secret parameters embedded in ciphertext components corresponding to these attributes need updating. Moreover, we find a reasonable way to integrate new ciphertext components into the old ciphertext components. The integration approach ensures that the distribution of the new ciphertext is the same as a ciphertext directly generated from corresponding policy by an encryption algorithm. On this case, the new ciphertext can be simulated with the ciphertext generated by encryption algorithm in security proof. The update for a ciphertext will not restrict further updates of that ciphertext such that the update times are unlimited. In addition, no additional public parameters are produced to increase the cost of original key generation algorithm or encryption algorithm.

The principle of our update is different from that of previous revocation/update scheme. The transformation starts from a ciphertext. We do not need to decrypt the ciphertext and re-encrypt corresponding plaintext again. The update only changes ciphertext components corresponding to particular attributes. The common ciphertext components corresponding to the plaintext are not changed. As a result, ciphertext do not need to be downloaded, re-generated, and re-uploaded. Since the update directly transforms a matched ciphertext to another, the balance state between the ciphertexts and associated private keys are not broken. User private keys do not need to be changed when the access policy and ciphertext are updated. Private channels to transmit secret keys are eliminated. This brings a new advantage for implementation of hardware: Private key may be solidified into smartcard and works throughout its lifecycle.

With the above techniques, we construct a new scheme called ciphertext-policy attribute-based encryption with policy update (PU-CP-ABE). [2] In PU-CP-ABE, the access policy

[2] 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.

of a ciphertext can be independently updated by the creator of the ciphertext without an auxiliary third party. The policy update algorithm not only has the ability to remove existing attributes from an access policy but also is able to add newly issued attributes to an access policy. To compatible with existing systems as much as possible, we do not use new ciphertext structure but choose a more difficult way to implement PU-CP-ABE: We first summarize the common structure of existing LSS-based CP-ABE schemes and then implement the policy update algorithm with the encryption algorithm of any existing LSS-based CP-ABE scheme. Thus, the new policy update method is generic for many LSS-based CP-ABE schemes. The PU-CP-ABE is secure if the given CP-ABE scheme is based on LSS and can be proved secure. The communication, computation and storage costs of our policy update algorithm are only relevant to the number of attributes in the access policy. Both the user number and the data scale will not affect the efficiency of the update.

If the ciphertext creator hopes to delegate the policy update ability to a trusted third party, e.g. cloud server, as in previous revocation/update schemes, for keeping the flexibility of our scheme, the ciphertext creator is able to grant some internal parameters selected in generating the ciphertext to the third party for only one time. [3] Then the access policy of the ciphertext can be updated flexibly either by the ciphertext creator or by the third party. There is no need for the ciphertext creator to provide new secrets for further updates.

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, 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 [2] to tolerant fuzzy biometric characteristics. Goyal, Pandey, Sahai, and Water [6] later formalized two flavors of ABE: KP-ABE and CP-ABE, and implemented first KP-ABE scheme. Then, Bethencourt, Sahai and Waters [7] 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. CP-ABE is very fit for controlling user access privilege in cloud systems. Subsequently, many expanded ABE branches were well studied as in traditional public key encryption: New proof method to achieve full secure [8, 9], new key structures to extend attribute universe [10, 11], new ciphertext generation pattern to decrease encryption or decryption time [12–16], new secure model to enhance security and reliability in practical systems [17, 18], new paradigms to hide access policy [19, 20], and new functions to prevent illegal key sharing among colluding users [21]. However, the problem of flexible access control has not been completely solved.

[3] Internal parameters are secret and need private channel to transmit.

Earlier approach [7, 22] used expiry time to postpone this problem. Pirretti, Traynor, Mcdaniel and Waters [22] 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 [7] presented that the expiry time should be an independent attribute so that different expiry time 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 time must be predefined and the system cannot deal with unexpected changes on user attributes. Thus, the method of adding user ID list into access policy was created as a supplementary approach [23, 24]. 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 re-generating ciphertexts under a new policy.

Yu, Wang, and Ren proposed an attribute revocation scheme with the proxy re-encryption technique [25]. The authority generates proxy re-keys, and then a proxy server updates secret keys except the user with revoked attributes utilizing these proxy re-keys. Then, data will be re-encrypted as well. Similarly, Sahai, Seyalioglu, and Waters [1] further expressed expiry time with LSS matrix [5] and constructed new revocable storage systems to exclude users. But the costs of this method are very huge. For example, to exclude a user from the storage system, authority does not update secret key for that user but updates secret keys for all the other users besides periodically updating ciphertexts. Since transmitting new secret keys for their owners needs many private channels, the costs for the update are far greater than re-generating the ciphertext if the user number is large. Yang et al. [26] proposed an ABE-based access control system based on reference [17]. This scheme extends the function of reference [1]. In [26], data owner generates an update key for a cloud server with no need of updating user private keys. Then the cloud server is able to dynamically change data access policy with the update key. As a result, cloud server shares many computation costs that would have been done by the data owner. However, each time the cloud server updates a ciphertext, a new update key is needed. Transmitting these update keys also needs private channels. Thus, the dilemma exists in reference [1] is still not solved.

As a result, the requirement of too many private channels significantly decreases the efficiency of previous revocation/update schemes and restricts the practical application of CP-ABE. An efficient dynamical policy update algorithm independent of private channels is an urgent demand.

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. 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 MSK.

• **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 policy update algorithm. It includes the state information of the ciphertext. CP-ABE schemes with above inputs can be easily upgraded to PU-CP-ABE without any modification. 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,MSK)$\leftarrow$Setup(1^λ), plaintext M, and SK$\leftarrow$KeyGen(PK,MSK,$\mathcal{S}$), 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 $CT_0^{(b)}$ 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 $CT_i^{(b)}$ to the adversary. Finally, the adversary obtains $\{CT_0^{(b)}, \cdots, CT_n^{(b)}\}$.
• **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 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 $\Pr[b' = b]\text{-}\frac{1}{2}$.

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 $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ 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 [5] 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 [1] 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 [1, 17]:

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 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 $(0, \cdots, 0|\text{-}1)$ 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 equals that of the longest vector. The other shorter vectors are padded with 0's.

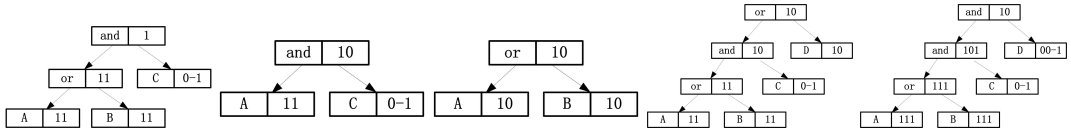
For example, a policy $\mathbb{A}$=((A *or* B) *and* C) can be described in the form of the binary tree as Fig.1(a). The LSS matrix W associated with the attributes A, B, and C can be obtained as follows:

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

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 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.



(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

4.3 LSS Based CP-ABE Schemes

A CP-ABE scheme includes four algorithms: Setup, KeyGen, Encrypt, and Decrypt. The common characteristic of LSS based CP-ABE schemes [7, 9, 10, 12, 13, 17, 21, 23, 27, 28] 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 [29] do not use LSS technique can be regarded as a special case that the access policy only includes attributes and connectors "*and*".

Although the ciphertext structures of 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 [23], m=2 in [9, 21, 27] and scheme1 of [28], m=3 in [7, 10, 12, 17], and m=n_{max} in scheme3 of [28]. 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, MSK, 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

$$B_i = e(C_{1,i}, K_1) \times e(C_{2,i}, K_{2,\rho(i)}) \times \cdots \times e(C_{m,i}, K_{m,\rho(i)})$$

$$= (e(g,g)^{x\lambda_i} e(g,g)^{x_1}) \times (e(g,g)^{-x_1} e(g,g)^{x_2}) \times (e(g,g)^{-x_2}$$
$$e(g,g)^{x_3}) \times \cdots \times (e(g,g)^{-x_{m-1}} e(g,g)^{x_m}) \times (e(g,g)^{-x_m})$$
$$= e(g,g)^{x\lambda_i}$$
$$B = \prod_{i \in I} (B_i)^{\omega_i} = e(g,g)^{x \overrightarrow{v} \sum_{i \in I} \omega_i W_i} = e(g,g)^{xs}$$

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 part, 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 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)$\rightarrow$ (A *or* B).
6. Change a connector "*or*" to "*and*", e.g.(A *or* B)$\rightarrow$ (A *and* B).
7. Change an attribute to another, e.g.(A *or* B)$\rightarrow$ (A *or* C), and (A *and* B)$\rightarrow$ (A *and* C).

Since the policy tree is generated from top to bottom, none of vectors associated with the higher level nodes will be affected when we change an attribute of a policy. Only the vectors associated with its sibling and the children of its sibling 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 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.

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 current policy. The attributes corresponding to these vectors can be recorded.

Continued from above example, Fig.1(b)-(e) show the new binary trees after changing an attribute on policy $\mathbb{A}$ as the case 1) to case 4), and following LSS matrices show the relationship between the 4 new policies and $\mathbb{A}$.

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

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

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

whose corresponding vectors are changed when adding attribute D to policy $\mathbb{A}$. 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 based on a CP-ABE scheme with the matrix update method presented in Section 4.4. We call the given CP-ABE scheme BASE (Setup, KeyGen, Encrypt, Decrypt). To describe conveniently, we suppose that an attribute in the access policy corresponds the ciphertext components that contains m group elements as in Section 4.2. Our construction is as follows.

Setup. Run BASE.Setup and output PK and MSK.

KeyGen. Given PK, MSK, 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→(A *or* B).

Atomic update 4.(A *or* B)→((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 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 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 $\overrightarrow{W}'_i = \overrightarrow{W}_i + \overrightarrow{W}_{A_1} + \cdots + \overrightarrow{W}_{A_q}$.

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

$$= e(g,g)^{xv\overrightarrow{W}'_i}$$
$$= e(g,g)^{x\lambda'_i}$$

Thus, the distribution of the secrets embedded into ciphertext components outputted by the policy update algorithm is the same as that of the secrets 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 [10]. 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:

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

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 $\lambda_1 = \overrightarrow{v}\overrightarrow{W}_1 = s + y_2$, $\lambda_2 = \overrightarrow{v}\overrightarrow{W}_2 = s + y_2$, $\lambda_3 = \overrightarrow{v}\overrightarrow{W}_3 = -y_2$.

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

$$
\begin{aligned}
&C_{1,1} = w^{(\lambda_1 = s+y_2)} v^{t_1}, C_{2,1} = (u^{\rho(1)}h)^{-t_1}, C_{3,1} = g^{t_1},\\
&C_{1,2} = w^{(\lambda_2 = s+y_2)} v^{t_2}, C_{2,2} = (u^{\rho(2)}h)^{-t_2}, C_{3,2} = g^{t_2},\\
&C_{1,3} = w^{(\lambda_3 = -y_2)} v^{t_3}\ , C_{2,3} = (u^{\rho(3)}h)^{-t_3}, C_{3,3} = g^{t_3})\\
IP &= (\lambda_1 = s + y_2, \lambda_2 = s + y_2, \lambda_3 = -y_2)
\end{aligned}
$$

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 $\omega = (1, 0, 1, 0)$

$$
\begin{aligned}
B_1 &= e(C_{1,1}, K_1)e(C_{2,1}, K_{2,\rho(1)})e(C_{3,1}, K_{3,\rho(1)})\\
&= e(w^{\lambda_1}v^{t_1}, g^r)e((u^{\rho(1)}h)^{-t_1}, g^{r_{\rho(1)}})e(g^{t_1}, (u^{\rho(1)}h)^{r_{\rho(1)}}v^{-r})\\
&= e(w, g)^{r\lambda_1}\\
B_3 &= e(C_{1,3}, K_1)e(C_{2,3}, K_{2,\rho(3)})e(C_{3,3}, K_{3,\rho(3)})\\
&= e(w^{\lambda_3}v^{t_3}, g^r)e((u^{\rho(3)}h)^{-t_3}, g^{r_{\rho(3)}})e(g^{t_3}, (u^{\rho(3)}h)^{r_{\rho(3)}}v^{-r})\\
&= e(w, g)^{r\lambda_3}\\
B &= (B_1B_3) = e(w, g)^{r(\lambda_1+\lambda_3)} = e(w, g)^{rs}\\
&\frac{C \cdot B}{e(C_0, K_0)} = \frac{Me(g, g)^{\alpha s}e(w, g)^{rs}}{e(g^s, g^\alpha w^r)} \Rightarrow M
\end{aligned}
$$

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 $N(((A \text{or} B)\text{and} C), C)$=(A,B). For attributes A, B, select random numbers $t_1^{(C)}$ and $t_2^{(C)}$, and compute

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

Finally, we have

$$CT_1 = (\mathbb{A}_1 = (A \ or \ B), C = Me(g,g)^{\alpha s}, C_0 = g^s,$$
$$C_{1,1} = w^{(\lambda_1 = s+y_2)} v^{t_1} w^{(\lambda_3 = -y_2)} v^{t_1^{(C)}}, C_{2,1} = (u^{\rho(1)}h)^{-t_1} (u^{\rho(1)}h)^{-t_1^{(C)}}, C_{3,1} = g^{t_1} g^{t_1^{(C)}},$$
$$C_{1,2} = w^{(\lambda_2 = s+y_2)} v^{t_2} w^{(\lambda_3 = -y_2)} v^{t_2^{(C)}}, C_{2,2} = (u^{\rho(2)}h)^{-t_2} (u^{\rho(2)}h)^{-t_2^{(C)}}, C_{3,2} = g^{t_2} g^{t_2^{(C)}})$$
$$IP_1 = (\lambda_1 = \lambda_1 + \lambda_3 = s, \lambda_2 = \lambda_2 + \lambda_3 = s)$$

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

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

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

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

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

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

$C_{3,1} = g^{t_1} g^{t_1^{(C)}} g^{t_1^{(B)}}$,

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

$IP_3 = (\lambda_1 = s + y_2', \lambda_2 = -y_2')$

4. Atomic update 4 satisfies case 3). Given CT_3 and IP_3, compute $\lambda_{(A\ and\ B)} = \lambda_1 + \lambda_2 = s$, set $\lambda_4 = \lambda_{(A\ and\ B)} = s$. 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

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

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

$C_{3,1} = g^{t_1} g^{t_1^{(C)}} g^{t_1^{(B)}},$

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

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

$IP_4 = (\lambda_1 = s + y_2', \lambda_2 = -y_2', \lambda_4 = s)$

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

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

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

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

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

$IP_4 = (\lambda_1 = s + y_2', \lambda_2 = -y_2', \lambda_4 = s)$

$t_1 + t_1^{(B)} + t_1^{(C)}$ can be regarded as a random number as well. The distribution of CT' outputted by the policy update algorithm is the same as CT' outputted by the encryption algorithm. Now, 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 $\omega = (1, 1, 0, 0)$

$B_1 = e(C_{1,1}, K_1) e(C_{2,1}, K_{2,\rho(1)}) e(C_{3,1}, K_{3,\rho(1)})$

$$
\begin{aligned}
&= e(w^{(\lambda_1=s+y_2)}v^{t_1}w^{(\lambda_3=-y_2)}v^{t_1^{(C)}}w^{y_2'}v^{t_1^{(B)}}, g^r)e((u^{\rho(1)}h)^{-t_1}(u^{\rho(1)}h)^{-t_1^{(C)}}(u^{\rho(1)}h)^{-t_1^{(B)}}, g^{r_{\rho(1)}}) \\
&\times e(g^{t_1}g^{t_1^{(C)}}g^{t_1^{(B)}}, (u^{\rho(1)}h)^{r_{\rho(1)}}v^{-r}) \\
&= e(w,g)^{r(s+y_2')}e(v^{t_1}v^{t_1^{(C)}}v^{t_1^{(B)}}, g^r)e((u^Ah)^{-t_1}(u^Ah)^{-t_1^{(C)}} \\
&\times (u^Ah)^{-t_1^{(B)}}, g^{r_A})e(g^{t_1}g^{t_1^{(C)}}g^{t_1^{(B)}}, (u^Ah)^{r_A}v^{-r}) \\
&= e(w,g)^{r(s+y_2')} \\
B_2 &= e(C_{1,2}, K_1)e(C_{2,2}, K_{2,\rho(2)})e(C_{3,2}, K_{3,\rho(2)}) \\
&= e(w^{(\lambda_2=-y_2')}v^{t_2^{(B)}}, g^r)e((u^{\rho(2)}h)^{-t_2^{(B)}}, g^{r_{\rho(2)}})e(g^{t_2^{(B)}}, (u^Bh)^{r_{\rho(2)}}v^{-r}) \\
&= e(w,g)^{-y_2'}e(v^{t_2^{(B)}}, g^r)e((u^Bh)^{-t_2^{(B)}}, g^{r_B})e(g^{t_2^{(B)}}, (u^Bh)^{r_B}v^{-r}) \\
&= e(w,g)^{-ry_2'} \\
B &= (B_1B_2) = e(w,g)^{r(s+y_2'-y_2')} = e(w,g)^{rs} \\
&\frac{C \cdot B}{e(C_0, K_0)} = \frac{Me(g,g)^{\alpha s}e(w,g)^{rs}}{e(g^s, g^\alpha w^r)} \Rightarrow M
\end{aligned}
$$

As a result, 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.

The only secret input of the policy update algorithm is IP. If the data owner gives IP to a third party with computation ability, e.g. cloud server, the update process can be executed by the third party. Then the input for subsequent updates is the new policy $\mathbb{A}'$. Usually, access policy can be transmitted publicly. Thus, the policy update algorithm can be executed by either the data owner or the cloud server. If the update is executed by the data owner, the ciphertext components should be uploaded to the cloud server. The computation and storage costs in client-side and communication cost are necessary for the update. Otherwise, the computation and storage costs are converged on the server-side and no communication cost for the update. Which side executes the update relies on the demands of practical applications.

Recall the application scenario of Alice's pictures. Let "relationship:A" represent friends of Alice and "relationship:A and relationship:B" represent relatives of Alice. To change the access privilege of the pictures, Alice generates new ciphertext components corresponding to new attribute "relationship:B". The computation and communication costs for the update are smaller than re-generating a ciphertext in previous CP-ABE schemes. If IP is given to the cloud server, all the update operations can be finished by the server. Communications between Alice and the server is eliminated. Furthermore, if Alice frequently uploads documents to the cloud server and needs to change access policy of these documents sometimes, she is able to create a special file to store IPs corresponding to these documents. Suppose an IP includes 20 numbers in average and each number occupies 1024

bits. Only about 2.5 kilobytes are needed to store an IP. The storage space is negligible for Alice.

7 Security Analysis of PU-CP-ABE

In this section, we discuss the security of our PU-CP-ABE.

Security intuition. The four basic algorithms Setup, KeyGen, Encrypt, and Decrypt of PU-CP-ABE are the same as in given CP-ABE scheme. If the basic CP-ABE scheme is secure, we do not need to doubt their security again. The ciphertext components generated by the policy update algorithm has the same distribution as the ciphertext components generated by the encryption algorithm. Thus, We can simply simulate the ciphertext outputted by the policy update algorithm with the encryption algorithm of the given CP-ABE scheme. Intuitively, PU-CP-ABE is secure if the basic CP-ABE is secure. The detailed proof processes are as follows.

Theorem 1. The PU-CP-ABE scheme is secure with respect to Definition 1 if the basic CP-ABE scheme is secure.

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 security of 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 of the basic CP-ABE scheme BASE.

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

Query phase 1. For each secret key query from $\mathcal{A}$, $\mathcal{B}$ passes it to the BASE challenger and return the key constructed by the BASE challenger 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 the BASE challenger. The BASE challenger 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 the BASE challenger. The BASE challenger 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 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 [10] and scheme1 of [28].

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 of the basic CP-ABE scheme. Thus, the PU-CP-ABE scheme is secure with respect to Definition 1 if the basic CP-ABE scheme is secure.

8 Performance Analysis

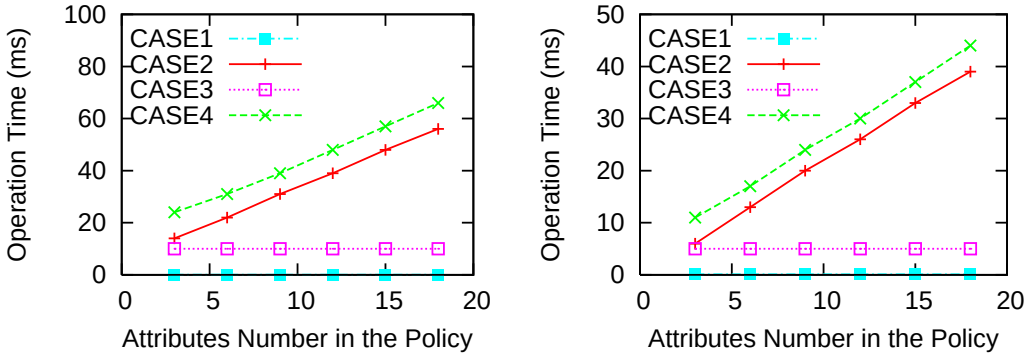
In this section, we compare the efficiency of our new policy update method with previous attribute revocation/update method and the ciphertext re-generation method.

First, in previous revocation/update schemes, user keys should be updated when changing access policy of a ciphertext. To compare with these schemes, we suppose key update can be executed between users and the authority although users are not always online in practical applications. The possible costs for information forward by network routers can be ignored as well. Setting up a private channel needs to execute an identity authentication protocol and a session key agreement protocol. Then, we suppose executing a session key agreement protocol between two parties needs 4 exponential operations and 2 data exchanges as in Diffie-Hellman scheme besides public key verification. Since executing an identity authentication protocol needs similar operations to negotiating a session key, we simply assume executing this protocol needs the same number of computation and communication operations as the key agreement protocol. As a result, at least 8 exponential operations and 4 data exchanges are necessary for setting up a private channel.

Recall the revocation processes of reference [1]. Authority needs to update secret keys for all the users except the revoked user. Suppose the system has n users, to exclude a user from the system, 8(n-1) exponential operations and 4(n-1) data exchanges are needed besides ciphertext update. In addition, n-1 update keys should be transmitted to each user. Comparing with previous revocation/update schemes, the advantage of our method is to eliminate these computation and communication costs for key update.

Next, we compare the executing time of our policy update method with the ciphertext re-generation method. The policy update time includes computation time and communication time. The computation time of the encryption algorithm is decided by the number of attributes in the input access policy, while the computation time of the policy update algorithm is decided by the number of different attributes between the old policy and the new one. If the attributes in new policy is completely different from the old one, the attributes to manipulate in the policy update algorithm may reach about twice of the encryption algorithm. To see the detailed execution time of each kind of operations in the policy update algorithm, we implement the policy update on a large universe CP-ABE scheme [10] and a small universe CP-ABE scheme Scheme1 of [28], respectively, with Charm [30]. The test machine is a Thinkpad X1 laptop with Intel Core i5-3337U 1.8GHz and 64-bit Windows 7 operation system. The algorithms are executed on a virtual machine: VMware Workstation with 2GB RAM and 2 processors. The guest operation system on the virtual machine is 64-bit Ubuntu v13.04. The programming language is Python 3.3. We use symmetric pairing group based on super singular elliptic curve over 512-bit base finite field. The group order of the curve is 160 bits. We use a random element selected from $\mathbb{G}_T$ to simulate the

plaintext M. In initial policy, the number of "*and*" nearly equals "*or*". The experiment results are shown in Fig. 2.



(a) Policy Update on [10] (b) Policy Update on Scheme1 of [28]

Fig. 2. Executing Time for Different Operations of Policy Update

Viewing from Fig.2, the policy update algorithm implemented on [10] needs more operation time than the algorithm implemented on Scheme1 of [28]. That is because the ciphertext components corresponding to an attribute have 3 group elements in [10] but 2 group elements in Scheme1 of [28]. For both of the two schemes, case 1) and case 3) needs little operation time than case 2) and case 4). That is because case 2) and case 4) deal with attributes connected by "*and*". Deleting or adding an attribute connected by "*and*" affects more attributes than "*or*". The execution time of all kinds of operations are within 100 ms although our test environment is not powerful. Since the executing time of the policy update is irrelevant to the common ciphertext components, such as C and C_0, changing the access privilege of a big file or a small file needs same time. In addition, the policy update is irrelevant to user number, changing the access privilege for a user or multiple users with same attributes needs same time. Then, we suppose that updating an attribute needs 50 ms in average, changing the access privilege of a file with 20 different attributes between a new policy and an old one needs about 1 second.

The communication cost of the policy update is decided by the number of attributes in the new access policy. The ciphertext components corresponding to these attributes only occupy dozens of kilobytes. The size of internal parameters is even smaller than these ciphertext components. Thus, the communication time for our policy update can be ignored. Generally speaking, our policy update can be finished within a few seconds.

Image that a 1 GB file is stored on cloud server and controlled by previous CP-ABE schemes. The owner wants to change the access policy of the file. Since hybird encryption may be used for encrypting the data, the decryption and re-encryption time is hard to estimate. However, downloading and re-uploading 1 GB data cannot be finished in a few

seconds. In addition, the data stored on a cloud service provider are far greater than a 1 GB file and may be changed frequently. Obviously, the files are bigger, the number of users is larger, and the policy update is more frequent, our new policy update algorithm is more competitive than previous schemes.

As a result, PU-CP-ABE is a very efficient access control tool for applications on public cloud storage systems.

9 Conclusions

In this paper, we create a policy update method for ciphertext-policy attribute-based encryption. Our method allows the creator of a ciphertext 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 in computation, communication, and storage costs and is very fit for applications in cloud environment.

References

1. A. Sahai, H. Seyalioglu and B. Waters, Dynamic Credentials and Ciphertext Delegation for Attribute-based Encryption, CRYPTO 2012, LNCS 7417, pp.199-217, 2012.
2. A. Sahai and B. Waters, Fuzzy identity based encryption, EUROCRYPT 2005, LNCS 3494, pp.457-473, Springer, Heidelberg, 2005.
3. D. Boneh and M. Franklin, Identity based encryption from the weil pairing, CRYPTO 2001, LNCS 2139, pp.213-229, Springer, Heidelberg, 2001.
4. A. Boldyreva, V. Goyal and V. Kumar, Identity-based Encryption with Efficient Revocation, ACM CCS 2008, pp.417-426, 2008.
5. A. Beimel, Secure Schemes for Secret Sharing and Key Distribution, PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1996.
6. 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.
7. J. Bethencourt, A. Sahai and B. Waters, Ciphertext-Policy Attribute-Based Encryption, IEEE S&P 2007, pp.321-334, 2007.
8. A. Lewko and B. Waters, New techniques for dual system encryption and fully secure hibe with short ciphertexts, TCC 2010, pp.455-479, 2010.
9. A. Lewko and B. Waters, New proof methods for attribute based encryption: Achieving full security through selective techniques, CRYPTO 2012, pp.180-198, 2012.
10. Y. Rouselakis and B. Waters, Practical constructions and new proof methods for large universe attribute-based encryption, ACM CCS 2013, pp.463-474, 2013.
11. R. Ostrovsky, A. Sahai, and B. Waters, Attribute Based Encryption with Non-Monotonic Access Structures, ACM CCS 2007, pp.195-203, 2007.
12. S. Hohenberger and B. Waters, Online/Offline Attribute-Based Encryption, PKC 2014, LNCS 8383, Springer, Heidelberg, pp.293-310, 2014.
13. S. Hohenberger and B. Waters, Attribute-Based Encryption with Fast Decryption, PKC 2013, LNCS 7778, Springer, Heidelberg, pp.162-179, 2013.
14. F. C. Guo, Y. Mu, W. Susilo, D. S. Wong and V. Varadharajan, CP-ABE with Constant-Size Keys for Lightweight Devices, IEEE Trans. Inf. Forensics Security, vol.9, no.5, pp.763-771, 2014.

15. N. Attrapadung, J. Herranz, F. Laguillaumie, B. Libert, E. D. Panafieue and C. Rà fols, Attribute-based encryption schemes with constant-Size ciphertexts, Theoretical Computer Science, vol.422, pp.15-38, 2012.
16. 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.
17. A. Lewko and B. Waters, Decentralizing Attribute-Based Encryption, EUROCRYPT 2011, LNCS 6632, pp.568-588, Springer, Heidelberg, 2011.
18. M. Chase, Multi-authority attribute based encryption, TCC 2007, pp.515-534, 2007.
19. 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.
20. 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.
21. 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.
22. M. Pirretti, P. Traynor, P. Mcdaniel and B. Waters, Secure attribute-based systems, ACM CCS 2006, pp.99-112, 2006.
23. N. Attrapadung and H. Imai, Conjunctive Broadcast and Attribute-Based Encryption, Pairing 2009, LNCS 5671, pp.248-265, Springer, Heidelberg, 2009.
24. A. Lewko, A. Sahai and B. Waters, Revocation Systems with Very Small Private Keys, IEEE S&P 2010, pp.273-285, 2010.
25. S. Yu, C. Wang, K. Ren, Attribute Based Data Sharing with Attribute Revocation, ACM ASIACCS 2010,pp.261-270, 2010.
26. Kan Yang, Xiaohua Jia, Kui Ren, Ruitao Xie and Liusheng Huang, Enabling Efficient Access Control with Dynamic Policy Updating for Big Data in the Cloud, INFOCOM 2014, pp.2013-2021, 2014.
27. 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.
28. B. Waters, Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization, PKC 2011, LNCS 6571, Springer, Heidelberg, pp.53-70, 2011.
29. L. Cheung, C. Newport, provably secure ciphertext policy ABE, ACM CCS 2007, pp.456-465, 2007.
30. A. Joseph, M. Green, and A. Rubin, Charm: A framework for rapidly prototyping cryptosystems, ePrint, Report 2011/617, 2011.