International Journal of Security and Its Applications
Vol.9, No.8 (2015), pp.95-110
http://dx.doi.org/10.14257/ijsia.2015.9.8.08

ABE based Access Control with Authenticated Dynamic Policy Updating in Clouds

Liang-Ao Zhang[1,2], Xingming Sun[1,2], Zhihua Xia[1,2] and Qiuju Ji[2]

[1]*Jiangsu Engineering Center of Network Monitoring, Nanjing University of Information Science & Technology, China*
[2]*School of Computer & Software, Nanjing University of Information Science & Technology, China*
zlo2010@163.com, sunnudt@163.com, xia_zhihua@163.com, ji_qiuju@163.com

Abstract

Attribute-Based Encryption (ABE) is a promising cryptographic primitive to implement access control for secure data storage in the cloud. Since the data owner may frequently change the access policies defined in the ciphertext, it is significant to provide the capacity for dynamic policy updating. However the cloud should also authenticate the owner because the adversary may modify the access policies of the files in the cloud to prevent the legal users from accessing them. In this paper, we focus on the owner's authentication in the ABE systems and propose a novel scheme which enables access control with authenticated dynamic policy updating in the cloud. We adapt the Pedersen commitment and Zero Knowledge Proof of Knowledge (ZKPK) to realize the anonymous authentication of the owner's policy updating key without increasing any secret information to the owner side. The analysis shows that our scheme is authentic and efficient as well as adaptive to different types of access policies.

Keywords: *Access Control, Authentication, Policy Updating, ABE, Cloud Computing*

1. Introduction

Advancing in computer and internet technology, cloud computing is receiving a lot of attention from academic and industrial fields. Benefiting from the capability of realizing reliability and flexibility with low consumption, cloud computing meets the increasing needs on storage and computing resource of the companies which are constrained by the limited local IT resources. However, the security and privacy of the sensitive data in the cloud have raised the users' great concerns as the cloud serves cannot be fully trusted [1]. There are some cryptographic primitives to implement access control which promises the secure data storage in the cloud. One of the promising cryptographic techniques is Attribute-Based Encryption (ABE) [2] which realizes a flexible access control system of encrypted data stored in the cloud, since the ABE enables the data provider to express how he wants to share data in the encryption algorithm.

Since the number of users and the data volume boom quickly and the data owners may dynamically and frequently change the access policies in the cloud, the dynamic policy updating method for ABE has been concerned [3]. However, when the access control scheme provides the ability for the data owners to dynamically update the policies of the files in the cloud, it should also provide authentication mechanism. Without the authentication mechanism, the malicious user may pretend to submit a policy updating request to the cloud server for the data which doesn't belong to him and the cloud will update the corresponding

ISSN: 1738-9976 IJSIA

ciphertext without checking his authentication [4]. This may result that the malicious owner could authorize illicit users to access other data owners' data or prevent the legal users from accessing the data which they are supposed to.

In this paper, we focus on solving the data owner's authentication problem when the owner sends a policy updating request to the cloud. We take advantage of Pedersen commitment and Zero Knowledge Proof of Knowledge (ZKPK) to implement a CP-ABE which enables the cloud to verify whether the access policy updating request for the specific file is issued by the actual data owner. If a malicious owner attempts to update the access policy to other owners' files, the cloud server will refuse to update the corresponding ciphertexts' access policies. Thereby, the security and availability of actual data owners will be preserved.

Our Contributions. We formulate the authentication problem when the access policy needs to be updated. In order to make the cloud qualified to authenticate the data owners, we extend the original scheme of ciphertext-policy ABE with dynamic policy updating in [3]. The core idea of our method is using Pedersen commitment and ZKPK to adapt to the policy updating algorithms. By adding an interact communication between cloud and data owner, the cloud could authenticate the owner without learning anything of the owner's data. Finally, we propose an access control system with authenticated dynamic policy updating for the cloud storage and our ideas could also be applied to other ABE systems.

1.1. Related Work

Since Sahai and Waters [5] first proposed the concept of attribute-based encryption in 2005, many ABE schemes were proposed to extend the ABE's security and functionality. Goyal et al. [6] further clarified the concept of ABE and proposed two complimentary forms of ABE: key-policy ABE (KP-ABE) and ciphertext-policy ABE (CP-ABE) . In CP-ABE scheme, the ciphertext is associated with an access policy on attributes, and the user's private key is associated with a set of attributes. A user is able to decrypt a ciphertext only if the set of attributes associated with the user's private key satisfies the access policy associated with the ciphertext. The KP-ABE scheme has the opposite situation where the access policy is correlated to the user's private key and describes the encrypted data with the user's attribute. There are several other works [7-10] to extend the functionality and security of KP-ABE. As CP-ABE schemes [11-14] are more natural to accomplish access control, we are focused on the CP-ABE to realize our scheme.

In the paper [3, 6, 15] , they discussed the policy updating problem and proposed concrete schemes to enable efficient access control system with dynamic policy updating for the cloud storage. The above schemes enable the cloud to update the policy of the encrypted data and guarantee that the adversary (including the curious cloud server) is not able to learn anything about the encrypted message. However the cloud server never authenticates whether the policy updating key is from the actual data owner in these schemes, the owners' data will be vulnerable to security attacks. The Attribute-Based Signature(ABS) [16] is adapted to ABE based access scheme to realize the anonymous authentication in [17, 18], but they can't directly enable the cloud to authenticate the owner's policy updating key. Furthermore, in the ABE based access control system that uses the ABS scheme to realize the authentication, the authorities have to maintain double attributes. This significantly increases the burden of authorities and the size of the user's key.

The above observation motivates us to develop a new method for the cloud to authenticate the owner's policy updating key the cloud server. We emphasize that the data owners are hypothesized to be fully trusted in Yang et al. scheme [3]. However, in our scheme the data owner may attempt to update other owners' data access policies to obtain their secrets or destroy the availability of their data stored in the cloud for the business profits.

1.2. Organization

We first give necessary preliminary information in section 2 and describe the system model and security model in section 3. Then we present a new access control scheme with authenticated dynamic policy updating in clouds on an adapted CP-ABE method in [3] in section 4. We give a comprehensive analysis of our scheme in security and performance in section 5. Finally, we state our conclusion in section 6.

2. Preliminaries

2.1. Access Structures

Definition 1 (Access Structure [19]): Let $\{P_1,\cdots,P_n\}$ be a set of parties. A collection $\mathbb{A}\subseteq 2^{\{P_1,\cdots,P_n\}}$ is monotone for $\forall B$ and C, if $B\subseteq\mathbb{A}$ and $B\subseteq C$ then $C\subseteq\mathbb{A}$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\{P_1,\cdots,P_n\}$, i.e., $\mathbb{A}\subseteq 2^{\{P_1,\cdots,P_n\}}\setminus\{\varnothing\}$. The sets in $\mathbb{A}$ are called authorized sets, and the sets not in $\mathbb{A}$ are called unauthorized sets.

In attribute-based encryption scheme, attributes play the role of parties and we restrict our attention to monotone access structures. Thus, the access structure $\mathbb{A}$ will contain the authorized sets of attributes. There are different describable types of access policies in ABE scheme, for example, Boolean Formulas, LSSS Structure and Access Tree. The LSSS Structure is introduced as follows. We recommend readers to refer to the Appendix of [12] for the description of Boolean Formulas and a discussion on how to convert any monotonic boolean formula into an LSSS representation. The Access Tree description could find in the Appendix of [3].

Definition 2 (Linear Secret-Sharing Schemes (LSSS)): A secret sharing scheme $\prod$ over a set of parties $\mathcal{P}$ is called linear (over Z_p) if

1) The shares for each party form a vector over Z_p.
2) There exists a matrix M with l rows and n columns called the share-generating matrix for $\prod$. For all $i=1,\cdots,l$, the i-th row of is labeled by a party $\rho(i)$ (ρ is a function from $\{1,\cdots,l\}$ to $\mathcal{P}$). When we consider the column vector $v=(s,r_2,\cdots,r_n)$, where s is the secret to be shared, and $r_2,\cdots,r_n\in Z_p$ are randomly chosen, then Mv is the vector of l shares of the secret according to $\prod$. The share $(Av)_i$ belongs to party $\rho(i)$.

According to the above definition, every linear secret-sharing scheme also enjoys the linear reconstruction property: suppose that $\prod$ is an LSSS for the access structure $\mathbb{A}$. Let $S\in\mathbb{A}$ be any authorized set, and let $I\in\{1,\cdots,l\}$ be defined as $I=\{i\mid\rho(i)\in S\}$. Then there exist constants $\{\omega_i\in Z_p\}_{i\in I}$ such that, if $\{\lambda\}_i$ are valid shares of any secret s according to $\prod$, then $\Sigma_{i\in I}\omega_i\lambda_i=s$. These constants $\{\omega_i\}$ can be found in time polynomial in the size of the share-generation matrix M. Note that, for unauthorized sets, no such constants $\{\omega_i\}$ exist.

2.2. Pedersen Commitment

The Pedersen Commitment scheme is first introduced in [20], It's an unconditionally hiding and computationally binding commitment scheme which is based on the intractability

of the discrete logarithm problem. We give the adapted Pedersen Commitment scheme in a more general language as follows [21].

Setup

A trusted third party T chooses a finite cyclic group G of large prime order p so that the computational Diffie-Hellman problem is hard in G. Write the group operation in G as multiplication. T chooses an element $g \in G$ as a generator, and another element $h \in G$ such that it is hard to find the discrete logarithm of h with respect to g, i.e., an integer α such that $h = g^{\alpha}$. T may or may not know the number α. T publishes G, p, g and h as the system's parameters.

Commit

The domain of committed values is the finite field $\mathbb{F}_p$ of p elements, which can be represented as the set of integers $\mathbb{F}_p = \{0,1,\cdots, p-1\}$. For a party U to commit a value $x \in \mathbb{F}_p$, it randomly chooses $r \in \mathbb{F}_p$, and computes the commitment $c = g^x h^r \in G$.

Open

U shows the values x and r to open a commitment c. The verifier checks whether $c = g^x h^r$.

2.3. Zero-knowledge Proof of Knowledge (ZKPK) Protocol

As the Pedersen Commitment scheme described above, a party U referred to as the prover, can convince the verifier V that U can open a commitment $c = g^x h^r$, without showing the values x and r in clear. Indeed, by the following zero-knowledge proof of knowledge (ZKPK) protocol that proposed in[21], V will learn nothing about the actual values of x and r .This ZKPK protocol, which works for Pedersen commitments, is an adapted version of the zero-knowledge proof protocol proposed by Schnorr[22].

As in the case of Pedersen commitment scheme, a trusted party T generates public parameters G, p, g, h. A prover U who holds private knowledge of values x and r can convince a verifier V that U can open the Pedersen commitment $c = g^x h^r$ as follows.

1) U randomly chooses $y, s \in \mathbb{F}_p^*$, and sends V the element $d = g^y h^s \in G$.
2) V picks a random value $e \in \mathbb{F}_p^*$, and sends e as a challenge to U.
3) U sends $u = y + ex$, $v = s + er$, both in $\mathbb{F}_p$, to V.
4) V accepts the proof if and only if $g^u h^v = d \cdot c^e$ in G.

We adapt this protocol to allow the cloud to authenticate whether the owners hold the data's secret information. In other words, the owners prove to the cloud that they actually own the data.

3. System and Security Model

3.1. System Model

We consider a multiple authorities cloud storage system that enables authenticated dynamic access policy updating. The system model in this paper involves four different entities: the data owners, the data users, the cloud server and the authorities, as illustrated in Figure 1.

Data owners outsource their local data to the cloud server in the encrypted form, and meanwhile keep the capability of updating the access policies, which is associated with the

encrypted data stored in the cloud. The data owners first encrypt the local data under appropriate access policies and outsource the ciphertext to the cloud server. They also issue the access policies updating key to the cloud and answer the cloud's challenge messages.

Data users are assigned with unique global user identities. Each user can freely obtain any ciphertext from the cloud server, but only the users whose attributes satisfy the access policy defined in the ciphertext can decrypt the ciphertext.

Cloud server stores the encrypted data and provides the data access service to the data users. It also processes the policy updating request from the data owners and performs the ciphertext updating if the data owners successfully answer the cloud's challenge messages.

Authority manages the attributes of users in its domain. It generates a secret and public key pair for each attributes in its domain, and generates a secret key for each user according to their attributes. Every authority only needs to share the common global public parameters and doesn't need to communicate with other authorities.

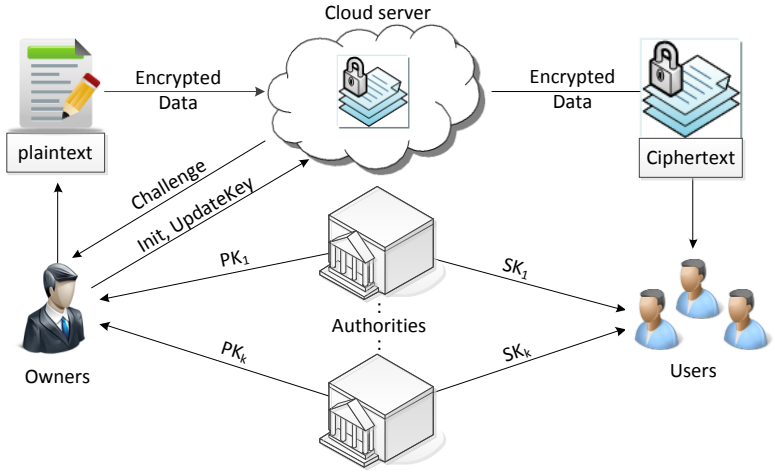


Figure 1. System Model

In order to meet the policy updating authentication, our access control scheme consists of the following nine algorithms:

- GlobalSetup $(\lambda) \rightarrow GP$. The global setup algorithm takes in the implicit security parameter λ and outputs global parameters GP for the system.
- AuthoritySetup $(GP, AID) \rightarrow (SK_{AID}, PK_{AID})$. Each authority AID runs the authority setup algorithm to generate its secret and public key pair (SK_{AID}, PK_{AID}) . It takes as inputs the global parameters GP and the authority identity AID .
- SKeyGen $(GID, GP, S_{GID,AID}, SK_{AID}) \rightarrow SK_{GID,AID}$. The secret key generation algorithm is run by every authority AID to generate the secret key $SK_{GID,AID}$ for user GID . This algorithm takes in the user's global identity GID , the global parameters GP , a set of attributes $S_{GID,AID}$ belonging to this authority AID , and the secret key SK_{AID} of this authority.
- Encryption $(\{PK\}, GP, m, \mathbb{A}) \rightarrow CT$. The data owners run the encryption algorithm to encrypt a message m . This algorithm takes as inputs the set of public keys $\{PK\}$ for the relevant authorities, the global parameters GP , a message m , and an access structure $\mathbb{A}$. It outputs a ciphertext CT .

- Decrypt $(CT, GP, \{SK_{GID,AID}\}) \rightarrow m$. The decryption algorithm takes in the ciphertext CT, the global parameters GP, and a collection of secret keys $\{SK_{GID,AID}\}$ from the relevant authorities AID for user GID. It outputs the message m when the collection of attributes satisfies the access structure corresponding to the ciphertext. Otherwise, decryption fails.
- UAuthKeyInit $(GP) \rightarrow UAK_{token}$. The authenticated update key initialization algorithm is run by the data owner before generating the update key UAK_m. This algorithm takes in the global parameters GP and generates the update key token UAK_{token}. The data owner will send the update key token UAK_{token} to the cloud server.
- UAuthKeyChallenge $(GP, UAK_{token}) \rightarrow UAK_{challenge}$. The update key challenge algorithm is run by the cloud upon receiving the update key token UAK_{token}. This algorithm takes in the global parameters and the update key token. It outputs the challenge message $UAK_{challenge}$, and then the cloud sends the message back to the data owner for authentication.
- UAuthKeyGen $(UAK_{challenge}, \{PK\}, EnInfo(m), \mathbb{A}, \mathbb{A}') \rightarrow UAK_m$. The data owner runs the authenticated update key generation algorithm upon receiving the challenge message $UAK_{challenge}$. It takes in the challenge message, the set of public keys $\{PK\}$ for the relevant authorities, the encryption information $EnInfo(m)$ of the message m, the previous access structure $\mathbb{A}$ and the new access structure $\mathbb{A}'$. It outputs the update key UAK_m of the message m that is used to update the ciphertext CT from the previous access structure to the new one. The data owner can generate a valid update key if only if he could answer the cloud's challenge.
- CTUpdate $(CT, UAK_{token}, UAK_m) \rightarrow CT'$. The cloud server runs the ciphertext updating algorithm. It takes in the previous ciphertext CT, the update key token UAK_{token}, and the update key UAK_m. It outputs a new ciphertext CT' corresponding to the new access structure $\mathbb{A}'$ if only if the owner's update key could answer the cloud's challenge. Otherwise, the cloud server will refuse update the previous ciphertext.

The former five algorithms constitute a traditional decentralized CP-ABE scheme in our new model and the latter four algorithms actually form an access policy updating transaction. This transaction ensures that the malicious users are not able to update the other owners' file and the cloud could learn nothing about the file via the data owners' policy update keys.

3.2. Security Model

We define security mode for our system by the following game between a challenger and an attacker. We assume the cloud server is semi-trusted (curious-but-honest), it is curious about the data and messages it received, but it will accurately perform the users' request. The authorities are assumed to be corrupted statically by the adversaries. But the adversaries can adaptively query the secret keys.

Setup. The global setup algorithm is run. The attacker specifies the statically corrupted authorities set $S_A' \subset S_A$. The challenger runs the authority setup algorithm to obtain public and secret key pairs. For uncorrupted authorities in $S_A - S_A'$, the challenger only gives the public keys to the attacker. For corrupted authorities, the attacker will get both the public and secret keys.

Phase 1. The attacker is allowed to issue queries for private keys by submitting pairs $(GID, S_{GID,AID})$ to the challenger, where GID is the identity and $S_{GID,AID}$ is a set of attributes

belonging to an uncorrupted authority AID. The challenger sends the corresponding secret keys $SK_{GID,AID}$ to the attacker.

Challenge. The attacker specifies two equal length messages, m_0, m_1, and a set of challenge access structure $\{(M_1^*, \rho_1^*), \cdots, (M_q^*, \rho_q^*)\}$ which must satisfy the constraint that the attacker cannot ask for a set of keys that allow decryption, in combination with any keys that can obtained from corrupted authorities. The challenger then flips a random coin b, and encrypts m_b under all access structures $\{(M_1^*, \rho_1^*), \cdots, (M_q^*, \rho_q^*)\}$. At last, the ciphertexts $\{CT_1^*, \cdots, CT_q^*\}$ are given to the attacker.

Phase 2. The attacker may submit more secret keys as Phase 1. The attacker can also issue queries for update keys by submitting the tuple $(UAK_{challenge}^*, EnInfo(m), (M_i^*, \rho_i^*), (M_j^*, \rho_j^*))$ where the $UAK_{challenge}^*$ should not equal to the cloud challenge $UAK_{challenge}$, the simulator returns the update key UAK_{m_b} to the attacker.

Guess. The attacker submits a guess b' of b .If $b' = b$, the attacker wins this game. The attacker's advantage is defined as $\Pr[b' = b] - \frac{1}{2}$.

Definition 3. Our scheme is secure (against static corruption of authorities) if all polynomial time adversaries have at most a negligible advantage in the above security game.

4. Access Control System with Authenticated Dynamic Policy Updating

We construct an attribute-based access control system with authenticated dynamic policy updating for the cloud storage based on an adapted CP-ABE scheme in [3]. Our scheme consists of five phases: System Initialization, Key Generation, Data Encryption, Data Decryption and Policy Updating.

4.1. System Initialization

The system initialization phase generates system global parameters and authority secret and public key pairs. It consists of two algorithms: global setup and authority setup.

Global Setup: In the global setup, two multiplicative groups G and G_T are chosen with the same prime order p and the bilinear map $e: G \times G \to G_T$ between them. A finite cyclic group G_c of prime order p is chosen so that the computational Diffie-Hellman problem is hard in G_c. The group operation in G_c is wrote as multiplication and g_c is a generator of G_c. Another element $h \in G_c$ is chosen such that it is hard to find the discrete logarithm of h with respect to g_c. Let H be a random oracle that maps user global identities GID to elements of G and g be a generator of G. The global parameters is set to be

$$GP = (p, g, H, g_c, h_c)$$

Authority Setup: Each authority AID manages an attribute set S_{AID} in its domain and generates secret and public key pair for every attribute in the set. For each attribute $x \in S_{AID}$, the authority chooses two random exponents $\alpha_x \beta_x \in Z_p$ and publishes its public key as

$$PK_{AID} = \{e(g,g)^{\alpha_x}, g^{\beta_x}\}_{\forall x \in S_{AID}}$$

It keeps $SK_{AID} = \{\alpha_x, \beta_x\}_{\forall x \in S_{AID}}$ as its secret key.

4.2. Key Generation

Each authority *AID* runs secret key generation algorithm to generate the secret key for user *GID*. The authority computes as follows to generate a set of secret keys for the attribute set $S_{GID,AID}$ that is assigned to the user.

$$SK_{GID,AID} = \{K_{x,GID} = g^{\alpha_x} H(GID)^{\beta_x}\}_{\forall x \in S_{GID,AID}}$$

4.3. Data Encryption

The data owner runs the data encryption algorithm to encrypt the message m under the specified access structure. In our scheme, the access structure is described by an $l \times n$ access matrix M with ρ mapping its rows to the attributes. The algorithm takes in a message m, the access matrix (M, ρ), the global parameters, and the public keys $\{PK\}$ for relevant authorities. It chooses a random encryption exponent $s \in Z_p$ and a random vector $\vec{v} = (s, y_2, \cdots, y_l) \in Z_p^l$, where $y_2, \cdots, y_l$ are used to share the encryption exponent s. For each row i of M, it computes $\lambda_i = M_i \cdot \vec{v}$. It also chooses a random vector $\vec{\omega} \in Z_p^l$ with 0 as its first entry and computes $\omega_i = M_i \cdot \vec{\omega}$. For each row i of M, it randomly chooses $r_i \in Z_p$ and computes the ciphertext as

$$CT = (C = m \cdot e(g,g)^s, \forall i = 1 \, to \, n : C_{1,i} = e(g,g)^{\lambda_i} e(g,g)^{\alpha_{\rho(i)} r_i},$$
$$C_{2,i} = g^{r_i}, C_{3,i} = g^{\beta_{\rho(i)} r_i} g^{\omega_i}, C_{4,i} = g_c^{\lambda_i} h_c^{\omega_i})$$

Then, the data owner keeps the encryption information *EnInfo*(*m*) of the data *m* which contains all the random numbers and vectors, i.e., $EnInfo(m) = \{\vec{v}, \vec{\omega}, r_1, \cdots, r_n\}$.

4.4. Data Decryption

To decrypt the ciphertext which is encrypted under the access matrix (M, ρ), the data user first obtains $H(GID)$ from the random oracle. If the user has the secret keys $\{K_{\rho(i),GID}\}$ for a subset of rows i of M such that $(1,0,\cdots,0)$ is in the span of these rows, then the user proceeds as follows. For each such i, the user computes:

$$\frac{C_{1,i} \cdot e(H(GID), C_{3,i})}{e(K_{\rho(i),GID}, C_{2,i})} = e(g,g)^{\lambda_i} e(H(GID), g)^{\omega_i}$$

The data user then chooses constants $c_i \in \mathbb{Z}_p$ such that $\sum_i c_i M_i = (1,0,\cdots,0)$. We recall that $\lambda_i = M_i \cdot \vec{v}$ and $\omega_i = M_i \cdot \vec{\omega}$, where $\vec{v} \cdot (1,0,\cdots,0) = s$ and $\vec{\omega} \cdot (1,0,\cdots,0) = 0$. The user computes:

$$\prod_i (e(g,g)^{\lambda_i} e(H(GID), g)^{\omega_i})^{c_i} = e(g,g)^s$$

At last, the message can be obtained as: $m = C_0 / e(g,g)^s$

4.5. Policy Updating

We allow the data owner to delegate the ciphertext update computations to the cloud server by sending the update key query. If the data owner could answer the cloud server's challenges, the cloud will update the access policy of the encrypted data stored in the cloud. This will significantly reduce the computation and communication of the data owner.

When the data owner wants to update the ciphertext from the previous access policy $\mathbb{A}$ to the new access policy $\mathbb{A}'$, he performs access policy updating transaction. Firstly, the data owner runs the UAuthKeyInit algorithm to generate the update key token UAK_{token} and then owner sends the token UAK_{token} to the cloud server. Upon receiving the token from the owner, the cloud server will run UAuthKeyChallenge algorithm to generate a challenge message $UAK_{challenge}$ and sends it back to the owner. The data owner generates an update key UAK_m corresponding to the challenge message by running the UAuthKeyGen algorithm and sends the update key UAK_m to the cloud server. The cloud server will run the ciphertext updating algorithm CTUpdate to update the ciphertext from the previous access policy $\mathbb{A}$ to the new one $\mathbb{A}'$ if the owner's update key could answer the cloud's challenge. Otherwise, the cloud server will refuse updating the previous ciphertext.

UAuthKeyInit: The data owner first runs this algorithm before he generates the update key UAK_m. The owner randomly chooses $\varepsilon, \sigma \in \mathbb{F}_p^*$ via the global parameters GP and computes the update key token UAK_{token} as

$$UAK_{token} = g_c^{\varepsilon} h_c^{\sigma}$$

Then, the data owner sends the token UAK_{token} to the cloud server for the alleged file that he will commit a policy update key and keep the ε, σ as secret.

UAuthKeyChallenge: Upon receiving the token UAK_{token} from the owner, the cloud server runs this algorithm to generate a challenge message $UAK_{challenge}$. The cloud first stores the UAK_{token} from the data owner and then picks a random value $e \in \mathbb{F}_p^*$. The cloud server sets the $UAK_{challenge} = e$ and then sends the challenge message $UAK_{challenge}$ back to data owner.

In the ABE scheme, the access policy can be expressed in monotonic boolean formulas, LSSS structures, or threshold gate access trees. It's easy to convert the monotonic boolean formulas into the LSSS structures. The threshold access trees can be also transformed to the LSSS structures, so the problem of updating a threshold gate can be transformed to update a LSSS structure [3]. In our access control scheme, we adopt the LSSS structure to express the access policy. We will also present the policy updating algorithms for monotonic boolean formulas, since, these algorithms allow the cloud server partially to update the ciphertexts corresponding to the update key.

4.5.1. Updating a Boolean Formula: all boolean formulas updating operation can be represented by four basic operations: **Attr2OR**, **Attr2AND**, **AttrRmOR** and **AttrRmAND** as shown in Figure 2 in [3].

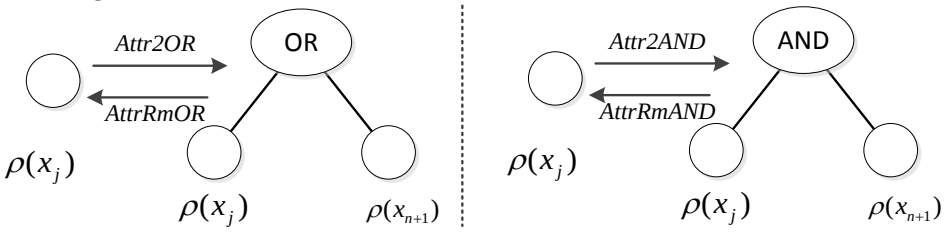


Figure 2. Operations of Boolean Formula

1) **Converting an attribute to an OR gate (Attr2OR):** The *Attr2OR* operation will convert an existing attribute $x_j (j \in [1, n])$ to an OR gate $(x_j \vee x_{n+1})$ by adding a new

attribute x_{n+1}. The new attribute x_{n+1} plays the same role as the previous attribute x_j in the new access policy. Therefore, the ciphertext component $C_{n+1} = (C_{1,n+1}, C_{2,n+1}, C_{3,n+1}, C_{4,n+1})$ for the new attribute can be constructed referring to the component C_j corresponding to the existing attribute x_j. The UAuthKeyGen algorithm takes in the challenge message $UAK_{challenge}$ of the data m from the cloud, the encryption information $EnInfo(m)$ and the set of public keys $\{PK\}$ for the relevant authorities. It chooses random $a_m, r_{n+1} \in Z_p$ and generates the update key as

$$UAK_m = (a_m, UK_{1,m} = \frac{e(g,g)^{a_{x_{n+1}} r_{n+1}}}{e(g,g)^{a_{x_j} r_j a_m}}, UK_{2,m} = g^{r_{n+1} - r_j},$$

$$UK_{3,m} = \frac{e(g,g)^{\beta_{x_{n+1}} r_{n+1}}}{e(g,g)^{\beta_{x_j} r_j a_m}}, \varepsilon + e\lambda_j, \sigma + e\omega_j)$$

Then, the data owner sends the tuple $(Attr2OR, UK_m)$ to the server and asks the server to update the ciphertext CT corresponding to m. The server runs the CTUpdate algorithm to check the data owner's authentication and determine whether updates the corresponding ciphertext component. If the following equation is established,

$$g_c^{\varepsilon + e\lambda_j} h_c^{\sigma + e\omega_j} = C_{4,j} \cdot UAK_{token}^{UAK_{challenge}}$$

the server will construct the new ciphertext component $C_{x_{n+1}}$ as follows

$$C_{1,n+1} = (C_{1,j})^{a_m} \cdot UK_{1,m} = e(g,g)^{\lambda_{n+1}} e(g,g)^{\alpha_{x_{n+1}} r_{n+1}}, C_{2,n+1} = C_{2,j} \cdot UK_{2,m} = g^{r_{n+1}},$$

$$C_{3,n+1} = (C_{3,j})^{a_m} \cdot UK_{3,m} = g^{\beta_{x_{n+1}} r_{n+1}} g^{\omega_{n+1}}, C_{4,n+1} = (C_{4,j})^{a_m} = g_c^{\lambda_{n+1}} h_c^{\omega_{n+1}})$$

where $\lambda_{n+1} = a_m \cdot \lambda_j$ and $\omega_{n+1} = a_m \cdot \omega_j$. Otherwise, the cloud server will refuse to update the corresponding ciphertext component.

1) **Converting an attribute to an AND gate (Attr2AND):** The *Attr2AND* operation will convert an existing attribute x_j ($j \in [1, n]$) to an AND gate $(x_j \wedge x_{n+1})$ by adding a new attribute x_{n+1}. The new attribute x_{n+1} and x_j in the new policy play the same role as the attribute x_j in the previous policy. The UAuthKeyGen algorithm takes in the challenge message $UAK_{challenge}$ of the data m from the cloud, the encryption information $EnInfo(m)$ and the set of public keys $\{PK\}$ for the relevant authorities. It chooses random $a_m, \lambda', \omega', r_{n+1} \in$ □ $_p$ and sets $\lambda'_j = \lambda_j + \lambda'$, $\lambda_{n+1} = a_m \cdot \lambda'$, as well as $\omega'_j = \omega_j + \omega'$ and $\omega_{n+1} = a_m \cdot \omega_j$. Then, the owner generates the update key as

$$UAK_m = (UK_{1,m} = e(g,g)^{\lambda'}, UK_{2,m} = g^{\omega'}, UK_{3,m} = g_c^{\lambda'_j} h_c^{\omega'_j}, \varepsilon + e\lambda_j, \sigma + e\omega_j, C_{n+1})$$

where the new ciphertext component C_{n+1} corresponding to the new attribute x_{n+1} is constructed as

$$C_{n+1} = (C_{1,n+1} = e(g,g)^{\lambda_{n+1}} e(g,g)^{\alpha_{x_{n+1}} r_{n+1}}, C_{2,n+1} = g^{r_{n+1}}, C_{3,n+1} = g^{\beta_{x_{n+1}} r_{n+1}} g^{\omega_{n+1}}, C_{4,n+1} = g_c^{\lambda_{n+1}} h_c^{\omega_{n+1}})$$

Then, the data owner sends the tuple $(Attr2AND, UK_m)$ to the server and asks the server to update the ciphertext CT corresponding to m. The server runs the CTUpdate algorithm to process the owner's request. If the following equation is established,

$$g_c^{\varepsilon+e\lambda_j} h_c^{\sigma+e\omega_j} = C_{4,j} \cdot UAK_{token}^{UAK_{challenge}}$$

the server will updates the corresponding ciphertext component by the following steps. The cloud server first adds the new ciphertext component C_{n+1} to the ciphertext, and then the previous ciphertext component C_j updates to the new version C_j' as

$$C_j' = (C_{1,j}' = C_{1,j} \cdot UK_{1,m} = e(g,g)^{\lambda_j} e(g,g)^{\alpha_{x_j} r_j}, C_{2,j}' = C_{2,j}, C_{3,j}' = C_{3,j} \cdot UK_{2,m} = g^{\beta_{x_j} r_j} g^{\omega_j}, UK_{3,m})$$

Otherwise, the cloud server will refuse to update the corresponding ciphertext component.

2) **Removing an attribute from an OR gate (AttrRmOR):** The *Attr2AND* operation will ask the cloud to remove a ciphertext component C_j corresponding to an existing attribute x_j ($j \in [1,n]$). The data owner runs UAuthKeyGen to generate the update key $UAK_m = (\varepsilon + e\lambda_j, \sigma + e\omega_j)$. The owner will send the tuple $(AttrRM, m, j)$ where $\rho(j) = x_j$ to the cloud server.

The cloud server runs the CTUpdate algorithm. If $g_c^{\varepsilon+e\lambda_j} h_c^{\sigma+e\omega_j} = C_{4,j} \cdot UAK_{token}^{UAK_{challenge}}$ is established, the cloud server will delete the corresponding ciphertext component C_j in the ciphertext. Otherwise, the cloud server will refuse to delete the ciphertext component.

3) **Removing an attribute from an AND gate (AttrRmAND):** To remove an attribute from an AND gate, all the shares should be re-randomized. This can be achieved by using the method of converting the monotonic boolean formulas to the LSSS structures. The problem of removing an attribute from an AND will be transformed to update a LSSS structure. We will illustrate how to update a LSSS structure later.

4.5.2. Updating a LSSS Structure: we have used the LSSS structure to construct our access control scheme. To save the cost of communication and computation, we won't change the encryption secret s to convert a LSSS structure (M, ρ) to a new one (M', ρ'), so we could take advantage of the previous ciphertext. The data owner should also know the public key $(g^{\alpha_x}, g^{\beta_x})$ of each attribute x to re-randomize the encryption secret s. When the data owner get challenge message $UAK_{challenge}$ from the cloud, he will run UAuthKeyGen algorithm to generate an update key UAK_m corresponding to the challenge message and send back UAK_m to the cloud. The cloud server will run the CTUpdate to update the ciphertext. These two algorithms are designed as follows.

Update Key Generation: The UAuthKeyGen algorithm takes in the public keys, the challenge message $UAK_{challenge}$ of the data m from the cloud, the encryption information $EnInfo(m)$, the previous access policy (M, ρ) and the new one (M', ρ'). We suppose that the new access policy is an $l' \times n'$ access matrix M' with ρ' mapping its rows to the attributes. Since the mapping functions ρ and ρ' are non-injective, we can divide the M' row indexes into three sets $I_{1,M'}$, $I_{2,M'}$, $I_{3,M'}$ by comparing to the previous access policy (M, ρ) according to [3].

This algorithm first constructs two random vectors: $\vec{v}' \in Z_p^l$ with s as its first entry and $\vec{\omega}' \in Z_p^l$ with 0 as its first entry. The data owner can easily

compute $\lambda_i = M_i \cdot \vec{v}$ and $\omega_i = M_i \cdot \vec{\omega}$ through the *EnInfo*(*m*) of *m* and the previous access policy (M, ρ). It also computes $\lambda'_j = M'_j \cdot \vec{v}$ and $\omega'_j = M'_j \cdot \vec{\omega}$ where M'_j is the vector corresponding to the *j* -th row of M'. Let $I_M = \{1, \cdots, l\}$ be the index set of the rows of M.

For each $j \in [1, l']$, if $(j,i) \in I_{1,M'}$ (Type1), the algorithm sets $r'_j = r_i$ and generates the update key component as

$$UAK_{j,i,m} = (UK^{(1)}_{j,i,m} = g^{\lambda'_j - \lambda_i}, UK^{(2)}_{j,i,m} = g^{\omega'_j - \omega_i}, UK^{(3)}_{j,i,m} = g_c^{\lambda'_j} h_c^{\omega'_j}, \varepsilon + e\lambda_i, \sigma + e\omega_i)$$

For each $i \in [1, l]$, if the i -th row of M is not exit in Type1, the algorithm generates the authentication component as

$$UAK_{*,i,m} = (\varepsilon + e\lambda_i, \sigma + e\omega_i)$$

If $(j,i) \in I_{2,M'}$ (Type2), the algorithm chooses random numbers $r'_j, a_j \in \mathbb{Z}_p$ and generates the update key component as

$$UAK_{j,i,m} = (a_j, UK^{(1)}_{j,i,m} = g^{\lambda'_j - a_j \lambda_i}, UK^{(2)}_{j,i,m} = g^{\omega'_j - a_j \omega_i}, UK^{(3)}_{j,i,m} = g_c^{\lambda'_j} h_c^{\omega'_j})$$

If $(j,i) \in I_{3,M'}$ (Type3), the algorithm chooses a random numbers $r'_j \in \mathbb{Z}_p$ and generates the update key component as

$$UAK_{j,i,m} = (UK^{(1)}_{j,i,m} = g^{\lambda'_j} \cdot e(g,g)^{\alpha_{\rho'(j)} r'_j}, UK^{(2)}_{j,i,m} = g^{r'_j}, UK^{(3)}_{j,i,m} = g^{\beta_{\rho'(j)} r'_j} g^{\omega'_j}, UK^{(4)}_{j,i,m} = g_c^{\lambda'_j} h_c^{\omega'_j})$$

Finally, the update key UAK_m is constructed as

$$UAK_m = ((Type1, \{UAK_{j,i,m}\}_{(j,i) \in I_{1,M'}}), (Type2, \{UAK_{j,i,m}\}_{(j,i) \in I_{2,M'}}),$$
$$(Type3, \{UAK_{j,i,m}\}_{(j,i) \in I_{3,M'}}), (Auth, \{UAK_{*,i,m}\}_{(*,i) \notin I_{1,M'}}))$$

Ciphertext Update: Upon receiving the update key UAK_m, the server runs the ciphertext updating algorithm which is described as follows.

Firstly, the cloud server will check the data owner's authentication. For each row $i \in [1, l]$ in M, the cloud checks whether the following equation is established

$$g_c^{\varepsilon + e\lambda_i} h_c^{\sigma + e\omega_i} = C_{4,i} \cdot UAK_{token}^{UAK_{challenge}}$$

If all the equations for each $i \in [1, l]$ are established, the cloud server will update ciphertext in following steps. Otherwise, the cloud will refuse to update the ciphertext and terminate the operations. For each $j \in [1, l']$ in M', the cloud will compute each ciphertext component C'_j as follows.

If Type1 ($(j,i) \in I_{1,M'}$), the algorithm will generate the ciphertext component C'_j as

$$C'_j = (C'_{1,j} = C_{1,i} \cdot e(g, UK^{(1)}_{j,i,m}) = e(g,g)^{\lambda'_j} e(g,g)^{\alpha_{\rho'(j)} r'_j}, C'_{2,j} = C_{2,i},$$
$$C'_{3,j} = C_{3,i} \cdot UK^{(2)}_{j,i,m} = g^{\beta_{\rho'(j)} r'_j} g^{\omega'_j}, UK^{(3)}_{j,i,m} = g_c^{\lambda'_j} h_c^{\omega'_j}) \quad \text{where } r'_j = r_i .$$

If Type2 ($(j,i) \in I_{2,M'}$), the algorithm will generate the ciphertext component C'_j as

$$C'_j = (C'_{1,j} = (C_{1,i})^{a_j} \cdot e(g, UK^{(1)}_{j,i,m}) = e(g,g)^{\lambda'_j} e(g,g)^{\alpha_{\rho'(j)} r'_j}, C'_{2,j} = (C_{2,i})^{a_j} = g^{r'},$$
$$C'_{3,j} = (C_{3,i})^{a_j} \cdot UK^{(2)}_{j,i,m} = g^{\beta_{\rho'(j)} r'_j} g^{\omega'_j}, UK^{(3)}_{j,i,m} = g_c^{\lambda'_j} h_c^{\omega'_j}) \quad \text{where } r'_j = a_j r_i .$$

If Type3 ($(j,i) \in I_{3,M'}$), the algorithm will generate the ciphertext component C'_j as

$$C'_j = (C'_{1,j} = e(g, UK^{(1)}_{j,i,m}) = e(g,g)^{\lambda'_j} e(g,g)^{\alpha_{\rho'(j)} r'_j}, C'_{2,j} = UK^{(2)}_{j,i,m},$$
$$C'_{3,j} = UK^{(3)}_{j,i,m} = g^{\beta_{\rho'(j)} r'_j} g^{\omega'_j}, UK^{(4)}_{j,i,m} = g_c^{\lambda'_j} h_c^{\omega'_j})$$

The cloud server constructs the new ciphertext CT' as follow

$$CT' = (m \cdot e(g,g)^s, C'_j \ \forall j \in [1,l'])$$

As we can see, the data owner only does the minimum computations since most of the pairing computations are moved to the cloud server in our scheme.

5. Analysis of Our Scheme

In this section, we give the analysis of our proposed scheme to show its security and authentication. Then, we will also give the analysis of performance in our scheme.

5.1. Security

We adapt the CP-ABE and policy update scheme(CP-ABE-Update) in [3] and also construct our access control scheme primarily on the prime order groups, since the prime order group operations are much faster than the ones on the composite order groups. The authentication component of our scheme is constructed on finite cyclic group. We will give the security analysis under the security model defined in Section 3. However, our scheme can also be extended to be provable secure in the random oracle model by using groups with composite orders.

Theorem 1. Our scheme is secure in the generic bilinear group model and random oracle model, if no polynomial time adversary can get non-negligible advantage in the security game.

Proof: Our access control scheme is constructed on the basis of the CP-ABE and policy update scheme with primer group order in [3], which has proved to be secure under generic bilinear group model and random oracle model. Since the authentication component in our scheme which will reveal nothing to the adversary is constructed on finite cyclic group, if there are any vulnerabilities in our scheme, then these vulnerabilities must exploit specific mathematical properties of elliptic curve groups or cryptographic hash functions used when instantiating the scheme. Suppose that we have an adversary A with non-negligible advantage $\varepsilon = adv_A$ to win the security game, and we will construct an A' to win the security game with non-negligible advantage in [3].

A' initializes the CP-ABE-Update security game and forwards the public key PK to A. A initializes a security game to get the public key PK_{fg} of the finite cyclic group and set $PK' = (PK, PK_{fg})$. Since the key generation step is the same between our scheme and CP-ABE-Update, A' queries its key generation oracle to simulate the key generation oracle of A for all $x \in S$ to respond to a $SK(GID, S)$ query. The simulation of challenge ciphertext of A is the same with the one of A' except that the simulator generates the ciphertext component $C_{4,i} = g_c^{\lambda_i} h_c^{\omega_i}$ through the PK_{fg} and the shares λ_i, ω_i for the corresponding attribute.

Next, we prove that the update key query in our security game will not increase the advantage of A'. Considering two update key queries $UAK(m_0, UAK^*_{challenge}, (M^*_i, \rho^*_i), (M^*_j, \rho^*_j))$ and $UAK(m_1, UAK^*_{challenge}, (M^*_i, \rho^*_i), (M^*_j, \rho^*_j))$, the update key generation oracle returns the same update keys which do not involve with the challenge data, if we consider the encryption random numbers are the same in encrypting m_0 and m_1 (this is because only one challenge message is chosen by the simulator by tossing a coin in the security game). Therefore, the update key will not reveal any information on the chosen challenging message. This completes the proof.

5.2. Authentication

Theorem 2: Our scheme guarantees the authentication of the data owner when the cloud performs the access policy updating request.

Proof: The component $C_{4,i} = g_c^{\lambda_i} h_c^{\omega_i}$ in the ciphertext is the Pedersen commitment of the encryption information *EnInfo*(*m*) of *m* . The data owner performs the UAuthKeyInit, UAuthKeyChallenge and UAuthKeyGen algorithms to prove to the cloud that he can open the Pedersen commitment without showing the *EnInfo*(*m*) information in clear. Since the adversary actually doesn't have the knowledge of *EnInfo*(*m*) , if he can break the authentication mechanism only with the knowledge from the update key generation oracle, the adversary could break the ZKPK protocol and the Pedersen commitment. Therefore, no one could pass the authentication of the cloud except the data owner who knows the encryption information. Our scheme can guarantee the authentication of the data owners when they ask the cloud to update access policy.

5.3. Performance Analysis

Compared with the scheme proposed in [3], the data owner only needs to increase a round of communication with the cloud server to generate the update keys in our scheme. The owner don't need the whole encrypted big data, therefore, our scheme can also keep the advantage that significantly reduces the communication cost during the policy updating. Suppose $|p|$ is the element size in the G , G_T , G_c and $\square_p$. We compare the size of update keys in our scheme with Yang's scheme, showing in Table I. Our scheme incurs acceptable communication cost to support the authentication feature compare to Yang's scheme.

Table 1. Comparison of the Update Key Size

Operation	Attr2OR	Attr2AND	AttrRmOR	Type1	Type2	Type3
Yang's (UK) Size	$4\|p\|$	$5\|p\|$	0	$2\|p\|$	$3\|p\|$	$3\|p\|$
Our (UAK) Size	$6\|p\|$	$9\|p\|$	$2\|p\|$	$5\|p\|$	$4\|p\|$	$4\|p\|$

Our scheme has made full use of the previous ciphertexts encrypted under the old access structure. Comparing to Yang's scheme, we only increase two exponential computations and several finite cyclic group operation for each attribute to realize the authentication in a policy updating transaction. We delegate all the pairing operations to the server in the LSSS structure updating scheme, so the data owner can significantly reduce the computational burden on the policy updating. Therefore, our scheme can minimize the workload of data owners on the policy updating operation and also prevent the unauthentic users from updating access policy of the ciphertext in the cloud.

6. Conclusion

In this paper, we propose an access control system with authenticated dynamic policy updating for the cloud. Taking advantage of the Pedersen commitment and Zero Knowledge Proof of Knowledge (ZKPK), we extend the CP-ABE with dynamic policy updating to support the authentication of data owner. Our proposed scheme guarantees that only the actual data owner could pass the cloud server's authentication and legally update the ciphertext corresponding to the owner's data. Furthermore, we have designed policy updating algorithms with authentication for access policy expressed by boolean formula and LSSS

structure. We also have given the analysis of our scheme on the security, authentication and performance. Since the cloud will learn nothing of the data owner except that the owner could open the commitment, our scheme supports anonymous authentication. Our ideas and methods of realizing the authentication could also be applied to other ABE systems. In the future, we will continue to explore reliable CP-ABE and authentication methods to improve our scheme.

Acknowledgements

This work is supported by the NSFC (61232016, U1405254, 61173141, 61173142, 61173136, 61373133), 201301030, 2013DFG12860, BC2013012 and PAPD fund.

References

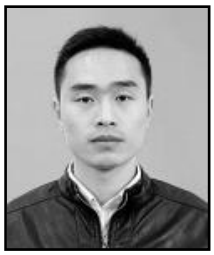
[1] L. Wei, H. Zhu, Z. Cao, X. Dong, W. Jia and Y. Chen, *et al.*, "Security and privacy for storage and computation in cloud computing," *Information Sciences*, vol. 258, (**2014**), pp. 371-386.
[2] K. Yang, X. Jia, K. Ren, and B. Zhang, "Dac-macs: Effective data access control for multi-authority cloud storage systems," in *INFOCOM, 2013 Proceedings IEEE*, (**2013**), pp. 2895-2903.
[3] K. Yang, X. Jia, K. Ren, R. Xie, and L. Huang, "Enabling efficient access control with dynamic policy updating for big data in the cloud," in *INFOCOM, 2014 Proceedings IEEE*, (**2014**), pp. 2013-2021.
[4] D. Ayed, P. Bichsel, J. Camenisch, and J. den Hartog, "Integration of Data-Minimising Authentication into Authorisation Systems," in *Trust and Trustworthy Computing*, ed: Springer, (**2014**), pp. 179-187.
[5] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Advances in Cryptology–EUROCRYPT 2005*, ed: Springer, (**2005**), pp. 457-473.
[6] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, (**2006**), pp. 89-98.
[7] M. Chase and S. S. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *Proceedings of the 16th ACM conference on Computer and communications security*, (**2009**), pp. 121-130.
[8] N. Attrapadung, B. Libert, and E. De Panafieu, "Expressive key-policy attribute-based encryption with constant-size ciphertexts," in *Public Key Cryptography–PKC 2011*, ed: Springer, (**2011**), pp. 90-108.
[9] S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters, "Attribute-based encryption for circuits from multilinear maps," in *Advances in Cryptology–CRYPTO 2013*, ed: Springer, (**2013**), pp. 479-499.
[10] Y. Shi, Q. Zheng, J. Liu, and Z. Han, "Directly revocable key-policy attribute-based encryption with verifiable ciphertext delegation," *Information Sciences*, vol. 295, (**2015**), pp. 221-231.
[11] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Security and Privacy, 2007. SP'07. IEEE Symposium on*, (**2007**), pp. 321-334.
[12] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Advances in Cryptology–EUROCRYPT 2011*, ed: Springer, (**2011**), pp. 568-588.
[13] J. Lai, R. H. Deng, C. Guan, and J. Weng, "Attribute-based encryption with verifiable outsourced decryption," *Information Forensics and Security, IEEE Transactions on*, vol. 8, (**2013**), pp. 1343-1354.
[14] S. Hohenberger and B. Waters, "Online/offline attribute-based encryption," in *Public-Key Cryptography–PKC 2014*, ed: Springer, (**2014**), pp. 293-310.
[15] A. Sahai, H. Seyalioglu, and B. Waters, "Dynamic credentials and ciphertext delegation for attribute-based encryption," in *Advances in Cryptology–CRYPTO 2012*, ed: Springer, (**2012**), pp. 199-217.
[16] H. Maji, M. Prabhakaran, and M. Rosulek, "Attribute-Based Signatures," in *Topics in Cryptology – CT-RSA 2011*. vol. 6558, A. Kiayias, Ed., ed: Springer Berlin Heidelberg, (**2011**), pp. 376-392.
[17] S. Ruj, M. Stojmenovic, and A. Nayak, "Decentralized Access Control with Anonymous Authentication of Data Stored in Clouds," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, (**2014**), pp. 384-394.
[18] Z. Liu, H. Yan, and Z. Li, "Server-aided anonymous attribute-based authentication in cloud computing," *Future Generation Computer Systems, (***2015**).
[19] A. Beimel, "Secure schemes for secret sharing and key distribution," Technion-Israel Institute of technology, Faculty of computer science, (**1996**).
[20] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology—CRYPTO'91*, (**1992**), pp. 129-140.

[21] F. Paci, N. Shang, S. Kerr, K. Steuer Jr, J. Woo, and E. Bertino, "Privacy-preserving management of transactions' receipts for mobile environments," in *Proceedings of the 8th Symposium on Identity and Trust on the Internet*, (**2009**), pp. 73-84.
[22] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology—Crypto'89 Proceedings*, (**1990**), pp. 239-252.

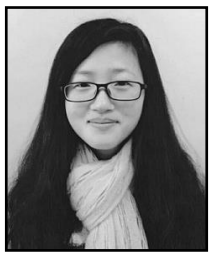
Authors

Liang-Ao Zhang, He received his BS in computer science and technology from Nanjing University of Information Science & Technology in 2013, China. He is currently pursuing his MS in computer science and technology at the College of Computer and Software, in Nanjing University of Information Science & Technology, China. His research interest is cloud computing security.

Xingming Sun, He received his BS in mathematics from Hunan Normal University, China, in 1984, MS in computing science from Dalian University of Science and Technology, China, in 1988, and PhD in computing science from Fudan University, China, in 2001. He is currently a professor in School of Computer & Software, Nanjing University of Information Science & Technology, China. His research interests include network and information security, digital watermarking.



Zhihua Xia, He received his BS in Hunan City University, China, in 2006, PhD in computer science and technology from Hunan University, China, in 2011. He works as a lecturer in School of Computer & Software, Nanjing University of Information Science & Technology. His research interests include cloud computing security, and digital forensic.



Qiuqu Ji, She is currently pursuing her BS in computer science and technology at the College of Computer and Software, in Nanjing University of Information Science & Technology, China. Her research interest is cloud computing security and android.

