Attribute-Based Encryption with Attribute Revocation and Grant Function Using Proxy Re-encryption and Attribute Key for Updating

Takeru Naruse[1], Masami Mohri[2], and Yoshiaki Shiraishi[1]

[1] Nagoya Institute of Technology, Aichi 466-8555, Japan
naruse.takeru@nitzlab.com
zenmei@nitech.ac.jp
[2] Gifu University, Gifu 501-1193, Japan
mmohri@gifu-u.ac.jp

Abstract. Ciphertext-Policy Attribute-Based Encryption (CP-ABE) is suitable for data access control on a cloud storage system. In CP-ABE, the data owner encrypts data under the access structure over attributes and a set of attributes assigned to users is embedded in user's secret key. A user is able to decrypt if his attributes satisfy the ciphertext's access structure. In CP-ABE, processes of user's attribute revocation and grant are concentrated on the authority and the data owner. In this paper, we propose a ciphertext-policy attribute-based encryption scheme delegating attribute revocation processes to Cloud Server by proxy re-encryption. The proposed scheme does not require generations of new secret key when granting attributes to a user and supports any Linear Secret Sharing Schemes (LSSS) access structure.

Keywords: cryptographic cloud storage, CP-ABE, attribute revocation and grant, proxy re-encryption.

1 Introduction

Sharing of data on a cloud storage has a risk of information leakage caused by service provider's abuse. In order to protect data, the data owner encrypts data shared on the cloud storage so that only authorized users can decrypt.

Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [1][2] is suitable for data access control in the cloud storage system. The authority manages the attributes in the system. The data owner chooses an access structure and encrypts message under the access structure. The set of attributes assigned to users is embedded in his secret key. A user is able to decrypt a ciphertext if his attributes satisfy the ciphertext's access structure.

There are user's attribute revocation and grant in CP-ABE. In simple processes of user's attribute revocation, when his attributes are revoked, the data owner re-encrypts the shared data so that revoked user cannot decrypt. Then, the authority redistributes new secret keys so that other users can decrypt. In simple processes of user's attribute grant, the authority generates a new secret key. These simple processes are concentrated on the data owner and the authority.

Some attribute revocable CP-ABE schemes have been proposed [3-5]. Yu et al. [3] proposed a scheme combining CP-ABE with proxy re-encryption. The authority can delegate re-encryption and secret key update to proxy servers. However, this scheme has a limitation in access policy because it can only express "AND" policy. Hur et al. [4] proposed a scheme using key encryption keys (KEKs). A service provider distributes KEKs to each user. The service provider re-encrypts a ciphertext by an attribute group key. Then, he encrypts attribute group key by using KEKs so that authorized user can decrypt. As the number of users system has increases, the number of KEKs also increases and management becomes complicated. Liangu et al. [5] proposed a scheme using user information (UI). UI is generated by Revocation Tree and Revocation List. An authorized user can decrypt ciphertexts by using secret key and UI. In this scheme, users whose attributes are revoked lose the access rights to all shared data by attribute revocation processes.

Moreover, in these schemes [3-5], the authority needs to generate a new key when granting attribute to users.

In this paper, we propose a CP-ABE scheme delegating attribute revocation processes to Cloud Server by proxy re-encryption and meets the following requirements.

1) Support any Linear Secret Sharing Scehems (LSSS) access structure.
2) Revoke the only specified attribute (attribute level user revocation).
3) Does not require the generation of new secret key when granting attribute to user.

2 Preliminaries

2.1 Bilinear Maps

Let G_1, G_2 be two cyclic groups of prime order p. Let P be a generator of G_1. A bilinear map is a map $e : G_1 \times G_1 \rightarrow G_2$ with the following properties:

1. Bilinearity: for all $P, Q \in \mathrm{G}_1$ and $a, b \in Z_p$, we have $e(aP, bQ) \rightarrow e(P, Q)^{ab}$.
2. Non-degeneracy: $e(P, P) \neq 1$.
3. Computability: There is an efficient algorithm to compute $e(P, Q)$ for all $P, Q \in G_1$.

2.2 Linear Secret Sharing Scheme (LSSS)

Definition 1 (Linear Secret Sharing Schemes (LSSS) [2][6]). A secret-sharing scheme Π 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 an M with l rows and n columns called the share-generating matrix for Π. For all $i = 1, \dots, l$, the i'th row of M we let the function ρ defined the party labeling row i as $\rho(i)$. When we consider the column vector

$v = (s, r_2, \dots, r_n)$, where $s \in Z_p$ is the secret to be shared, and $r_2, \dots, r_n \in Z_p$ are randomly chosen, then Mv is the vector of l shares of the secret s according to Π. The share $(Mv)_i$ belongs to party $\rho(i)$.

Suppose that Π is an LSSS for the access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be any authorized set, and let $I \subset \{1,2,\dots,l\}$. 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 Π, then $\sum_{i \in I} \omega_i \lambda_i = s$. Futhermore, there these constants $\{\omega_i\}$ can be found in time polynomial in the size of the share-generating matrix M [6].

3 Our Scheme

3.1 Model

There are four entities in the proposed scheme as follows.

User: The user downloads the shared data from Cloud Server.
Data owner: The data owner encrypts the shared data then uploads to Cloud Server.
Authority: The authority manages attributes in the system and publishes the parameters used for encryption. It generates a secret key that user's attributes are embedded and PRE keys used for re-encryption and updating secret key. The authority is trusted party.
Cloud Server: Cloud Server stores shared data. It re-encrypts encrypted shared data and update secret key by using PRE keys received from the authority. Similar to previous schemes [3][4], we assume Cloud Server to be curious-but-honest. That is, it will honestly execute the tasks assigned by legitimate parties in the system. However, it would like to learn information of encrypted shared data as much as possible.

3.2 Overview

The proposed scheme is based on Waters's scheme of CP-ABE [2]. Water's scheme supports any LSSS access structure. We apply the idea of attribute revocation shown in [3] to the proposed scheme. In the proposed scheme, the attribute key is included in the ciphertext and secret key to delegate attribute revocation processes to Cloud Server. The attribute key is master key components corresponding to each attribute in the system. When user's attributes are revoked, the authority re-defines the attribute keys, and generates PRE keys for updating the attribute keys. Cloud Server re-encrypts ciphertext and updates secret key by updating attribute key by using PRE key. Each attribute is associated with version number for updating attribute key.

Cloud Server keeps user list UL, re-encryption key list RKL and the key for granting an attribute to secret key J. UL records user's ID, user's attribute information, secret key components, t_{ID}. t_{ID} is a random number that randomize each secret key to prevent users' collusion attack. t_{ID} should "bind" components of one user's key together so that they cannot be combined with another user's key components[2]. RKL records update history of attribute (version number) and PRE keys.

When granting attributes to users, Cloud Server generates user's secret key components correspond to granting attribute from t_{ID} and J, and sends secret key component to the user. The user joins secret key component to own secret key. Thus, it is possible to grant attributes to users without generation of new secret key by the authority.

3.3 Algorithm

Auth.Setup(U). The setup algorithm takes as input the number of system attributes U. It first chooses a group G_1 of prime order p, a generator $P \in G_1$. It then chooses random group elements $Q_1, \dots, Q_U \in G_1$ that are associated with the U attributes in the system. In addition, it chooses two random $\alpha, a \in Z_p$, and random $Att_1, \dots, Att_U \in Z_p$ as the attribute key.

The public parameters are

$$PK := < P, e(P,P)^{\alpha}, aP, Q_1, \dots, Q_U, T_1 = Att_1 P, \dots, T_U >.$$

The master key is $MK := < \alpha, Att_1, \dots, Att_U >$.

The keys for granting an attribute are $J := < \{x, J_x = 1/Att_x\}_{1 \le x \le U} >$.

DO.Enc$(\boldsymbol{PK}, (\boldsymbol{M}, \boldsymbol{\rho}), \boldsymbol{\mathcal{M}})$. The Encryption algorithm takes as input the public parameters $\boldsymbol{PK}$, an LSSS access structure $(\boldsymbol{M}, \boldsymbol{\rho})$, and a message $\boldsymbol{\mathcal{M}}$. The function $\boldsymbol{\rho}$ associates rows of $\boldsymbol{M}$ to attributes. Let $\boldsymbol{M}$ be an $\boldsymbol{l} \times \boldsymbol{n}$ matrix. It first chooses a random vector $\vec{\boldsymbol{v}} = (\boldsymbol{s}, \boldsymbol{y_2}, \dots, \boldsymbol{y_n}) \in \boldsymbol{Z_p}$. For $\boldsymbol{i} = \mathbf{1}$ **to** $\boldsymbol{l}$, it computes $\boldsymbol{\lambda_i} := \vec{\boldsymbol{v}} \cdot \boldsymbol{M_i}$. It then chooses random $\boldsymbol{r_1}, \dots, \boldsymbol{r_l} \in \boldsymbol{Z_p}$ and outputs the ciphertext

$$CT := < C, C', (C_1, D_1), \dots, (C_l, D_l) > =$$

$$< Ke(P,P)^{\alpha s}, sP, \left(\lambda_1(aP) - r_1 Q_{\rho(1)}, r_1 T_{\rho(1)}\right), \dots, \left(\lambda_l(aP) - r_l Q_{\rho(l)}, r_l T_{\rho(l)}\right) >$$

with (M, ρ).

Auth.Ext$(\boldsymbol{MK}, \boldsymbol{S})$. The key extraction algorithm takes as input the master key $\boldsymbol{MK}$, and a set of attributes $\boldsymbol{S}$. It first chooses a random $\boldsymbol{t_{ID}} \in \boldsymbol{Z_p}$. It then outputs t_{ID} and the secret key

$$SK := < K, L, \forall x \in S\ K_x > =$$

$$< \alpha P + t_{ID}(aP), t_{ID} P, \forall x \in S\ (t_{ID}/Att_x) Q_x >.$$

U.Dec$(\boldsymbol{SK}, \boldsymbol{CT})$. The decryption algorithm takes as input a secret key $\boldsymbol{SK}$ for a set $\boldsymbol{S}$ and a ciphertext $\boldsymbol{CT}$ for access structure $(\boldsymbol{M}, \boldsymbol{\rho})$. Suppose that $\boldsymbol{S}$ satisfies the access structure and let $\boldsymbol{I}$ be defined as $\boldsymbol{I} = \{\boldsymbol{i} : \boldsymbol{\rho}(\boldsymbol{i}) \in \boldsymbol{S}\}$. Then, let $\{\boldsymbol{\omega_i} \in \boldsymbol{Z_p}\}_{\boldsymbol{i} \in \boldsymbol{I}}$ be as set of consistants such that if $\{\boldsymbol{\lambda_i}\}$ are valid shares of the secret $\boldsymbol{s}$ according to $\boldsymbol{M}$, then $\sum_{i \in I} \omega_i \lambda_i = s$.

The decryption algorithm first computes

$$\frac{e(C',K)}{\prod_{i\in I}(e(C_i,L)e(D_i,K_{\rho(i)}))^{\omega_i}} =$$

$$\frac{e(P,P)^{\alpha s}e(P,P)^{ast_{ID}}}{\prod_{i\in I}(e(P,P)^{ta\lambda_i\omega_i})} = e(P,P)^{\alpha s}.$$

It can then decrypt the message $\mathcal{M} = C/e(P,P)^{\alpha s}$.

Auth.ReKeyGen$(\boldsymbol{MK},\boldsymbol{\gamma})$. The re-encryption key generation algorithm takes as input the master key $\boldsymbol{MK}$ and a set of attributes $\boldsymbol{\gamma}$ for update. For each $\boldsymbol{x}\in\boldsymbol{\gamma}$, it chooses random $\boldsymbol{Att'_x}\in\boldsymbol{Z_p}$ as the new attribute key, and computes $T'_x := Att'_xP$, $\boldsymbol{rk}_{x\to x'} := \frac{Att'_x}{Att_x}$. It then replaces each $\boldsymbol{Att_x}$ of the master key component with $\boldsymbol{Att'_x}$, and each $\boldsymbol{T_x}$ of public parameter with $\boldsymbol{T'_x}$. It outputs the redefined the master key $\boldsymbol{MK'}$, the redefined public parameters $\boldsymbol{PK'}$, and the PRE keys $\boldsymbol{rk} := \{\boldsymbol{x},\boldsymbol{rk_x}\}_{x\in\gamma}$.

C.ReEnc$(\boldsymbol{y}(=\boldsymbol{\rho(i)}),\boldsymbol{D_i},\boldsymbol{RKL_y})$. The re-encryption algorithm takes as input an attribute $\boldsymbol{y}(=\boldsymbol{\rho(i)})$ for update, the ciphertext component $\boldsymbol{D_i}$ and a PRE key list $\boldsymbol{RKL_y}$. It first checks version of attribute $\boldsymbol{y}$. If $\boldsymbol{y}$ has the latest version, it outputs $\perp$ and exit. Let $\boldsymbol{Att_{y^{(n)}}}$ be defined as an attribute key of the latest version of attribute $\boldsymbol{y}$. It computes $rk_{y\leftrightarrow y^{(n)}} := rk_{y\leftrightarrow y'}\cdot rk_{y'\leftrightarrow y''}\cdots rk_{y^{(n-1)}\leftrightarrow y^{(n)}} = Att_{y^{(n)}}/Att_y$. Then, it outputs the re-encrypted ciphertext component $\boldsymbol{D'_i} := rk_{y\leftrightarrow y^{(n)}}\cdot D_i = (Att_{y^{(n)}}/Att_y)\ \cdot r_iAtt_yP = r_iAtt_{y^{(n)}}P$.

C.ReKey$\left(\boldsymbol{w},\boldsymbol{K_{w,ID}},\boldsymbol{RKL_w}\right)$. The key regeneration algorithm takes as input an attribute $\boldsymbol{w}$ for update, the secret key component $\boldsymbol{K_w}$ and the PRE key list $\boldsymbol{RKL_w}$. It first checks version of attribute $\boldsymbol{w}$. If $\boldsymbol{w}$ has the latest version, it outputs $\perp$ and exit. Let $\boldsymbol{Att_{w^{(n)}}}$ be defined as the attribute key for the latest version of attribute $\boldsymbol{w}$. It computes $rk_{w\leftrightarrow w^{(n)}} := rk_{w\leftrightarrow w'}\cdot rk_{w'\leftrightarrow w''}\cdots rk_{w^{(n-1)}\leftrightarrow w^{(n)}} = Att_{w^{(n)}}/Att_w$. It then outputs the updated secret key component $K'_w := rk^{-1}_{w\leftrightarrow w^{(n)}}\cdot K_w = \left(Att_w/Att_{w^{(n)}}\right)\cdot(\boldsymbol{t_{ID}}/Att_w)Q_w = (\boldsymbol{t_{ID}}/Att_{w^{(n)}})\,Q_w$.

C.GrantAtt$(\boldsymbol{v},\boldsymbol{J_v},\boldsymbol{t_{ID}},\boldsymbol{RKL_v})$. The attribute grant algorithm takes as input an attribute $\boldsymbol{v}$, the key of granting an attribute $\boldsymbol{J_v}$, $\boldsymbol{t_{ID}}$ and the PRE key list $\boldsymbol{RKL_v}$. It first checks version of attribute $\boldsymbol{v}$. Let $\boldsymbol{Att_{v^{(n)}}}$ be defined as the attribute key for the latest version of attribute $\boldsymbol{v}$. It first computes $rk_{v\leftrightarrow v^{(n)}} := rk_{v\leftrightarrow v'}\cdot rk_{v'\leftrightarrow v''}\cdots rk_{v^{(n-1)}\leftrightarrow v^{(n)}} = Att_{v^{(n)}}/Att_v$. It then outputs secret key component for $K_v := \boldsymbol{t_{ID}}\cdot rk^{-1}_{v\leftrightarrow v^{(n)}}\cdot J_v = \boldsymbol{t_{ID}}\cdot\left(Att_v/Att_{v^{(n)}}\right)\cdot(1/Att_v)Q_v = \left(t_{ID}/Att_{v^{(n)}}\right)Q_v$ and redefines the key of granting an attribute $\boldsymbol{J'_v} := rk^{-1}_{v\leftrightarrow v^{(n)}}\cdot\boldsymbol{J_v} = \left(1/Att_{v^{(n)}}\right)Q_v$.

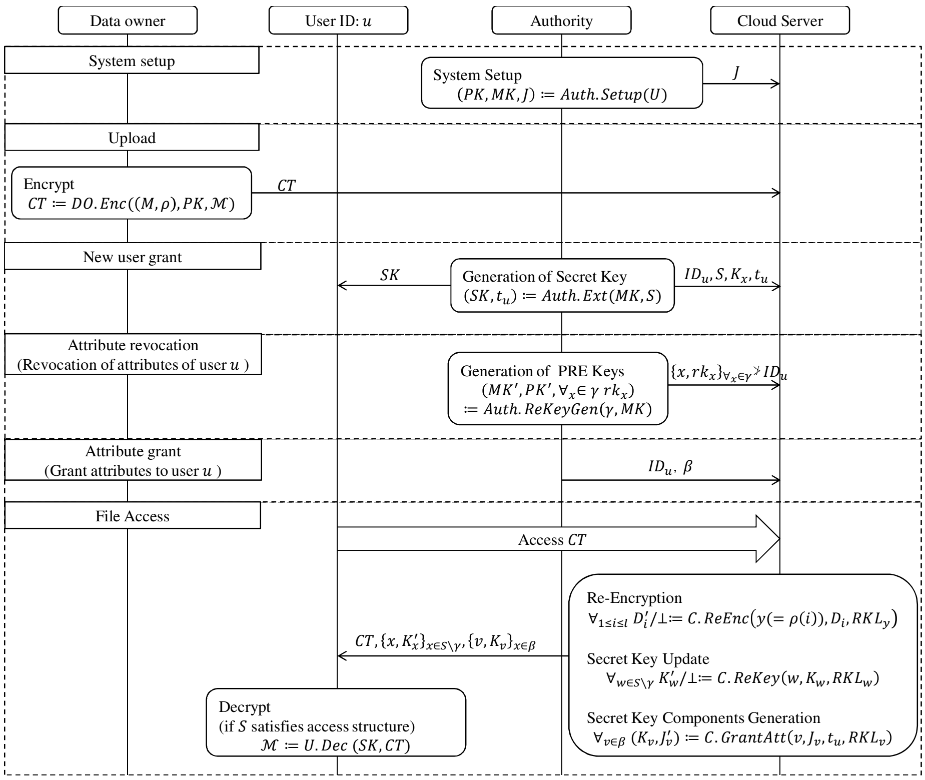
Data owner
User ID: u
Authority
Cloud Server
System setup
System Setup
$(PK, MK, J) \coloneqq Auth.Setup(U)$
J
Upload
Encrypt
$CT \coloneqq DO.Enc((M,\rho), PK, \mathcal{M})$
CT
New user grant
SK
Generation of Secret Key
$(SK, t_u) \coloneqq Auth.Ext(MK, S)$
ID_u, S, K_x, t_u
Attribute revocation
(Revocation of attributes of user u)
Generation of PRE Keys
$(MK', PK', \forall_{x} \in \gamma\ rk_x)$
$\coloneqq Auth.ReKeyGen(\gamma, MK)$
$\{x, rk_x\}_{\forall_x \in \gamma}, ID_u$
Attribute grant
(Grant attributes to user u)
ID_u, β
File Access
Access CT
Re-Encryption
$\forall_{1 \le i \le l}\ D_i'/\bot \coloneqq C.ReEnc(y(=\rho(i)), D_i, RKL_y)$
Secret Key Update
$\forall_{w \in S \setminus \gamma}\ K_w'/\bot \coloneqq C.ReKey(w, K_w, RKL_w)$
Secret Key Components Generation
$\forall_{v \in \beta}\ (K_v, J_v') \coloneqq C.GrantAtt(v, J_v, t_u, RKL_v)$
$CT, \{x, K_x'\}_{x \in S \setminus \gamma}, \{v, K_v\}_{x \in \beta}$
Decrypt
(if S satisfies access structure)
$\mathcal{M} \coloneqq U.Dec\ (SK, CT)$

Fig. 1. Flow of the proposed scheme

We show the flow of our scheme in Fig 1. In Fig 1, γ denotes a set of user u's attributes which are revoked and β denotes a set of attributes that granting to user u.

4 Security Proof

We prove that unauthorized users and Cloud Server cannot decrypt ciphertext CT that was encrypted by using the proposed scheme. Since we assume Cloud Server is honest, we do not consider active attacks from CloudServer by colluding with revoked users as in [3][4].

An unauthorized user cannot decrypt CT because his secret key does not contain components that corresponds to attributes necessary for decryption. In addition, each secret key is randomized with a freshly chosen exponent t_{ID} to prevent collusion attack. t_{ID} should “bind” the components of one user's key together so that they cannot be combined with another user's key components [2]. Therefore, unauthorized users cannot decrypt the ciphertext CT.

Cloud Server keeps secret key components K_x, t_{ID} and the keys for granting an attribute to secret key J. It can generate any K_x that corresponds attribute in the system, but CT cannot be decrypted only with K_x. Therefore, Cloud Server cannot decrypt the ciphertext CT.

Table 1. Comparison of schemes

	scheme[3]	scheme[4]	Scheme[5]	proposed scheme
Supporting Access Policy Type	‘AND’	‘AND’,‘OR’	Any LSSS	Any LSSS
Attribute level user revocation	Possible	Possible	Impossible	Possible
Grant attributes to users	The authority generates a new secret key	The authority generates a new secret key	The authority generates a new secret key	Cloud Server adds attributes to user’s secret key

5 Comparison and Conclusion

This paper proposed a ciphertext-policy attribute-based encryption scheme delegating attribute revocation processes to Cloud Server by proxy re-encryption. Cloud Server re-encrypts a ciphertext and updates a secret key by updating attribute key with PRE key for updating the attribute keys. We compared the proposed scheme with schemes of [3-5] and show the comparison result at Table 1.

The proposed scheme meets three requirements as follows;

First, the proposed scheme supports any LSSS access structure. Second, the authority can only revoke specified attribute by updating attribute key included in ciphertext corresponding to his attributes which are revoked. Finally, when granting attributes to a user, generation of a new secret key becomes unnecessary because Cloud Server generates secret key components corresponding to granting attributes.

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-Policy Attribute-Based Encryption. In: IEEE Symposium on Security and Privacy, pp. 181–194 (2007)
2. Waters, B.: Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 53–70. Springer, Heidelberg (2011)
3. Yu, S., Wang, C., Ren, K., Lou, W.: Attribute Based Data Sharing with Attribute Revocation. In: 5th ACM Symposium on Information, Computer and Communications Security, pp. 261–270 (2010)
4. Hur, J., Nor, D.K.: Attribute-Based Access Control with Efficient Revocation in Data Outsourcing Systems. IEEE Transactions on Parallel and Distributed Systems 22, 1214–1221 (2011)
5. Liang, X., Lu, R., Lin, X., Shen, X.: Ciphertext Policy Attribute Based Encryption with Efficient Revocation. Technical report, Univ. of Warterloo (2011)
6. Beimel, A.: Secure Schemes for Secret Sharing and Key Distribution. PhD thesis, Israel Institute of Technology (1996)