Expressive, Efficient, and Revocable Data Access Control for Multi-Authority Cloud Storage

Kan Yang, *Student Member, IEEE,* and Xiaohua Jia, *Fellow, IEEE*

Abstract—Data access control is an effective way to ensure the data security in the cloud. Due to data outsourcing and untrusted cloud servers, the data access control becomes a challenging issue in cloud storage systems. Ciphertext-Policy Attribute-based Encryption (CP-ABE) is regarded as one of the most suitable technologies for data access control in cloud storage, because it gives data owners more direct control on access policies. However, it is difficult to directly apply existing CP-ABE schemes to data access control for cloud storage systems because of the attribute revocation problem. In this paper, we design an expressive, efficient and revocable data access control scheme for multi-authority cloud storage systems, where there are multiple authorities co-exist and each authority is able to issue attributes independently. Specifically, we propose a revocable multi-authority CP-ABE scheme, and apply it as the underlying techniques to design the data access control scheme. Our attribute revocation method can efficiently achieve both forward security and backward security. The analysis and simulation results show that our proposed data access control scheme is secure in the random oracle model and is more efficient than previous works.

Index Terms—Access control, multi-authority, CP-ABE, attribute revocation, cloud storage

1 Introduction

Cloud storage is an important service of cloud computing [1], which offers services for data owners to host their data in the cloud. This new paradigm of data hosting and data access services introduces a great challenge to data access control. Because the cloud server cannot be fully trusted by data owners, they can no longer rely on servers to do access control. Ciphertext-Policy Attribute-based Encryption (CP-ABE) [2], [3] is regarded as one of the most suitable technologies for data access control in cloud storage systems, because it gives the data owner more direct control on access policies. In CP-ABE scheme, there is an authority that is responsible for attribute management and key distribution. The authority can be the registration office in a university, the human resource department in a company, etc. The data owner defines the access policies and encrypts data according to the policies. Each user will be issued a secret key reflecting its attributes. A user can decrypt the data only when its attributes satisfy the access policies.

There are two types of CP-ABE systems: single-authority CP-ABE [2], [3], [4], [5] where all attributes are managed by a single authority, and multi-authority CP-ABE [6], [7], [8] where attributes are from different domains and managed by different authorities. Multi-authority CP-ABE is more appropriate for data access control of cloud storage systems, as users may hold attributes issued by multiple authorities and data owners may also share the data using access policy defined over attributes from different authorities. For example, in an E-health system, data owners may share the data using the access policy ''Doctor AND Researcher'', where the attribute ''Doctor'' is issued by a medical organization and the attribute ''Researcher'' is issued by the administrators of a clinical trial. However, it is difficult to directly apply these multi-authority CP-ABE schemes to multi-authority cloud storage systems because of the attribute revocation problem.

In multi-authority cloud storage systems, users' attributes can be changed dynamically. A user may be *entitled* some new attributes or *revoked* some current attributes. And his permission of data access should be changed accordingly. However, existing attribute revocation methods [9], [10], [11], [12] either rely on a trusted server or lack of efficiency, they are not suitable for dealing with the attribute revocation problem in data access control in multi-authority cloud storage systems.

In this paper, we first propose a revocable multi-authority CP-ABE scheme, where an efficient and secure revocation method is proposed to solve the attribute revocation problem in the system. As described in Table 1, our attribute revocation method is efficient in the sense that it incurs less communication cost and computation cost, and is secure in the sense that it can achieve both *backward security* (The revoked user cannot decrypt any new ciphertext that requires the revoked attribute to decrypt) and *forward security* (The newly joined user can also decrypt the previously published ciphertexts[1], if it has sufficient

- *The authors are with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. E-mail: kan.yang@my.cityu.edu.hk; csjia@cityu.edu.hk.*

Manuscript received 30 May 2013; revised 17 Aug. 2013; accepted 22 Sept. 2013. Date of publication 3 Oct. 2013; date of current version 13 June 2014. Recommended for acceptance by K. Wu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.253

1. The previous ciphertexts may be associated with the attribute in a previous version, while the newly joined user may be issued an attribute in a new version.

TABLE 1
Comprehensive Comparison of Attribute Revocation Methods for CP-ABE Systems

Scheme	Authority	Revocation Message	Backward Security	Forward Security	Revocation Enforcer	CT Updater
[11]	Single	$O(n_{non,x}\log\frac{n_u}{n_{non,x}})$	Yes	Yes	Server*	Server*
[13]	Multiple	$O(n_{c,x}\cdot n_{non,x})$	Yes	No	Owner	Owner
[14]	Multiple	$O(n_{c,aid}+n_{non,x})$	Yes	Yes	AA	Server†
Our	Multiple	$O(n_{non,x})$	Yes	Yes	AA	Server†

*: The server is fully trusted; †: The server is semi-trusted.
$|p|$ is the size of element in the groups with the prime order p; n_u denotes the number of users in the system; $n_{non,x}$ denotes the number of non-revoked users who hold the revoked attribute x and $n_{c,x}$ is the number of ciphertexts which contain the revoked attribute x; $n_{c,aid}$ denotes the total number of attributes belongs to the AA_{aid} in all the ciphertexts.

attributes). Our scheme does not require the server to be fully trusted, because the key update is enforced by each attribute authority not the server. Even if the server is not semi-trusted in some scenarios, our scheme can still guarantee the backward security. Then, we apply our proposed revocable multi-authority CP-ABE scheme as the underlying techniques to construct the expressive and secure data access control scheme for multi-authority cloud storage systems.

Compared to the conference version [14] of this work, we have the following improvements:

1. We modify the framework of the scheme and make it more practical to cloud storage systems, in which data owners are not involved in the key generation. Specifically, a user's secret key is not related to the owner's key, such that each user only needs to hold one secret key from each authority instead of multiple secret keys associated to multiple owners.
2. We greatly improve the efficiency of the attribute revocation method. Specifically, in our new attribute revocation method, only the ciphertexts that associated with the revoked attribute needs to be updated, while in [14], all the ciphertexts that associated with *any* attribute from the authority (corresponding to the revoked attribute) should be updated. Moreover, in our new attribute revocation method, both the key and the ciphertext can be updated by using the same update key, instead of requiring the owner to generate an update information for each ciphertext, such that owners are not required to store each random number generated during the encryption.
3. We also highly improve the expressiveness of our access control scheme, where we remove the limitation that each attribute can only appear at most once in a ciphertext.

The remaining of this paper is organized as follows. We give the definition of the system model, framework and the security model in Section 2. Section 3 gives the detailed construction of our data access control scheme for multi-authority cloud storage systems. Sections 4 and 5 give the security analysis and performance analysis respectively. Section 6 gives the related work on ABE and attribute revocation methods. The conclusion is given in Section 7. In the supplemental file which is available in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/253, we give some preliminary definitions and describe the full security proof of our data access control scheme.

2 System Model and Security Model

2.1 System Model

We consider a data access control system in multi-authority cloud storage, as described in Fig. 1. There are five types of entities in the system: a certificate authority (CA), attribute authorities (AAs), data owners (owners), the cloud server (server) and data consumers (users).

The CA is a global trusted certificate authority in the system. It sets up the system and accepts the registration of all the users and AAs in the system. For each legal user in the system, the CA assigns a global unique user identity to it and also generates a global public key for this user. However, the CA is *not* involved in any attribute management and the creation of secret keys that are associated with attributes. For example, the CA can be the Social Security Administration, an independent agency of the United States government. Each user will be issued a Social Security Number (SSN) as its global identity.

Every AA is an independent attribute authority that is responsible for entitling and revoking user's attributes according to their role or identity in its domain. In our scheme, every attribute is associated with a single AA, but each AA can manage an arbitrary number of attributes. Every AA has full control over the structure and semantics of its attributes. Each AA is responsible for generating a public attribute key for each attribute it manages and a secret key for each user reflecting his/her attributes.

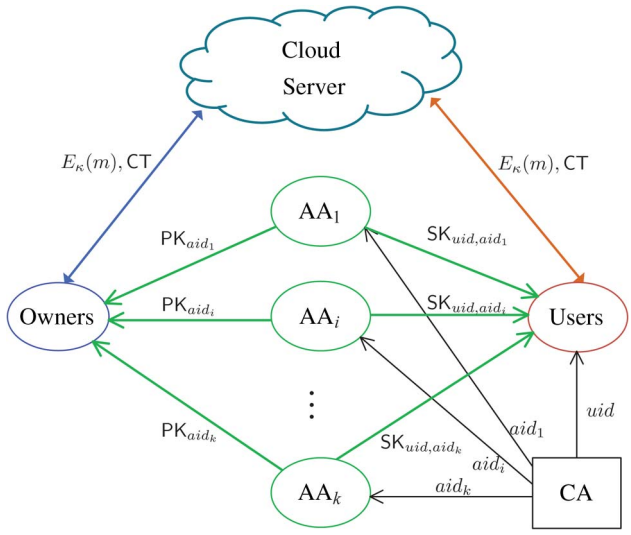


Fig. 1. System model of data access control in multi-authority cloud storage.

Each user has a global identity in the system. A user may be entitled a set of attributes which may come from multiple attribute authorities. The user will receive a secret key associated with its attributes entitled by the corresponding attribute authorities.

Each owner first divides the data into several components according to the logic granularities and encrypts each data component with different content keys by using symmetric encryption techniques. Then, the owner defines the access policies over attributes from multiple attribute authorities and encrypts the content keys under the policies. Then, the owner sends the encrypted data to the cloud server together with the ciphertexts.[2] They do not rely on the server to do data access control. But, the *access control happens inside the cryptography*. That is only when the user's attributes satisfy the access policy defined in the ciphertext, the user is able to decrypt the ciphertext. Thus, users with different attributes can decrypt different number of content keys and thus obtain different granularities of information from the same data.

2.2 Framework

The framework of our data access control scheme is defined as follows.

Definition 1 (Framework of Multi-Authority Access Control Scheme). *The framework of data access control scheme for multi-authority cloud storage systems contains the following phases:*

Phase 1: System Initialization. *This phase consists of CA setup and AA setup with the following algorithms:*

- $\mathbf{CASetup}(1^\lambda) \rightarrow (\mathsf{GMK}, \mathsf{GPP}, (\mathsf{GPK}_{uid}, \mathsf{GPK}'_{uid}), (\mathsf{GSK}_{uid}, \mathsf{GSK}'_{uid}), Certificate(uid))$. The CA setup algorithm is run by the CA. It takes no input other than the implicit security parameter λ. It generates the global master key GMK of the system and the global public parameters GPP. For each user uid, it generates the user's global public keys $(\mathsf{GPK}_{uid}, \mathsf{GPK}'_{uid})$, the user's global secret keys $(\mathsf{GSK}_{uid}, \mathsf{GSK}'_{uid})$ and a certificate $Certificate(uid)$ of the user.
- $\mathbf{AASetup}(\mathcal{U}_{aid}) \rightarrow (\mathsf{SK}_{aid}, \mathsf{PK}_{aid}, \{\mathsf{VK}_{x_{aid}}, \mathsf{PK}_{x_{aid}}\}_{x_{aid} \in \mathcal{U}_{aid}})$. The attribute authority setup algorithm is run by each attribute authority. It takes the attribute universe $\mathcal{U}_{aid}$ managed by the AA_{aid} as input. It outputs a secret and public key pair $(\mathsf{SK}_{aid}, \mathsf{PK}_{aid})$ of the AA_{aid} and a set of version keys and public attribute keys $\{VK_{x_{aid}}, PK_{x_{aid}}\}_{x_{aid} \in \mathcal{U}_{aid}}$ for all the attributes managed by the AA_{aid}.

Phase 2: Secret Key Generation by AAs.

- $\mathbf{SKeyGen}(\mathsf{GPP}, \mathsf{GPK}_{uid}, \mathsf{GPK}'_{uid}, \mathsf{GSK}_{uid}, \mathsf{SK}_{aid}, S_{uid,aid}, \{\mathsf{VK}_{x_{aid}}, \mathsf{PK}_{x_{aid}}\}_{x_{aid} \in S_{uid,aid}}) \rightarrow \mathsf{SK}_{uid,aid}$. The secret key generation algorithm is run by each AA. It takes as inputs the global public parameters GPP, the global public keys $(\mathsf{GPK}_{uid}, \mathsf{GPK}'_{uid})$ and one global secret key GSK_{uid} of the user uid, the secret key SK_{aid} of the AA_{aid}, a set of attributes $S_{uid,aid}$ that describes the user uid from the AA_{aid} and its corresponding version keys $\{\mathsf{VK}_{x_{aid}}\}$ and public attribute keys $\{\mathsf{PK}_{x_{aid}}\}$. It outputs a secret key $SK_{uid,aid}$ for the user uid which is used for decryption.

Phase 3: Data Encryption by Owners. *Owners first encrypt the data m with content keys by using symmetric encryption methods, then they encrypt the content keys by running the following encryption algorithm:*

- $\mathbf{Encrypt}(\mathsf{GPP}, \{\mathsf{PK}_{aid_k}\}_{aid_k \in I_\mathcal{A}}, \kappa, \mathbb{A}) \rightarrow \mathsf{CT}$. The encryption algorithm is run by the data owner to encrypt the content keys. It takes as inputs the global public parameters GPP, a set of public keys $\{\mathsf{PK}_{aid_k}\}_{aid_k \in I_\mathcal{A}}$ for all the AAs in the encryption set $I_\mathcal{A}$[3], the content key κ and an access policy $\mathbb{A}$.[4] The algorithm encrypts κ according to the access policy and outputs a ciphertext CT. We will assume that the ciphertext implicitly contains the access policy $\mathbb{A}$.

Phase 4: Data Decryption by Users. *Users first run the decryption algorithm to get the content keys, and use them to further decrypt the data.*

- $\mathbf{Decrypt}\ (\mathsf{CT}, \mathsf{GPK}_{uid}, \mathsf{GSK}'_{uid}, \{\mathsf{SK}_{uid,aid_k}\}_{aid_k \in I_\mathcal{A}}) \rightarrow \kappa$. The decryption algorithm is run by users to decrypt the ciphertext. It takes as inputs the ciphertext CT which contains an access policy $\mathbb{A}$, a global public key GPK_{uid} and a global secret key GSK'_{uid} of the user uid, and a set of secret keys $\{\mathsf{SK}_{uid,aid_k}\}_{aid_k \in I_\mathcal{A}}$ from all the involved AAs. If the attributes $\{S_{uid,aid_k}\}_{aid_k \in I_\mathcal{A}}$ of the user uid satisfy the access policy $\mathbb{A}$, the algorithm will decrypt the ciphertext and return the content key κ.

Phase 5: Attribute Revocation. *This phase contains three steps: Update Key Generation by AAs, Secret Key Update by Non-revoked Users[5] and Ciphertext Update by Server.*

- $\mathbf{UKeyGen}(\mathsf{SK}_{aid'}, \tilde{x}_{aid'}, \mathsf{VK}_{\tilde{x}_{aid'}}) \rightarrow (\widetilde{\mathsf{VK}}_{\tilde{x}_{aid'}}, \mathsf{UK}_{s,\tilde{x}_{aid'}}, \mathsf{UK}_{c,\tilde{x}_{aid'}})$. The update key generation algorithm is run by the corresponding $AA_{aid'}$ that manages the revoked attribute $\tilde{x}_{aid'}$. It takes as inputs the secret key $\mathsf{SK}_{aid'}$ of $AA_{aid'}$, the revoked attribute $\tilde{x}_{aid'}$ and its current version key $\mathsf{VK}_{\tilde{x}_{aid'}}$. It outputs a new version key $\widetilde{\mathsf{VK}}_{\tilde{x}_{aid'}}$ and the update key $\mathsf{UK}_{s,\tilde{x}_{aid'}}$ (for secret key update) and the update key $\mathsf{UK}_{c,\tilde{x}_{aid'}}$ (for ciphertext update).
- $\mathbf{SKUpdate}(\mathsf{SK}_{uid,aid'}, \mathsf{UK}_{s,\tilde{x}_{aid'}}) \rightarrow \widetilde{\mathsf{SK}}_{uid,aid'}$. The secret key update algorithm is run by each non-revoked user uid. It takes as inputs the current secret key of the non-revoked user $\mathsf{SK}_{uid,aid'}$ and the update key $\mathsf{UK}_{s,\tilde{x}_{aid'}}$. It

2. In this paper, we simply use the ciphertext to denote the encrypted content keys with CP-ABE.

3. Note that not all the AAs are involved in the encryption. We use *encryption set* $I_\mathcal{A}$ to denote the set of those AAs involved in the encryption.

4. The access policy is a LSSS structure (M, ρ), which is defined in the supplemental file available online.

5. We denote those users who possess the revoked attributes $\tilde{x}_{aid'}$ but have not be revoked as the *non-revoked users*.

outputs a new secret key $\widetilde{\mathsf{SK}}_{uid,aid'}$ for each non-revoked user uid.

- $\mathbf{CTUpdate}(\mathsf{CT}, \mathsf{UK}_{c,\tilde{x}_{aid'}}) \rightarrow \widetilde{\mathsf{CT}}$. The ciphertext update algorithm is run by the cloud server. It takes as inputs the ciphertexts which contain the revoked attribute $\tilde{x}_{aid'}$, and the update key $\mathsf{UK}_{c,\tilde{x}_{aid'}}$. It outputs new ciphertexts $\widetilde{\mathsf{CT}}$ which contain the latest version of the revoked attribute $\tilde{x}_{aid'}$.

2.3 Security Model

In multi-authority cloud storage systems, we make the following assumptions:

- The CA is fully trusted in the system. It will not collude with any user, but it should be prevented from decrypting any ciphertexts by itself.
- Each AA is trusted but can be corrupted by the adversary.
- The server is curious but honest. It is curious about the content of the encrypted data or the received message, but will execute correctly the task assigned by each attribute authority.
- Each user is dishonest and may collude to obtain unauthorized access to data.

2.3.1 Decisional q-Parallel Bilinear Diffie-Hellman Exponent Assumption

We recall the definition of the decisional q-parallel Bilinear Diffie-Hellman Exponent (q-parallel BDHE) problem in [3] as follows. Chooses a group $\mathbb{G}$ of prime order p according to the security parameter. Let $a, b_1, \ldots, b_q, s \in \mathbb{Z}_p$ be chosen at random and g be a generator of $\mathbb{G}$. If an adversary is given

$$\vec{y}=\Big(g, g^s, g^a, \ldots, g^{(a^q)}, , \ g^{(a^{q+2})}, \ldots, g^{(a^{2q})}$$
$$\forall_{1\leq j\leq q} \ g^{s\cdot b_j}, \ g^{a/b_j}, \ldots, g^{(a^q/b_j)}, , g^{(a^{q+2}/b_j)}, \ldots, g^{(a^{2q}/b_j)}$$
$$\forall_{1\leq j,l\leq q, l\neq j} \ g^{a\cdot s\cdot b_l/b_j}, \ldots, g^{(a^q\cdot s\cdot b_l/b_j)}\Big),$$

it must be hard to distinguish a valid tuple $e(g,g)^{a^{q+1}s} \in \mathbb{G}_T$ from a random element R in $\mathbb{G}_T$.

An algorithm $\mathcal{B}$ that outputs $z \in \{0,1\}$ has advantage ϵ in solving q-parallel BDHE in $\mathbb{G}$ if

$$\Big|Pr\Big[\mathcal{B}(\vec{y}, T=e(g,g)^{a^{q+1}s})=0\Big]-Pr[\mathcal{B}(\vec{y}, T=R)=0]\Big|\geq\epsilon.$$

Definition 2. *The decisional q-parallel BDHE assumption holds if no polynomial time algorithm has a non-negligible advantage in solving the q-parallel BDHE problem.*

2.3.2 Security Model

We now describe the security model for our revocable multi-authority CP-ABE systems by the following game between a challenger and an adversary. Similar to the identity-based encryption schemes [15], the security model allows the adversary to query for any secret keys and update keys that cannot be used to decrypt the challenge ciphertext. We assume that the adversaries can corrupt authorities only statically similar to [6], [7], [8], but key queries are made adaptively. Let $S_{\mathcal{A}}$ denote the set of all the attribute authorities. The security game is defined as follows.

Setup. The global public parameters are generated by running the CA setup algorithm. The adversary specifies a set of corrupted attribute authorities $S'_{\mathcal{A}} \subset S_{\mathcal{A}}$. The challenger generates the public keys by running the attribute authority setup algorithm and generates the secret keys by running the secret key generation algorithm. For uncorrupted attribute authorities in $S_{\mathcal{A}} - S'_{\mathcal{A}}$, the challenger only sends the public keys to the adversary. For corrupted authorities in $S'_{\mathcal{A}}$, the challenger sends both the public keys and secret keys to the adversary. The adversary can also get the global public parameters.

Phase 1. The adversary makes secret key queries by submitting pairs (uid, S_{uid}) to the challenger, where $S_{uid} = \{S_{uid,aid_k}\}_{aid_k \in S_{\mathcal{A}}-S'_{\mathcal{A}}}$ is a set of attributes belonging to several uncorrupted AAs, and uid is a user identifier. The challenger gives the corresponding set of secret keys $\{SK_{uid,aid_k}\}$ to the adversary. The adversary also makes update key queries by submitting a set of attributes S'_{aid}. The challenger gives the corresponding update keys to the adversary.

Challenge. The adversary submits two equal length messages m_0 and m_1. In addition, the adversary gives a challenge access structure (M^*, ρ^*) which must satisfy the following constraints: Let V denote the subset of rows of M^* labeled by attributes controlled by corrupted AAs. For each uid, let V_{uid} denote the subset of rows of M^* labeled by attributes x belongs to the attribute sets that the adversary has queried. For each uid, we require that the subspace spanned by $V \cup V_{uid}$ must not include $(1, 0, \ldots, 0)$. In other words, the adversary cannot ask for a set of keys that allow decryption, in combination with any keys that can obtained from corrupted AAs. The challenger then flips a random coin c, and encrypts m_c under the access structure (M^*, ρ^*). Then, the ciphertext CT^* is given to the adversary.

Phase 2. The adversary may query more secret keys and update keys, as long as they do not violate the constraints on the challenge access structure (M^*, ρ^*) and the following constraints: None of the updated secret keys (generated by the queried update keys and the queried secret keys[6]) is able to decrypt the challenged ciphertexts. In other words, the adversary is not able to query the update keys that can update the queried secret keys to the new secret keys that can decrypt the challenge ciphertext.

Guess. The adversary outputs a guess c' of c.

The advantage of an adversary A in this game is defined as $Pr[c' = c] - \frac{1}{2}$.

Definition 3. *A revocable multi-authority CP-ABE scheme is secure against static corruption of authorities if all polynomial time adversaries have at most a negligible advantage in the above security game.*

6. There is another reason that makes the queried secret keys cannot decrypt the challenge ciphertext. That is at least one of the attributes in the previous queried secret keys may be not in the current version.

3 Our Data Access Control Scheme

In this section, we first give an overview of the challenges and techniques. Then, we propose the detailed construction of our access control scheme which consists of five phases: System Initialization, Key Generation, Data Encryption, Data Decryption and Attribute Revocation.

3.1 Overview

To design the data access control scheme for multi-authority cloud storage systems, the main challenging issue is to construct the underlying *Revocable Multi-authority CP-ABE* protocol. In [6], Chase proposed a multi-authority CP-ABE protocol, however, it cannot be directly applied as the underlying techniques because of two main reasons: 1) *Security Issue*: Chase's multi-authority CP-ABE protocol allows the central authority to decrypt all the ciphertexts, since it holds the master key of the system; 2) *Revocation Issue*: Chase's protocol does not support attribute revocation.

We propose a new revocable multi-authority CP-ABE protocol based on the single-authority CP-ABE proposed by Lewko and Waters in [16]. That is we extend it to multi-authority scenario and make it revocable. We apply the techniques in Chase's multi-authority CP-ABE protocol [6] to tie together the secret keys generated by different authorities for the same user and prevent the collusion attack. Specifically, we separate the functionality of the authority into a global certificate authority (CA) and multiple attribute authorities (AAs). The CA sets up the system and accepts the registration of users and AAs in the system.[7] It assigns a global user identity uid to each user and a global authority identity aid to each attribute authority in the system. Because the uid is globally unique in the system, secret keys issued by different AAs for the same uid can be tied together for decryption. Also, because each AA is associated with an aid, every attribute is distinguishable even though some AAs may issue the same attribute.

To deal with the security issue in [6], instead of using the system unique public key (generated by the unique master key) to encrypt data, our scheme requires all attribute authorities to generate their own public keys and uses them to encrypt data together with the global public parameters. This prevent the certificate authority in our scheme from decrypting the ciphertexts.

To solve the attribute revocation problem, we assign a version number for each attribute. When an attribute revocation happens, only those components associated with the revoked attribute in secret keys and ciphertexts need to be updated. When an attribute of a user is revoked from its corresponding AA, the AA generates a new version key for this revoked attribute and generates an update key. With the update key, all the users, except the revoked user, who hold the revoked attributes can update its secret key (Backward Security). By using the update key, the components associated with the revoked attribute in the ciphertext can also be updated to the current version. To improve the efficiency, we delegate the workload of ciphertext update to the server by using the proxy re-encryption method, such that the newly joined user is also able to decrypt the previously published data, which are encrypted with the previous public keys, if they have sufficient attributes (Forward Security). Moreover, by updating the ciphertexts, all the users need to hold only the latest secret key, rather than to keep records on all the previous secret keys.

7. Note that the CA is *not* involved in any attribute management and any secret key generation.

3.2 System Initialization

The system initialization contains CA Setup and AA Setup.

3.2.1 CA Setup

The CA sets up the system by running the CA setup algorithm $\mathsf{CASetup}$, which takes a security parameter as input. The CA first chooses two multiplicative groups $\mathbb{G}$ and $\mathbb{G}_T$ with the same prime order p and a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. It also choose a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}$ that matches the string to an element in $\mathbb{G}$, such that the security will be modeled in the random oracle. Then, the CA chooses two random numbers $a, b \in \mathbb{Z}_p$ as the global master key $\mathsf{GMK} = (a, b)$ of the system and computes the global public parameters as

$$\mathsf{GPP} = (g, g^a, g^b, H).$$

The CA accepts both *User Registration* and *AA Registration*.

1) *User Registration*: Every user should register to the CA during the system initialization. If the user is a legal user in the system, the CA then assigns a globally unique user identity uid to this user. For each user uid, the CA first generates two random numbers $u_{uid}, u'_{uid} \in \mathbb{Z}_p$ as its global secret keys

$$\mathsf{GSK}_{uid} = u_{uid}, \mathsf{GSK}'_{uid} = u'_{uid}.$$

It then generates the user's global public keys as

$$\mathsf{GPK}_{uid} = g^{u_{uid}}, \mathsf{GPK}'_{uid} = g^{u'_{uid}}.$$

The CA also generates a certificate $Certificate(uid)$ for the user uid. Then, the CA sends one of the user's global public keys GPK_{uid}, one global secret key GSK'_{uid} and the Certificate $Certificate(uid)$ to the user uid.

2) *AA Registration:* Each AA should also register itself to the CA during the system initialization. If the AA is a legal authority in the system, the CA first assigns a global attribute authority identity aid to this AA. Then, the CA sends the other global public/secret key of each user $(\mathsf{GPK}'_{uid}, \mathsf{GSK}_{uid})$ to the AA_{aid}. It also sends a verification key to the AA_{aid}, which can be used to verify the certificates of users issued by the CA.

3.2.2 AA Setup

Let S_{aid} denote the set of all attributes managed by each attribute authority AA_{aid}. It chooses three random numbers $\alpha_{aid}, \beta_{aid}, \gamma_{aid} \in \mathbb{Z}_p$ as the authority secret key

$$\mathsf{SK}_{aid} = (\alpha_{aid}, \beta_{aid}, \gamma_{aid}),$$

where α_{aid} is used for data encryption, β_{aid} is used to distinguish attributes from different AAs and γ_{aid} is used

CT_1	$E_{\kappa_1}(m_1)$	···	···	CT_n	$E_{\kappa_n}(m_n)$

Fig. 2. Format of data on cloud server.

for attribute revocation. It also generates the public key PK_{aid} as

$$\mathsf{PK}_{aid} = \left(e(g,g)^{\alpha_{aid}}, g^{\beta_{aid}}, g^{\frac{1}{\beta_{aid}}}\right).$$

For each attribute $x_{aid} \in S_{aid}$, the AA_{aid} generates a public attribute key as

$$\mathsf{PK}_{x_{aid}} = \left(\mathsf{PK}_{1,x_{aid}} = H(x_{aid})^{v_{x_{aid}}}, \mathsf{PK}_{2,x_{aid}} = H(x_{aid})^{v_{x_{aid}}\gamma_{aid}}\right)$$

by implicitly choosing an attribute version key as $\mathsf{VK}_{x_{aid}} = v_{x_{aid}}$. All the public attribute keys $\{\mathsf{PK}_{x_{aid}}\}_{x_{aid}\in S_{aid}}$ are published on the public bulletin board of the AA_{aid}, together with the public key PK_{aid} of the AA_{aid}.

3.3 Secret Key Generation

Each user uid is required to authenticate itself to the AA_{aid} before it can be entitled some attributes from the AA_{aid}. The user submits its certificate $Certificate(uid)$ to the AA_{aid}. The AA_{aid} then authenticates the user by using the verification key issued by the CA.

If it is a legal user, the AA_{aid} entitles a set of attributes $S_{uid,aid}$ to the user uid according to its role or identity in its administration domain. Otherwise, it aborts. Then, the AA_{aid} generates the user's secret key $\mathsf{SK}_{uid,aid}$ by running the secret key generation algorithm $\mathsf{SKeyGen}$. It chooses a random number $t_{uid,aid} \in \mathbb{Z}_p$ and computes the user's secret key as

$$\mathsf{SK}_{uid,aid} = \Big(K_{uid,aid} = g^{\alpha_{aid}} g^{au_{uid}} g^{bt_{uid,aid}}, K'_{uid,aid} = g^{t_{uid,aid}},$$
$$\forall x_{aid} \in S_{uid,aid} : K_{x_{aid},uid} = g^{u'_{uid} t_{uid,aid}\beta_{aid}} H(x_{aid})^{v_{x_{aid}}\beta_{aid}(u_{uid}+\gamma_{aid})}\Big).$$

If the user uid does not hold any attribute from AA_{aid}, the secret key $\mathsf{SK}_{uid,aid}$ only contains the first component $K_{uid,aid}$.

3.4 Data Encryption

Before hosting data m to cloud servers, the owner processes the data as follows.

1) It divides the data into several data components as $m = \{m_1, \ldots, m_n\}$ according to the logic granularities. For example, the personal data may be divided into {name, address, security number, employer, salary}.
2) It encrypts data components with different content keys $\{\kappa_1, \ldots, \kappa_n\}$ by using symmetric encryption methods.
3) It then defines an access structure M_i for each content key $\kappa_i (i = 1, \ldots, n)$ and encrypts it by running the encryption algorithm $\mathsf{Encrypt}$.

The encryption algorithm $\mathsf{Encrypt}$ takes as inputs the global public parameters GPP, a set of public keys $\{\mathsf{PK}_{aid_k}\}_{aid_k\in I_{\mathcal{A}}}$ for all the AAs in the encryption set $I_{\mathcal{A}}$, the content key κ and an access structure (M, ρ) over all the involved attributes. Let M be a $\ell \times n$ matrix, where ℓ denotes the total number of all the attributes. The function ρ maps each row of M to an attribute. In this construction, we remove the limitation that ρ should be an injective function (i.e., an attribute can associate with more than one rows of M).

To encrypt the content key κ, the encryption algorithm first chooses a random encryption exponent $s \in \mathbb{Z}_p$ and chooses a random vector $\vec{v} = (s, y_2, \ldots, y_n) \in \mathbb{Z}_p^n$, where $y_2, \ldots, y_n$ are used to share the encryption exponent s. For $i = 1$ to ℓ, it computes $\lambda_i = \vec{v} \cdot M_i$, where M_i is the vector corresponding to the i-th row of M. Then, it randomly chooses $r_1, r_2, \ldots, r_\ell \in \mathbb{Z}_p$ and computes the ciphertext as

$$\mathsf{CT} = \Bigg(C = \kappa \cdot \Bigg(\prod_{aid_k\in I_{\mathcal{A}}} \mathsf{PK}_{aid_k}\Bigg)^s, C' = g^s, C'' = g^{bs}, .$$
$$\forall 1 \le i \le \ell, \rho(i) \in S_{aid_k} :$$
$$C_i = g^{a\lambda_i} \cdot \left(\mathsf{PK}_{1,\rho(i)}\right)^{-r_i}, C'_i = g^{r_i},$$
$$D_i = g^{\frac{r_i}{\beta_{aid_k}}}, D'_i = \left(\mathsf{PK}_{2,\rho(i)}\right)^{r_i}\Bigg).$$

After that, the owner sends the data to the server in the format as described in Fig. 2.

3.5 Data Decryption

All the legal users in the system can freely query any interested encrypted data. Upon receiving the data from the server, the user runs the decryption algorithm $\mathsf{Decrypt}$ to decrypt the ciphertext by using its secret keys from different AAs. Only the attributes the user possesses satisfy the access structure defined in the ciphertext CT, the user can get the content key.

The decryption algorithm $\mathsf{Decrypt}(\mathsf{CT}, \mathsf{GPK}_{uid}, \mathsf{GSK}'_{uid}, \{\mathsf{SK}_{uid,aid_k}\}_{aid_k\in I_{\mathcal{A}}}) \to \kappa$ can be constructed as follows. It takes as inputs the ciphertext CT which contains an access policy (M, ρ), a global public key GPK_{uid} and a global secret key GSK'_{uid} of the user uid, and a set of secret keys $\{\mathsf{SK}_{uid,aid_k}\}_{aid_k\in I_{\mathcal{A}}}$ from all the involved AAs. If the user's attributes can satisfy the access structure, then the user uid proceeds as follows.

Let I be $\{I_{aid_k}\}_{aid_k\in I_{\mathcal{A}}}$, where $I_{aid_k} \subset \{1, 2, \ldots, \ell\}$ is defined as $I_{aid_k} = \{i : \rho(i) \in S_{aid_k}\}$. Let $n_{\mathcal{A}} = |I_{\mathcal{A}}|$ be the number of AAs involved in the ciphertext. Then, it chooses a set of constants $\{w_i \in \mathbb{Z}_p\}_{i\in I}$ and reconstructs the encryption exponent as $s = \sum_{i\in I} w_i\lambda_i$ if $\{\lambda_i\}$ are valid shares of the secret s according to M. The decryption algorithm first computes

$$\prod_{aid_k\in I_{\mathcal{A}}} e(C', K_{uid,aid_k}) e\left(C'', K'_{uid,aid_k}\right)^{-1}$$
$$= e(g,g)^{au_{uid}n_{\mathcal{A}}s} \cdot \prod_{aid_k\in I_{\mathcal{A}}} e(g,g)^{s\alpha_{aid}}.$$

For each $i \in I$, suppose $\rho(i) \in S_{aid_k}$, it computes

$$e(C_i, \mathsf{GPK}_{uid})\, e\left(D_i, K_{\rho(i),uid}\right) e\left(C'_i, K'^{-\mathsf{GSK}'_{uid}}_{uid,aid_k}\right) e\left(g, D'_i\right)^{-1}$$
$$= e(g,g)^{au_{uid}\lambda_i}.$$

Then, it computes

$$\prod_{aid_k\in I_{\mathcal{A}}} \prod_{i\in I_{aid_k}} \left(e(g,g)^{au_{uid}\lambda_i}\right)^{w_i n_{\mathcal{A}}} = e(g,g)^{au_{uid}n_{\mathcal{A}}s}.$$

Thus, the user can obtain $\prod_{k\in I_A} e(g,g)^{\alpha_{aid_k} s}$ and use it to decrypt the ciphertext as

$$\kappa = C/\prod_{aid_k\in I_A} e(g,g)^{\alpha_{aid_k} s}.$$

Then, the user can use the decrypted content key κ to further decrypt the encrypted data component.

3.6 Attribute Revocation

As we described before, there are two requirements of the attribute revocation: 1) The revoked user (whose attribute is revoked) cannot decrypt new ciphertexts encrypted with new public attribute keys (Backward Security); 2) the newly joined user who has sufficient attributes should also be able to decrypt the previously published ciphertexts, which are encrypted with previous public attribute keys (Forward Security). For example, in a university, some archive documents are encrypted under the policy ''CS Dept. **AND** (Professor **OR** PhD Student)'', which means that only the professors or PhD students in CS department are able to decrypt these documents. When a new professor/PhD student joins the CS department of the university, he/she should also be able to decrypt these documents. Our attribute revocation methods can achieve both forward security and backward security.

Suppose an attribute $\tilde{x}_{aid'}$ is revoked from the user uid' by the $AA_{aid'}$. The attribute $\tilde{x}_{aid'}$ is denoted as the *Revoked Attribute* and the user uid' is denoted as the *Revoked User*. We also use the term of *Non-revoked Users* to denote the set of users who possess the revoked attribute $\tilde{x}_{aid'}$ but have not been revoked. Our revocation methods contains the following three steps:

3.6.1 Update Key Generation

When an attribute $\tilde{x}_{aid'}$ is revoked from a user, the corresponding authority $AA_{aid'}$ runs the update key generation algorithm $\mathsf{UKeyGen}$ to compute the update keys. The algorithm takes as inputs the secret key $\mathsf{SK}_{aid'}$ of $AA_{aid'}$, the revoked attribute $\tilde{x}_{aid'}$ and its current version key $\mathsf{VK}_{\tilde{x}_{aid'}}$. It generates a new version key $\mathsf{VK}'_{\tilde{x}_{aid'}} = v'_{\tilde{x}_{aid'}} (v'_{\tilde{x}_{aid'}} \neq v_{\tilde{x}_{aid'}})$ for the revoked attribute $\tilde{x}_{aid'}$.

The $AA_{aid'}$ then generates a unique update key $\mathsf{UK}_{s,\tilde{x}_{aid'},uid}$ for secret key update by each non-revoked user uid as

$$\mathsf{UK}_{s,\tilde{x}_{aid'},uid} = H(\tilde{x}_{aid'})^{\beta_{aid'}(v'_{\tilde{x}_{aid'}} - v_{\tilde{x}_{aid'}})(u_{uid}+\gamma_{aid'})}$$

and generates the update key $\mathsf{UK}_{c,\tilde{x}_{aid'}}$ for ciphertext update as

$$\mathsf{UK}_{c,\tilde{x}_{aid'}} = \left(\mathsf{UK}_{1,\tilde{x}_{aid'}} = \frac{v'_{\tilde{x}_{aid'}}}{v_{\tilde{x}_{aid'}}}, \mathsf{UK}_{2,\tilde{x}_{aid'}} = \frac{v_{\tilde{x}_{aid'}} - v'_{\tilde{x}_{aid'}}}{v_{\tilde{x}_{aid'}}\gamma_{aid'}}\right).$$

The $AA_{aid'}$ sends the $\mathsf{UK}_{s,\tilde{x}_{aid'},uid}$ to non-revoked user uid and sends $\mathsf{UK}_{c,\tilde{x}_{aid'}}$ to the cloud server.

Then, the $AA_{aid'}$ updates the public attribute key of the revoked attribute $\tilde{x}_{aid'}$ as

$$\widetilde{\mathsf{PK}}_{\tilde{x}_{aid'}} = \left(\mathsf{PK}_{\tilde{x}_{aid'}}\right)^{\mathsf{UK}_{1,\tilde{x}_{aid'}}}$$

and publishes it on its public bulletin board. Then, the $AA_{aid'}$ broadcasts a message for all the owners that the public attribute key of the revoked attribute $\tilde{x}_{aid'}$ is updated.

3.6.2 Secret Key Update by Non-Revoked Users

Upon receiving the update key $\mathsf{UK}_{s,\tilde{x}_{aid'},uid}$, the user uid then update his/her secret key by running the new secret key update algorithm $\mathsf{SKUpdate}$ as

$$\begin{aligned}\widetilde{\mathsf{SK}}_{uid,aid'} = \Big(&\tilde{K}_{uid,aid'} = K_{uid,aid'}, \tilde{K}'_{uid,aid'} = K'_{uid,aid'}, \\ &\tilde{K}_{\tilde{x}_{aid'},uid} = K_{\tilde{x}_{aid'},uid} \cdot \mathsf{UK}_{s,\tilde{x}_{aid'},uid}, \\ &\forall x_{aid'} \in S_{uid,aid'} \setminus \{\tilde{x}_{aid'}\} : \tilde{K}_{x_{aid'},uid} = K_{x_{aid'},uid}\Big).\end{aligned}$$

Note that only the component associated with the revoked attribute $\tilde{x}_{aid'}$ in the secret key needs to be updated, while other components are kept unchanged.

3.6.3 Ciphertext Update by Cloud Server

To ensure that the newly joined user who has sufficient attributes can still decrypt those previous data which are published before it joined the system (Forward Security), all the ciphertexts associated with the revoked attribute are required to be updated to the latest version. Intuitively, the ciphertext update should be done by data owners, which will incur a heavy overhead on the data owner. To improve the efficiency, we move the workload of ciphertext update from data owners to the cloud server, such that it can eliminate the huge communication overhead between data owners and cloud server, and the heavy computation cost on data owners. The ciphertext update is conducted by using proxy re-encryption method, which means that the server does not need to decrypt the ciphertext before updating.

Upon receiving the update key $\mathsf{UK}_{c,\tilde{x}_{aid'}}$ from the authority. The cloud server runs the ciphertext update algorithm $\mathsf{CTUpdate}$ to update the ciphertext associated with the revoked attribute $\tilde{x}_{aid'}$. It takes as inputs the ciphertexts associated with the revoked attribute $\tilde{x}_{aid'}$ and the update key $\mathsf{UK}_{c,\tilde{x}_{aid'}}$. It updates the ciphertext that are associated with the revoked attribute $\tilde{x}_{aid'}$ as

$$\begin{aligned}\widetilde{CT} = \big(&\tilde{C} = C, \tilde{C}' = C', \tilde{C}'' = C'', \\ &\forall 1 \le i \le \ell : \tilde{C}'_i = C'_i, \tilde{D}_i = D_i, \\ &\text{if } \rho(i) = \tilde{x}_{aid'} : \tilde{C}_i = C_i \cdot \left(D'_i\right)^{\mathsf{UK}_{2,\tilde{x}_{aid'}}}, \\ &\qquad \tilde{D}'_i = \left(D'_i\right)^{UK_{1,\tilde{x}_{aid'}}}, \\ &\text{if } \rho(i) \neq \tilde{x}_{aid'} : \tilde{C}_i = C_i, \tilde{D}'_i = D'_i\big). \qquad (3.1)\end{aligned}$$

From the above equation Eq. (3.1), it is easy to find that our scheme only requires to update those components associated with the revoked attribute of the ciphertext, while the other components which are not related to the revoked attribute are not changed. In this way, our scheme can greatly improve the efficiency of attribute revocation.

The ciphertext update not only can guarantee the backward security of the attribute revocation, but also can reduce the storage overhead on users (i.e., all the users need to hold only the latest secret key, rather than to keep records on all the previous secret keys). The cloud server in our system is required to be semi-trusted. Even if the cloud server is not semi-trusted in some scenarios, the server will not update the ciphertexts correctly. In this situation, the forward security cannot be guaranteed, but our system can still achieve the backward security.

TABLE 2
Storage Overhead on Each Entity

Entity	[13]	[14]	Our
$AA_{aid}(\|p\|)$	$2n_{aid}$	$n_{aid}+2n_{\mathcal{O}}+1$	$n_{aid}+2n_{\mathcal{U}}+3$
Owner $(\|p\|)$	n_c+2n_a	$n_c+n_{\mathcal{A}}+n_a+2$	$3n_{\mathcal{A}}+n_a+3$
User $(\|p\|)$	$n_{c,x}+n_{a,uid}$	$n_{\mathcal{O}}(n_{\mathcal{A}}+n_{a,uid})$	$2n_{\mathcal{A}}+n_{a,uid}$
Server $(\|p\|)$	$3\ell+1$	$\ell+2$	$4\ell+3$

n_c: the total number of ciphertexts stored on the cloud server; $n_{c,x}$: the number of ciphertexts contain the revoked attribute x.

4 Security Analysis

We prove that our data access control is secure under the security model we defined, which can be summarized as in the following theorems.

Theorem 1. *When the decisional q-parallel BDHE assumption holds, no polynomial time adversary can selectively break our system with a challenge matrix of size $l^* \times n^*$, where $n^* \leq q$.*

Proof. The proof is given in the supplemental file available online. □

Theorem 2. *Our scheme can achieve both Forward Security and Backward Security.*

Proof. Actually, the Forward Security and Backward Security are two basic requirements of attribute revocation. Now we prove that our scheme can achieve this two requirements as follows.

Backward Security: During the secret key update phase, the corresponding AA generates an update key for each non-revoked user. Because the update key is associated with the user's global identity uid, the revoked user cannot use update keys of other non-revoked users to update its own secret key, even if it can compromise some non-revoked users. Moreover, suppose the revoked user can corrupt some other AAs (not the AA corresponding to the revoked attributes), the item $H(x_{aid})^{v_{x_{aid}}\beta_{aid}\gamma_{aid}}$ in the secret key can prevent users from updating their secret keys with update keys of other users, since γ_{aid} is only known by the AA_{aid} and kept secret to all the users. This guarantees the backward security.

Forward Security: After each attribute revocation operation, the version of the revoked attribute will be updated. When new users join the system, their secret keys are associated with attributes with the latest version. However, previously published ciphertexts are encrypted under attributes with old version. The ciphertext update algorithm in our protocol can update previously published ciphertexts into the latest attribute version, such that newly joined users can still decrypt previously published ciphertexts, if their attributes can satisfy access policies associated with ciphertexts. This guarantees the forward security. □

Theorem 3. *Our access control scheme can resist the collusion attack, even when some AAs are corrupted by the adversary.*

Proof. Users may collude and combine their attributes to decrypt the ciphertext, although they are not able to decrypt the ciphertext alone. Due to the random number t and the aid in the secret key, each component associated with the attribute in the secret key is distinguishable from each other, although some AAs may issue the same attributes. Moreover, the secret key is also associated with the user's globally unique identity uid. Thus, users cannot collude together to gain illegal access by combining their attributes together.

However, when some AAs is corrupted by the adversary, the collusion resistance becomes more complicated. Specifically, the adversary may launch *Attribute Forge Attack*, defined as follows. Suppose a user uid_0 possesses an attribute "x_{aid_0}" from AA_{aid_0}, while the adversary μ does not hold the attribute "x_{aid_0}" from AA_{aid_0}. The adversary μ attempts to forge ("clone") the attribute "x_{aid_0}" from the user uid_0's secret key by colluding with some other AAs.

In our scheme, the item $g^{u'_{uid}t_{uid,aid}\beta_{aid}}$ in the secret key construction helps to resist this attack. When the adversary corrupts any AAs, he/she can get all the global secret key GSK_{uid} for all the users in the system (because each AA has full knowledge on one of the user's global secret keys GSK_{uid}). Suppose all the $K_{x_{aid},uid}$ in the secret key is constructed without this item. The adversary can successfully forge the attribute "x_{aid_0}" as

$$K_{x_{aid_0},\mu} = (K_{x_{aid_0},uid_0})^{\mathsf{GSK}_{\mu}/\mathsf{GSK}_{uid_0}}.$$

By adding the item $g^{u'_{uid}t_{uid,aid}\beta_{aid}}$, such attribute forge attack will be eliminated. □

Privacy-Preserving Guarantee: Although the CA holds the global master key GMK, it does not have any secret key issued from the AA. Without the knowledge of $g^{\alpha_{aid}}$, the CA cannot decrypt any ciphertexts in the system. Our scheme can also prevent the server from getting the content of the cloud data by using the proxy-encryption method.

5 Performance Analysis

In this section, we analyze the performance of our scheme by comparing with the Ruj's DACC scheme [13] and our previous scheme in the conference version [14], in terms of storage overhead, communication cost and computation efficiency.

We conduct the comparison under the same security level. Let $|p|$ be the element size in the $\mathbb{G}, \mathbb{G}_T, \mathbb{Z}_p$. Suppose there are $n_{\mathcal{A}}$ authorities in the system and each attribute authority AA_{aid} manages n_{aid} attributes. Let $n_{\mathcal{U}}$ and $n_{\mathcal{O}}$ be the total number of users and owners in the system respectively. For a user uid, let $n_{uid,aid_k} = |S_{uid,aid_k}|$ denote the number of attributes that the user uid obtained from AA_{aid_k}. Let ℓ be the total number of attributes in the ciphertext.

5.1 Storage Overhead

The storage overhead is one of the most significant issues of the access control scheme in cloud storage systems. Let $n_a = \sum_{k=1}^{n_{\mathcal{A}}} n_{aid_k}$ denote the total number attributes in the system and $n_{a,uid} = \sum_{k=1}^{n_{\mathcal{A}}} n_{uid,aid_k}$ denote the total number of attributes the user uid holds from all the AAs in the system. We compare the storage overhead on each entity in the system, as shown in Table 2.

TABLE 3
Communication Cost for Attribute Revocation

Operation	[13]	[14]	Our
Key Update	None	$n_{non,x}\|p\|$	$n_{non,x}\|p\|$
CT Update	$(n_{c,x} \cdot n_{non,x}+1)\|p\|$	$n_{c,aid}\|p\|$	$2\|p\|$

$n_{non,x}$: num of non-revoked users hold x;
$n_{c,x}$: num of ciphertexts contains x;
$n_{c,aid}$: num of attributes from the AA_{aid} in all ciphertexts.

1) *Storage Overhead on Each: AA* Each AA needs store the information of all the attributes in its domain. Besides, in [14], each AA_{aid} also needs to store the secret keys from all the owners, where the storage overhead on each AA is also linear to the total number of owners $n_{\mathcal{O}}$ in the system. In our scheme, besides the storage of attributes, each AA_{aid} also needs to store a public key and a secret key for each user in the system. Thus, the storage overhead on each AA in our scheme is also linear to the number of users $n_{\mathcal{U}}$ in the system.

2) *Storage Overhead on Each Owner:* The public parameters contribute the main storage overhead on the owner. Besides the public parameters, in [13], owners are required to re-encrypt the ciphertexts and in [14] owners are required to generate the update information during the revocation, where the owner should also hold the encryption secret for every ciphertext in the system. This incurs a heavy storage overhead on the owner, especially when the number of ciphertext is large in cloud storage systems.

3) *Storage Overhead on Each User:* The storage overhead on each user in our scheme comes from the secret keys issued by all the AAs. However, in [13], the storage overhead on each user consists of both the secret keys issued by all the AAs and the ciphertext components that associated with the revoked attribute x, because when the ciphertext is re-encrypted, some of its components related to the revoked attributes should be sent to each non-revoked user who holds the revoked attributes. In [14], the user needs to hold multiple secret keys for multiple data owners, which means that the storage overhead on each user is also linear to the number of owners $n_{\mathcal{O}}$ in the system.

4) *Storage Overhead on Server:* The ciphertexts contribute the main storage overhead on the server (here we do not consider the encrypted data which are encrypted by the symmetric content keys).

5.2 Communication Cost

The communication cost of the normal access control is almost the same. Here, we only compare the communication cost of attribute revocation, as shown in Table 3. The communication cost of attribute revocation in [13] is linear to the number of ciphertexts which contain the revoked attribute. In [14], the communication overhead is linear to the total number of attributes $n_{c,aid}$ belongs to the AA_{aid} in all the ciphertexts. It is not difficult to find that our scheme incurs much less communication cost during the attribute revocation.

5.3 Computation Efficiency

We implement our scheme and DACC scheme [13] on a Linux system with an Intel Core 2 Duo CPU at 3.16GHz and 4.00 GB RAM. The code uses the Pairing-Based Cryptography (PBC) library version 0.5.12 to implement the access control schemes. We use a symmetric elliptic curve α-curve, where the base field size is 512-bit and the embedding degree is 2. The α-curve has a 160-bit group order, which means p is a 160-bit length prime. All the simulation results are the mean of 20 trials.

We compare the computation efficiency of both encryption and decryption in two criteria: the number of authorities and the number of attributes per authority. Fig. 3a describes the comparison of encryption time versus the number of authorities, where the involved number of attributes per authority is set to be 10. Fig. 3c gives the encryption time comparison versus the number of attributes per authority, where the involved number of authority is set to be 10. It is easy to find that our scheme incurs less encryption time than DACC scheme in [13].

Fig. 3b shows the comparison of decryption time versus the number of authorities, where the number of attributes the user holds from each authority is set to be 10. Suppose the user has the same number of attributes from each authority, Fig. 3d describes the decryption time comparison versus the number of attributes the user holds from each authority. In Fig. 3d, the number of authority for the user is fixed to be 10. It is not difficult to see that our scheme incurs less decryption on the user than DACC scheme in [13].

Fig. 3e describes the time of cihertext update/re-encryption versus the number of revoked attributes, and our scheme is more efficient than [13]. The ciphertext update/re-encryption contributes the main computation overhead of the attribute revocation. In our conference version [14], when an attribute is revoked from its corresponding authority $AA_{aid'}$, all the ciphertexts which are associated with *any* attributes from $AA_{aid'}$ should be updated. In this paper, however, the attribute revocation method only requires the update of ciphertexts which are associated with the revoked attribute.

6 Related Work

Ciphertext-Policy Attribute-Based Encryption (CP-ABE) [2]-[3] is a promising technique that is designed for access control of

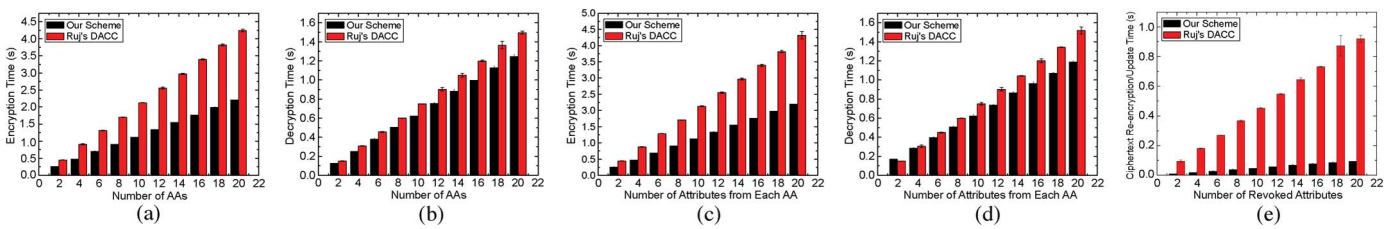


Fig. 3. Comparison of Computation Time. (a) Encryption. (b) Decryption. (c) Encryption. (d) Decryption. (e) Re-encryption.

encrypted data. There are two types of CP-ABE systems: single-authority CP-ABE [2], [3], [4], [5] where all attributes are managed by a single authority, and multi-authority CP-ABE [6], [7], [8] where attributes are from different domains and managed by different authorities. Multi-authority CP-ABE is more appropriate for the access control of cloud storage systems, as users may hold attributes issued by multiple authorities and the data owners may share the data using access policy defined over attributes from different authorities. However, due to the attribute revocation problem, these multi-authority CP-ABE schemes cannot be directly applied to data access control for such multi-authority cloud storage systems.

To achieve revocation on attribute level, some re-encryption-based attribute revocation schemes [9], [11] are proposed by relying on a trusted server. We know that the cloud server cannot be fully trusted by data owners, thus traditional attribute revocation methods are no longer suitable for cloud storage systems.

Ruj, Nayak and Ivan proposed a DACC scheme [13], where an attribute revocation method is presented for the Lewko and Waters' decentralized ABE scheme [8]. Their attribute revocation method does not require a fully trusted server. But, it incurs a heavy communication cost since it requires the data owner to transmit a new ciphertext component to every non-revoked user.

7 Conclusion

In this paper, we proposed a revocable multi-authority CP-ABE scheme that can support efficient attribute revocation. Then, we constructed an effective data access control scheme for multi-authority cloud storage systems. We also proved that our scheme was provable secure in the random oracle model. The revocable multi-authority CP-ABE is a promising technique, which can be applied in any remote storage systems and online social networks etc.

Acknowledgment

This work was supported by the Research Grants Council of Hong Kong under Project CityU 114112.

References

[1] P. Mell and T. Grance, ''The NIST Definition of Cloud Computing,'' National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep., 2009.

[2] J. Bethencourt, A. Sahai, and B. Waters, ''Ciphertext-Policy Attribute-Based Encryption,'' in *Proc. IEEE Symp. Security and privacy (S&P'07)*, 2007, pp. 321-334.

[3] B. Waters, ''Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization,'' in *Proc. 4th Int'l Conf. Practice and Theory in Public Key Cryptography (PKC'11)*, 2011, pp. 53-70.

[4] V. Goyal, A. Jain, O. Pandey, and A. Sahai, ''Bounded Ciphertext Policy Attribute Based Encryption,'' in *Proc. 35th Int'l Colloquium on Automata, Languages, and Programming (ICALP'08)*, 2008, pp. 579-591.

[5] A.B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, ''Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption,'' in *Proc. Advances in Cryptology-EUROCRYPT'10*, 2010, pp. 62-91.

[6] M. Chase, ''Multi-Authority Attribute Based Encryption,'' in *Proc. 4th Theory of Cryptography Conf. Theory of Cryptography (TCC'07)*, 2007, pp. 515-534.

[7] M. Chase and S.S.M. Chow, ''Improving Privacy and Security in Multi-Authority Attribute-Based Encryption,'' in *Proc. 16th ACM Conf. Computer and Comm. Security (CCS'09)*, 2009, pp. 121-130.

[8] A.B. Lewko and B. Waters, ''Decentralizing Attribute-Based Encryption,'' in *Proc. Advances in Cryptology-EUROCRYPT'11*, 2011, pp. 568-588.

[9] S. Yu, C. Wang, K. Ren, and W. Lou, ''Attribute Based Data Sharing with Attribute Revocation,'' in *Proc. 5th ACM Symp. Information, Computer and Comm. Security (ASIACCS'10)*, 2010, pp. 261-270.

[10] M. Li, S. Yu, Y. Zheng, K. Ren, and W. Lou, ''Scalable and Secure Sharing of Personal Health Records in Cloud Computing Using Attribute-Based Encryption,'' *IEEE Trans. Parallel Distributed Systems*, vol. 24, no. 1, pp. 131-143, Jan. 2013.

[11] J. Hur and D.K. Noh, ''Attribute-Based Access Control with Efficient Revocation in Data Outsourcing Systems,'' *IEEE Trans. Parallel Distributed Systems*, vol. 22, no. 7, pp. 1214-1221, July 2011.

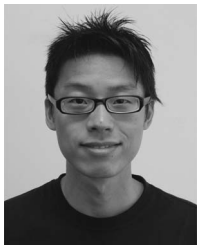
[12] S. Jahid, P. Mittal, and N. Borisov, ''Easier: Encryption-Based Access Control in Social Networks with Efficient Revocation,'' in *Proc. 6th ACM Symp. Information, Computer and Comm. Security (ASIACCS'11)*, 2011, pp. 411-415.

[13] S. Ruj, A. Nayak, and I. Stojmenovic, ''DACC: Distributed Access Control in Clouds,'' in *Proc. 10th IEEE Int'l Conf. TrustCom*, 2011, pp. 91-98.

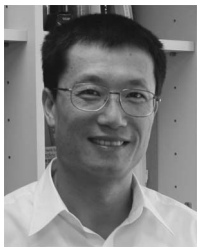
[14] K. Yang and X. Jia, ''Attribute-Based Access Control for Multi-Authority Systems in Cloud Storage,'' in *Proc. 32th IEEE Int'l Conf. Distributed Computing Systems (ICDCS'12)*, 2012, pp. 1-10.

[15] D. Boneh and M.K. Franklin, ''Identity-Based Encryption from the Weil Pairing,'' in *Proc. 21st Ann. Int'l Cryptology Conf.: Advances in Cryptology - CRYPTO'01*, 2001, pp. 213-229.

[16] A.B. Lewko and B. Waters, ''New Proof Methods for Attribute-Based Encryption: Achieving Full Security through Selective Techniques,'' in *Proc. 32st Ann. Int'l Cryptology Conf.: Advances in Cryptology - CRYPTO'12*, 2012, pp. 180-198.



Kan Yang received the BEng degree from University of Science and Technology of China, in 2008 and the PhD degree from City University of Hong Kong, Hong Kong, in August 2013. He was a visiting scholar in State University of New York at Buffalo, in 2012. His research interests include cloud security, big data security, cloud data mining, cryptography, social networks, wireless communication and networks, and distributed systems. He is a Student Member of the IEEE.



Xiaohua Jia received the BSc and MEng degrees from University of Science and Technology of China, in 1984 and 1987, respectively, and the DSc degree in information science from University of Tokyo, Japan, in 1991. He is currently Chair Professor at the Department of Computer Science at City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, wireless sensor networks and mobile wireless networks. Prof. Jia is an editor of IEEE Transactions on Parallel and Distributed Systems (2006-2009), Wireless Networks, Journal of World Wide Web, Journal of Combinatorial Optimization, etc. He is the General Chair of ACM MobiHoc 2008, TPC Co-Chair of IEEE MASS 2009, Area-Chair of IEEE INFOCOM 2010, TPC Co-Chair of IEEE GlobeCom 2010, Ad Hoc and Sensor Networking Symp, and Panel Co-Chair of IEEE INFOCOM 2011. He is a Fellow of the IEEE.

www.redpel.com
+917620593389