DAC-MACS: Effective Data Access Control for Multi-Authority Cloud Storage Systems

Kan Yang*, Xiaohua Jia†, Kui Ren‡
*Department of Computer Science, City University of Hong Kong, Email: kanyang3@student.cityu.edu.hk
†Department of Computer Science, City University of Hong Kong, Email: jia@cs.cityu.edu.hk
‡Department of Electrical and Computer Engineering, Illinois Institute of Technology, Email: kren@ece.iit.edu

***Abstract*—Data access control is an effective way to ensure the data security in the cloud. However, due to data outsourcing and untrusted cloud servers, the data access control becomes a challenging issue in cloud storage systems. Existing access control schemes are no longer applicable to cloud storage systems, because they either produce multiple encrypted copies of the same data or require a fully trusted cloud server. Ciphertext-Policy Attribute-based Encryption (CP-ABE) is a promising technique for access control of encrypted data. It requires a trusted authority manages all the attributes and distributes keys in the system. In cloud storage systems, there are multiple authorities co-exist and each authority is able to issue attributes independently. However, existing CP-ABE schemes cannot be directly applied to the access control for multi-authority cloud storage systems, due to the inefficiency of decryption and revocation. In this paper, we propose DAC-MACS (Data Access Control for Multi-Authority Cloud Storage), an effective and secure data access control scheme with efficient decryption and revocation. Specifically, we construct a new multi-authority CP-ABE scheme with efficient decryption and also design an efficient attribute revocation method that can achieve both forward security and backward security. The analysis and the simulation results show that our DAC-MACS is highly efficient and provably secure under the security model.**

***Index Terms*—Access Control, CP-ABE, Decryption Outsourcing, Attribute Revocation, Multi-authority Cloud.**

I. Introduction

Cloud storage is an important service of cloud computing [1]. It allows data owners to host their data in the cloud that provides "24/7/365" data access to the users (data consumers). Data access control is an effective way to ensure the data security in the cloud. However, cloud storage service separates the roles of the data owner from the data service provider, and the data owner does not interact with the user directly for providing data access service, which makes the data access control a challenging issue in cloud storage systems. Because the cloud server cannot be fully trusted by data owners, traditional server-based access control methods are no longer applicable to cloud storage systems. To prevent the untrusted servers from accessing sensitive data, traditional methods usually encrypt the data and only users holding valid keys can access the data. These methods require complicated key management schemes and the data owners have to stay online all the time to deliver the keys to new user in the system. Moreover, these methods incur high storage overhead on the server, because the server should store multiple encrypted copies of the same data for users with different keys.

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 and does not require the data owner to distribute keys. 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 under the policies. Each user will be issued a secret key according to its attributes. A user can decrypt the ciphertext only when its attributes satisfy the access policies.

Extensive research has been done for single authority systems [2]–[6]. However, in cloud storage systems, a user may hold attributes issued by multiple authorities and the owner may share data to the users administrated to different authorities. For instance, a data owner may want to share medical data only with a user who has the attribute of "Doctor" issued by a hospital and the attribute "Medical Researcher" issued by a medical research center. Although some multi-authority CP-ABE schemes [7]–[10] have been proposed for data encryption, they cannot be directly applied to do data access control for multi-authority cloud storage systems, because they either require a global central attribute authority to manage all the attributes across different organizations or lack of efficiency. The aim of this paper is to study the data access control issue in multi-authority cloud storage systems.

One critical requirement in the design of access control schemes is the *efficiency in computation*. There are two operations in access control that require efficient computation, namely decryption and revocation. The users may use their smart phones to access the data in nowadays cloud storage systems, but the computation ability of smart phones is not as strong as the PCs. Thus, the decryption on each user should be as efficient as possible in the design of data access control schemes. When a user is degraded or leaving the system, some attributes should be revoked from this user. There are two requirements of the efficient attribute revocation: 1) The revoked user (whose attribute is revoked) cannot decrypt the new ciphertext that is encrypted with new public key (Forward Security); 2) The newly joined user can also decrypt the previous published ciphertexts that are encrypted with previous public key if it has sufficient attributes (Backward Security).

In this paper, we first construct a new multi-authority CP-ABE scheme with efficient decryption and design an efficient attribute revocation method for it. Then, we apply them to design an effective access control scheme for multi-authority systems. The main contributions of this work can be summarized as follows.

1) We propose DAC-MACS (Data Access Control for Multi-Authority Cloud Storage), an effective and secure data access control scheme for multi-authority cloud storage systems, which is provably secure in the random oracle model and has better performance than existing schemes.
2) We construct a new multi-authority CP-ABE scheme with efficient decryption. Specifically, we outsource the main computation of the decryption by using a token-based decryption method.
3) We also design an efficient immediate attribute revocation method for multi-authority CP-ABE scheme that achieves both forward security and backward security. It is efficient in the sense that it incurs less communication cost and computation cost of the revocation.

The remaining of this paper is organized as follows. We first give the definition of the system model and security model in Section II. Then, we propose a new multi-authority CP-ABE scheme with efficient decryption and revocation and apply it to design DAC-MACS in Section III. In Section IV, we analyze DAC-MACS in terms of both the security and the performance. Section V gives the related work on data access control and the attribute revocation in ABE systems. Finally, the conclusion is given in Section VI and the detailed security proof is described in the Appendix.

II. System Model and Security Model

A. Definition of System Model and Framework

We consider a cloud storage system with multiple authorities, as shown in Fig.1. The system model consists of five types of entities: a global certificate authority (CA), the attribute authorities (*AA*s), the cloud server (server), the data owners (owners) and the 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 *AA*s in the system. The CA is responsible for the distribution of global secret key and global public key for each legal user in the system. 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 issuing, revoking and updating user's attributes according to their role or identity in its domain. In DAC-MACS, 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.

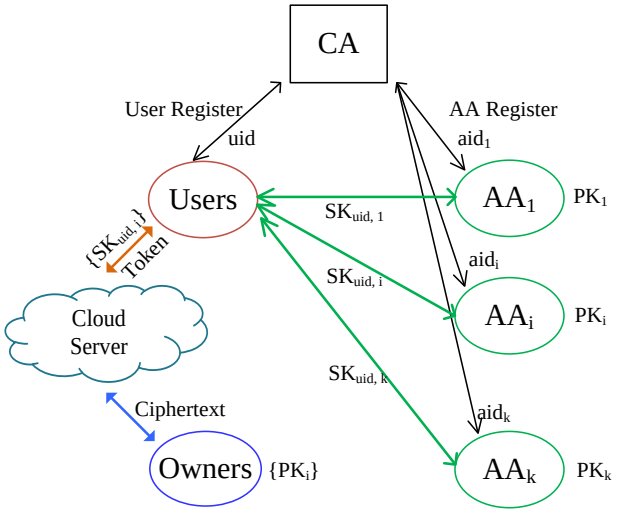


Fig. 1. System Model of Multi-authority Access Control in Cloud Storage

Each *AA* is responsible for generating a public attribute key for each attribute it manages and a secret key for each user associates with their attributes.

The cloud server stores the owners' data and provides data access service to users. It generates the decryption token of a ciphertext for the user by using the secret keys of the user issued by the *AA*s. The server also does the ciphertext update when an attribute revocation happens.

The data owners define the access policies and encrypt the data under the policies before hosting them in the cloud. They do not rely on the server to do the data access control. Instead, the ciphertext can be accessed by all the legal users in the system. 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 can decrypt the ciphertext.

Each user is assigned with a global user identity from the CA. Each user can freely get the ciphertexts from the server. To decrypt a ciphertext, each user may submit their secret keys issued by some *AA*s together with its global public key to the server and ask it to generate an decryption token for some ciphertext. Upon receiving the decryption token, the user can decrypt the ciphertext by using its global secret key. Only when the user's attributes satisfy the access policy defined in the ciphertext, the server can generate the correct decryption token. The secret keys and the global user's public key can be stored on the server; subsequently, the user does not need to submit any secret keys if no secret keys are updated for the further decryption token generation.

Then, we give the definition of the framework of our data access control in multi-authority cloud storage systems as follows.

Definition 1 (DAC-MACS)**.** *DAC-MACS is a collection of algorithms that combines a set of CP-ABE algorithms:* CASetup, AASetup, UserRegister, KeyGen, Encrypt, TKGen, Decrypt *and a set of attribute revocation algorithms:*

UKeyGen, KeyUpdate *and* CiphertextUpdate.

CASetup$(\lambda) \rightarrow (\mathsf{MSK}, \mathsf{SP}, sk_{CA}, pk_{CA})$. *The CA setup algorithm takes no input other than the implicit security parameter* λ. *It outputs the master key* MSK, *the system parameter* SP *and the pair of CA's secret key and public key* (sk_{CA}, pk_{CA}).

AASetup$(aid) \rightarrow (\mathsf{SK}_{aid}, \{\mathsf{VK}_{x_{aid}}, \mathsf{PK}_{x_{aid}}\})$. *The authority generation algorithm takes the authority id aid as input. It outputs the authority secret key* SK_{aid}, *the set of version keys and public attribute keys* $\{\mathsf{VK}_{x_{aid}}, \mathsf{PK}_{x_{aid}}\}$ *for all attributes x issued by the* AA_{aid}.

UserRegister$(sk_{CA}) \rightarrow (uid, \mathsf{GPK}_{uid}, \mathsf{GSK}_{uid}, Sig_{sk_{CA}}(uid))$. *The user registration algorithm takes the input as the CA's secret key* sk_{CA}. *For each legal user in the system, it outputs a global user id uid, the pair of global public key and secret key* $(\mathsf{GPK}_{uid}, \mathsf{GSK}_{uid})$ *and a user certification* $Sig_{sk_{CA}}(uid)$.

KeyGen$(S_{uid,aid}, \mathsf{SK}_{aid}, \{\mathsf{PK}_{x_{aid}}\}, \mathsf{SP}, Sig_{sk_{CA}}(uid)) \rightarrow (\mathsf{PK}_{aid}, \mathsf{SK}_{uid,aid})$. *The key generation algorithm takes as inputs a set of attributes* $S_{uid,aid}$ *that describes the secret key, the authority secret key* SK_{aid}, *the set of public attribute keys* $\{\mathsf{PK}_{x_{aid}}\}$, *the system parameter and the certification of the user with uid. It outputs the public key* PK_{aid} *and a secret key* $\mathsf{SK}_{uid,aid}$ *for the user with uid.*

Encrypt$(\{\mathsf{PK}_k\}_{k\in I_A}, \{\mathsf{PK}_{x_k}\}_{k\in I_A}, m, \mathbb{A}) \rightarrow CT$. *The encryption algorithm takes as inputs a set of public keys* $\{\mathsf{PK}_k\}_{k\in I_A}$ *from the involved authority set* I_A, *a set of public attribute keys* $\{\mathsf{PK}_{x_k}\}_{k\in I_A}$, *a message m and an access structure* $\mathbb{A}$ *over all the selected attributes from the involved AAs. The algorithm encrypts m according to the access structure and outputs a ciphertext* CT. *We will assume that the ciphertext implicitly contains the access structure* $\mathbb{A}$.

TKGen$(\mathsf{CT}, \mathsf{GPK}_{uid}, \{\mathsf{SK}_{uid,k}\}_{k\in I_A}) \rightarrow \mathsf{TK}$. *The decryption token generation algorithm takes as input the ciphertext* CT *which contains an access structure* $\mathbb{A}$, *user's global public key* GPK_{uid} *and a set of user's secret keys* $\{SK_{uid,k}\}_{k\in I_A}$. *If the set of attributes S satisfies the access structure* $\mathbb{A}$, *the algorithm can successfully compute the decryption token* TK *of the ciphertext.*

Decrypt$(\mathsf{CT}, \mathsf{TK}, \mathsf{GSK}_{uid}) \rightarrow m$. *The decryption algorithm takes as inputs the ciphertext* CT, *the decryption token* TK *and the user's global secret key* GSK_{uid}. *It outputs the message m.*

UKeyGen$(\mathsf{SK}_{aid}, \{u_j\}_{j\in S_U}, \mathsf{VK}_{\tilde{x}_{aid}}) \rightarrow (\mathsf{UUK}_{j,\tilde{x}_{aid}}, \mathsf{CUK}_{\tilde{x}_{aid}})$. *The update key generation algorithm takes as inputs the authority secret key* SK_{aid}, *a set of user's secret* $\{u_j\}$ *and the previous version key of the revoked attribute* $\mathsf{VK}_{\tilde{x}_{aid}}$. *It outputs both the User Update Key* $\mathsf{UUK}_{j,\tilde{x}_{aid}}(j \in S_U, j \neq \mu, \tilde{x}_k \in S_{j,aid})$ *and the Ciphertext Update Key* $\mathsf{CUK}_{\tilde{x}_k}$.

KeyUpdate$(\mathsf{SK}_{j,aid}, \mathsf{UUK}_{j,\tilde{x}_{aid}}) \rightarrow \mathsf{SK}'$. *The user's secret key update algorithm takes as inputs the non-revoked user's current secret key* $\mathsf{SK}_{j,aid}$ *and the user update key* $\mathsf{UUK}_{j,\tilde{x}_{aid}}$. *It outputs a new secret key* SK′ *to this non-revoked user.*

CiphertextUpdate$(\mathsf{CT}, \mathsf{CUK}_{\tilde{x}_{aid}}) \rightarrow \mathsf{CT}'$. *The ciphertext update algorithm takes as inputs the current ciphertext CT and the ciphertext update key* $\mathsf{CUK}_{\tilde{x}_{aid}}$. *It outputs a new ciphertext* CT′.

B. Definition of Security Model

In cloud storage systems, we consider the case that the server may send the owners' data to the users who do not have access permission. The server is also curious about the content of the encrypted data. But we assume that the server will execute correctly the task assigned by the attribute authority. The users, however, are dishonest and may collude to obtain unauthorized access to data. The *AA* can be corrupted or compromised by the attackers.

We now describe the security model for multi-authority CP-ABE systems by the following game between a challenger and an adversary. Similar to the identity-based encryption schemes [11]–[13], the security model allows the adversary to query for any secret keys that cannot be used to decrypt the challenge ciphertext. We assume that the adversaries can corrupt authorities only statically similar to [10], but key queries are made adaptively. Let S_A denote the set of all the authorities. The security game is defined as follows.

Setup. The CA runs the CASetup and each *AA* runs the AASetup. The adversary specifies a set $S'_A \subset S_A$ of corrupted authorities. The challenger generates the pairs of public key and the secret key by running the key generation algorithm. For uncorrupted authorities in $S_A - S'_A$, the challenger sends only the public keys to the adversary. For corrupted authorities in S'_A, the challenger sends both the public keys and secret keys to the adversary.

Secret Key Query Phase 1. The adversary makes secret key queries by submitting pairs $(Sig_{sk_{CA}}(uid), S_{aid,1})$, $\cdots$, $(Sig_{sk_{CA}}(uid), S_{aid,q_1})$ to the challenger, where $S_{aid,i}$ is a set of attributes belonging to an uncorrupted AA_{aid}, and $Sig_{sk_{CA}}(uid)$ is a user certificate. The challenger gives the corresponding secret keys $\{\mathsf{SK}_{uid,aid,i}\}_{i\in[1,q_1]}$ 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. We let V denote the subset of rows of M^* labeled by attributes controlled by corrupted *AA*s. For each *uid*, we let V_{uid} denote the subset of rows of M^* labeled by attributes 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 *AA*s. The challenger then flips a random coin b, and encrypts m_b under the access structure (M^*, ρ^*). Then, the ciphertext CT^* is given to the adversary.

Secret Key Query Phase 2. The adversary may query more secret keys, as long as they do not violate the constraints on the challenge access structure (M^*, ρ^*).

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

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

Definition 2. *A 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.*

III. DAC-MACS: Data Access Control for Multi-Authority Cloud Storage

In this section, we first give an overview of the challenges and techniques of designing access control schemes for multi-authority cloud storage systems. Then, we propose the detailed construction of our access control scheme DAC-MACS.

A. Overview of Our Solutions

Although the existing multi-authority CP-ABE scheme [10] proposed by Lewko and Waters has high policy expressiveness and has been extended to support attribute revocation in [14], it still cannot be applied to access control for multi-authority cloud storage systems due to the inefficiency of decryption and revocation. In order to design an efficient access control scheme for multi-authority systems, we first construct a new multi-authority CP-ABE scheme with efficient decryption and then propose an efficient attribute revocation for it.

One challenging issue in the design of a multi-authority CP-ABE scheme is how to tie the secret keys together and prevent the collusion attack. Without a central authority, it is hard to tie together different components of a user's secret key and use the key randomization method to prevent the collusion attack. In our method, we separate the authority into a global certificate authority (CA) and multiple attribute authorities (*AA*s). The CA sets up the system and accepts the registration of all the users and *AA*s in the system. However, the CA is *not* involved in any attribute management and the creation of secret keys that are associated with attributes. The CA assigns a global user identity *uid* to each user and a global authority identity *aid* to each attribute authority in the system. Since the *uid* is global unique in the system, secret keys issued by different *AA*s for the same *uid* can be tied together for decryption. Also, since each *AA* is associated with an *aid*, every attribute is distinguishable even though some *AA*s may issue the same attribute. Thus, the collusion attack can be resisted by using the *aid* and *uid*.

To achieve efficient decryption on the user, we propose a token-based decryption outsourcing method. We apply the decryption outsourcing idea from [14] and extend it to multiple authority systems by letting the CA generate a pair of global secret key and global public key for each legal user in the system. During the decryption, the user submits its secret keys issued by *AA*s to the server and asks the server to compute a decryption token for the ciphertext. The user can then decrypt the ciphertext by using the decryption token together with its global secret key.

To solve the attribute revocation problem, we assign a version number for each attribute, such that 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 any *AA*, the *AA* generates a new version number and generate several user update keys and a ciphertext update key. With the user update key, each non-revoked user who holds the revoked attributes can update their secret key. Because the update keys are distinguishable for different users, the revoked user cannot update its secret key by using other users' update keys (Forward Security). By using the ciphertext update key, the component associated with the revoked in the ciphertext can be updated to the current version. To improve the efficiency, we delegate the ciphertext update workload to the server by using the proxy re-encryption method, such that the new joined user is also able to decrypt the previous published data which are published before it joins the system (Backward Security). Moreover, all the users need to hold the latest secret key, rather than to keep records on all the previous secret keys.

To realize the fine-grained access control, the 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 methods. Then, the owner applies our proposed multi-authority CP-ABE scheme to encrypt each content key, such that only the user whose attributes satisfy the access structure in the ciphertext can decrypt the content keys. Users with different attributes can decrypt different number of content keys and thus obtain different granularities of information from the same data.

B. Construction of Access Control with Efficient Decryption

Let S_A and S_U denote the set of attribute authorities and the set of users in the system respectively. Let $\mathbb{G}$ and $\mathbb{G}_T$ be the multiplicative groups with the same prime order p and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be the bilinear map. Let g be the generator of $\mathbb{G}$. Let $H : \{0,1\}^* \to \mathbb{G}$ be a hash function such that the security is in the random oracle.

Our access control scheme consists of four phases: System Initialization, Key Generation, Encryption and Decryption.

Phase 1: System Initialization

There are two steps during the system initialization phase: the CA Setup and the *AA* Setup.

1) *CA Setup*

The CA runs the CA setup algorithm $\mathsf{CASetup}$. It takes a security parameter as input. The CA first chooses a random number $a \in \mathbb{Z}_p$ as the master key MSK of the system and compute the system parameter $\mathsf{SP} = g^a$. Then, the CA generates a pair of secret key and public key (sk_{CA}, pk_{CA}). The CA accepts both *User Registration* and *AA Registration*.

a) *User Registration*

When a user joins the system, the CA first authenticates this user. If the user is legal in the system, the CA then assigns a global unique user id *uid* to this user. After that, it generates the global public key $\mathsf{GPK}_{uid} = g^{u_{uid}}$ and the global secret key $\mathsf{GSK}_{uid} = z_{uid}$ by randomly choosing two numbers $u_{uid}, z_{uid} \in \mathbb{Z}_p$. The CA also generates a certificate by using its secret key sk_{CA} as $Sig_{sk_{CA}}(uid, u_{uid}, g^{\frac{1}{z_{uid}}})$. Then, the CA gives the global public key GPK_{uid}, the global secret key GSK_{uid} and the user's certificate $Sig_{sk_{CA}}(uid, u_{uid}, g^{1/z_{uid}})$ to the user with *uid*.

b) *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 authority id *aid* to this AA. Then, the CA sends its public key pk_{CA} to this authority, together with the system parameter SP.

2) *AA Setup*

Each $AA_k (k \in S_A)$ runs the authority setup algorithm $\mathsf{AASetup}$. Let S_{A_k} denote the set of all attributes managed by this authority AA_k. It chooses three random numbers $\alpha_k, \beta_k, \gamma_k \in \mathbb{Z}_p$ as the authority secret key $\mathsf{SK}_k = (\alpha_k, \beta_k, \gamma_k)$. For each attribute $x_k \in S_{A_k}$, the authority generates a public attribute key as $\mathsf{PK}_{x_k} = (g^{v_{x_k}} H(x_k))^{\gamma_k}$ by implicitly choosing an attribute version number v_{x_k}. All the public attribute keys are published on the board of AA_k.

Phase 2: Key Generation

Each *AA* runs the key generation algorithm KeyGen to generate the owner's public key (for encryption) and the user's secret key (for decryption).

a) *Public Key Generation*

The AA_k computes the authority public key as

$$\mathsf{PK}_k = \left(e(g,g)^{\alpha_k}, g^{\frac{1}{\beta_k}}, g^{\frac{\gamma_k}{\beta_k}} \right).$$

Each owner can construct the full public key as

$$\mathsf{PK} = \left(g, g^a, \{\mathsf{PK}_k\}_{k \in S_A}, \{\mathsf{PK}_{x_k}\}_{x_k \in S_{A_k}}^{k \in S_A} \right).$$

b) *User's Secret Key Generation*

For each user $U_j (j \in S_U)$, each $AA_k (k \in S_A)$ first authenticates whether this user is a legal user by verifying the certificate of the user. It decrypts the $Sig_{sk_{CA}}(uid_j, u_j, g^{1/z_j})$ and authenticates the user. If the user is a legal user, the AA_k assigns a set of attributes $S_{j,k}$ to this user according to its role or identity in its administration domain. Then, the AA_k generates the user's secret key $SK_{j,k}$ as

$$\begin{aligned} \mathsf{SK}_{j,k} = (\ & K_{j,k} = g^{\frac{\alpha_k}{z_j}} \cdot g^{a u_j}, L_{j,k} = g^{\frac{\beta_k}{z_j}}, \\ & \forall x_k \in S_{j,k} : K_{j,x_k} = g^{\frac{\beta_k \gamma_k}{z_j}} \cdot (g^{v_{x_k}} \cdot H(x_k))^{\gamma_k \beta_k u_j}\ ). \end{aligned}$$

where $j \in S_U$ and $k \in S_A$.

Phase 3: Encryption

The owner first encrypts the data component with a content key by using symmetric encryption methods. It then runs the encryption algorithm $\mathsf{Encrypt}$ to encrypt the content key. It takes as inputs the public key PK and the content keys k and an access structure (M, ρ) over all the selected attributes from the involved set of authorities I_A. Let M be a $\ell \times n$ matrix, where ℓ denotes the total number of all the attributes. The function ρ associates rows of M to attributes.

The encryption algorithm first chooses a random encryption exponent $s \in \mathbb{Z}_p$ and chooses a random vector $\vec{v} = (s, y_2, \cdots, y_n) \in \mathbb{Z}_p^n$, where $y_2, \cdots, y_n$ 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 $r_1, r_2, \cdots, r_\ell \in \mathbb{Z}_p$ and computes the ciphertext as

$$\begin{aligned} \mathsf{CT} = (\ & C = k \cdot (\prod_{k \in I_A} e(g,g)^{\alpha_k})^s,\ C' = g^s, \\ \forall i = 1\ to\ l : & C_i = g^{a\lambda_i} \cdot ((g^{v_{\rho(i)}} H(\rho(i)))^{\gamma_k})^{-r_i}, \\ & D_{1,i} = g^{\frac{r_i}{\beta_k}},\ D_{2,i} = g^{-\frac{\gamma_k}{\beta_k} r_i},\ \rho(i) \in S_{A_k}\ ). \end{aligned}$$

Phase 4: Decryption

The decryption phase consists of two steps: Server Token Generation and User Data Decryption.

a) *Server Token Generation*

The user $U_j (j \in S_U)$ sends its secret keys $\{\mathsf{SK}_{j,k}\}_{k \in S_A}$ to the server and asks the server to compute a decryption token for the ciphertext CT by running the token generation algorithm TKGen. Only when the attributes the user U_j possesses satisfy the access structure defined in the ciphertext CT, the server can successfully compute the decryption token TK.

Let I be $\{I_{A_k}\}_{k \in I_A}$, where $I_{A_k} \subset \{1, \cdots, l\}$ is defined as $I_{A_k} = \{i : \rho(i) \in S_{A_k}\}$. Let $N_A = |I_A|$ be the number of *AA*s involved in the ciphertext. 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 token generation algorithm algorithm computes the decryption token TK as

$$\begin{aligned} \mathsf{TK} &= \prod_{k \in I_A} \frac{e(C', K_{j,k})}{\prod_{i \in I_{A_k}} \left(e(C_i, GPK_{U_j}) \cdot e(D_{1,i}, K_{j,\rho(i)}) \cdot e(D_{2,i}, L_{j,k}) \right)^{w_i N_A}} \\ &= \frac{e(g,g)^{a u_j s N_A} \cdot \prod_{k \in I_A} e(g,g)^{\frac{\alpha_k}{z_j} s}}{e(g,g)^{u_j a N_A \sum_{i \in I} \lambda_i w_i}} \\ &= \prod_{k \in I_A} e(g,g)^{\frac{\alpha_k}{z_j} s}. \end{aligned} \tag{1}$$

It outputs the decryption token TK for the ciphertext CT and sends it to the user U_j.

b) *User Data Decryption*

Upon receiving this decryption token TK, the user U_j can decrypt the ciphertext by using its global secret key $\mathsf{GSK}_{U_j} = z_j$ as

$$k = \frac{C}{\mathsf{TK}^{z_j}}.$$

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

C. Efficient Attribute Revocation

Suppose an attribute $\tilde{x}_k$ of the user U_μ is revoked from the AA_k. The attribute revocation includes three phases: *Update Key Generation*, *Key Update* and *Ciphertext Update*. The key update can prevent the revoked user from decrypting the new data which is encrypted by the new public keys (Forward Security). The ciphertext update can make sure that the newly joined user can still access the previous data which is published before it joins the system, when its attributes satisfy the access policy associated with the ciphertext (Backward Security).

Phase 1. Update Key Generation

The authority AA_k runs the update key generation algorithm $\mathsf{UKeyGen}$ to compute the update keys. The algorithm takes as inputs the authority secret key SK_k, the current attribute version key $v_{\tilde{x}_k}$ and the user's global public keys GPK_{U_j}. It

TABLE I
COMPREHENSIVE COMPARISON OF CP-ABE WITH ATTRIBUTE REVOCATION SCHEMES

Scheme	Authority	Computation		Revocation Message ($\|p\|$)	Revocation Security		Revocation Enforcer	Ciphertext Updater
		Encrypt	Decrypt*		Forward	Backward		
Hur's [15]	Single	$O(t_c + \log n_u)$	$O(t_u)$	$O(n_{non,x} \log \frac{n_u}{n_{non,x}})$	Yes	Yes	Server†	Server†
DACC [16]	Multiple	$O(t_c)$	$O(t_u)$	$O(n_{c,x} \cdot n_{non,x})$	Yes	No	Owner	Owner
DAC-MACS	Multiple	$O(t_c)$	$O(1)$	$O(n_{non,x})$	Yes	Yes	AA	Server‡

$*$: The decryption computation on the user; †: The server is fully trusted; ‡: The server is semi-trusted.

generates a new attribute version key $v'_{\tilde{x}_k}$. It first calculates the Attribute Update Key as $\mathsf{AUK}_{\tilde{x}_k} = \gamma_k(v'_{\tilde{x}_k} - v_{\tilde{x}_k})$, then it applies this $\mathsf{AUK}_{\tilde{x}_k}$ to compute the User Update Key $\mathsf{UUK}_{j,\tilde{x}_k} = g^{u_j \beta_k \cdot \mathsf{AUK}_{\tilde{x}_k}}$ and the Ciphertext Update Key as $\mathsf{CUK}_{\tilde{x}_k} = \frac{\beta_k}{\gamma_k} \cdot \mathsf{AUK}_{\tilde{x}_k}$). Then, the AA_k updates the public attribute key of the revoked attribute $\tilde{x}_k$ as $\mathsf{PK}'_{\tilde{x}_k} = \mathsf{PK}_{\tilde{x}_k} \cdot g^{\mathsf{AUK}_{\tilde{x}_k}}$ and broadcasts a message for all the owners that the public attribute key of the revoked attribute $\tilde{x}_k$ is updated. Then, all the owners can update their public key by getting the new public attribute key. It outputs both the user update key $\mathsf{UUK}_{j,\tilde{x}_k} (j \in S_U, j \neq \mu, \tilde{x}_k \in S_{j,k})$ and the ciphertext update key $\mathsf{CUK}_{\tilde{x}_k}$.

Phase 2. Key Update

For each non-revoked user $U_j (j \in S_U)$ who has the attribute $\tilde{x}_k$, the AA_k sends the corresponding user update key $\mathsf{UUK}_{j,\tilde{x}_k}$ to it. Upon receiving the user update key $\mathsf{UUK}_{j,\tilde{x}_k}$, the user $U_j (j \in S_U)$ runs the key update algorithm KeyUpdate to update its secret key as

$$\begin{aligned} \mathsf{SK}'_{j,k} = (\ & K'_{j,k} = K_{j,k}, L'_{j,k} = L_{j,k}, \\ & K'_{j,\tilde{x}_k} = K_{j,\tilde{x}_k} \cdot \mathsf{UUK}_{j,\tilde{x}_k}, \\ & \forall x \in S_u, x \neq \tilde{x} : K'_{j,k} = K_{j,k}\ ). \end{aligned}$$

Note: Because the $\mathsf{UUK}_{j,\tilde{x}_k}$ are distinguishable for different non-revoked users, the revoked user U_μ cannot use any other user's update keys to update its secret key.

Phase 3. Ciphertext Update

The AA_k sends a ciphertext update key $\mathsf{CUK}_{\tilde{x}_k}$ to the server. Upon receiving the $\mathsf{CUK}_{\tilde{x}_k}$, the server runs the ciphertext update algorithm CiphertextUpdate to update all the ciphertexts which are associated with the revoked attribute $\tilde{x}_k$. It takes inputs as the current ciphertext CT and the $\mathsf{CUK}_{\tilde{x}_k}$. It only needs to update those components of the ciphertext, which are associated with the revoked attribute $\tilde{x}_k$. The new ciphertext CT' is published as

$$\begin{aligned} \mathsf{CT}' = (\ & C = k \cdot (\prod_{k \in I_A} e(g,g)^{\alpha_k})^s,\ C' = g^s, \\ \forall i = 1\ to\ l : & C_i = g^{a\lambda_i} \cdot ((g^{v_{x_k}} H(x_k))^{\gamma_k})^{-r_i},\ D_{1,i} = g^{\frac{r_i}{\beta_k}}, \\ & D_{2,i} = g^{-\frac{\gamma_k}{\beta_k} r_i},\ if\ \rho(i) \neq \tilde{x}_k, \\ & C'_i = C_i \cdot D_{2,i}^{\mathsf{CUK}_{\tilde{x}_k}},\ D_{1,i} = g^{\frac{r_i}{\beta_k}}, \\ & D_{2,i} = g^{-\frac{\gamma_k}{\beta_k} r_i},\ if\ \rho(i) = \tilde{x}_k\ ). \end{aligned} \tag{2}$$

DAC-MACS only requires to update the revoked attribute associated component of the ciphertext, while the other components which are not related to the revoked attributes are not changed. Thus, this can greatly improve the efficiency of attribute revocation.

IV. ANALYSIS OF OUR DAC-MACS

In this section, we first provide a comprehensive analysis of our DAC-MACS. Then, we give the security analysis and performance analysis.

A. Comprehensive Analysis

Let $|p|$ be the size of element in the groups with the prime order p. Let t_c be the total number of attributes in a ciphertext and t_u be the total number of attributes of a user. Let n_u denote the number of users in the system. For the revoked attribute x, let $n_{non,x}$ be the number of non-revoked users who hold the revoked attribute and let $n_{c,x}$ be the number of ciphertexts which contain the revoked attribute.

Table I shows the comparison among our DAC-MACS and two existing schemes, which are all based on the ciphertext re-encryption to achieve the attribute revocation. We conduct the comparison in terms of the support of multi-authority, the computation efficiency (encryption on the owner and decryption on the user), the revocation communication cost, the revocation security, the revocation enforcer and the ciphertext updater/re-encryptor. From the table, we can see that DAC-MACS incurs less computation cost for the decryption on the user and less communication cost for the revocation. In DAC-MACS, the attribute revocation is controlled and enforced by each AA independently, but the ciphertexts are updated by the semi-trusted server, which can greatly reduce the workload on the owner. For the security of attribute revocation, DAC-MACS can achieve both forward security and backward security.

B. Security Analysis

Under the security model we defined in Section II-B, we conclude the security analysis into 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: Suppose we have an adversary $\mathcal{A}$ with non-negligible advantage $\varepsilon = Adv_{\mathcal{A}}$ in the selective security game against our construction and suppose it chooses a challenge matrix M^* with the dimension at most $q-1$ columns. In the

security game, the adversary can query any secret keys that cannot be used for decryption in combination with any keys it can obtain from the corrupted *AA*s. With these constraints, the security game in multi-authority systems can be treated equally to the one in single authority systems. Similarly, we can build a simulator $\mathcal{B}$ that plays the decisional q-parallel BDHE problem with non-negligible advantage. The detailed proof will be described in the Appendix. ■

Theorem 2. *DAC-MACS is secure against the collusion attack.*

Proof: In DAC-MACS, each user in the system is assigned with a global unique identity *uid*, and all the secret keys issued to the same user from different *AA*s are associated to the *uid* of this user. Thus, it is impossible for two or more users to collude and decrypt the ciphertext. Moreover, due to the unique *aid* of each *AA*, all the attributes are distinguishable, even though some *AA*s may issue the same attribute. This can prevent the user from replacing the components of a secret key issued by an *AA* with those components from other secret keys issued by another *AA*. ■

Privacy-Preserving Guarantee Due to the decryption outsourcing, the server can get the users' secret keys. However, the server still cannot decrypt the ciphertext without the knowledge of the users' global secret keys. Moreover, the ciphertext update is done by using the proxy re-encryption method, thus the server does not need to decrypt the ciphertext.

C. Performance Analysis

We conduct the performance analysis between our DAC-MACS and the Ruj's DACC scheme under the metrics of *Storage Overhead*, *Communication Cost* and *Computation Cost*.

1) Storage Overhead

The storage overhead is one of the most significant issues of the access control scheme in cloud storage systems. Suppose there are N_A *AA*s in the system. Let $|p|$ be the element size in the $\mathbb{G}, \mathbb{G}_T, \mathbb{Z}_p$. Let $n_{a,k}$ and $n_{a,k,uid}$ denote the total number of attributes managed by AA_k and the number of attributes assigned to the user with *uid* from AA_k respectively. We compare the storage overhead on each entity in the system, as shown in Table II.

In DAC-MACS, the storage overhead on each AA_k consists of the version number of each attribute and the authority secret key. From Table II, we can see that DAC-MACS incurs less storage overhead on each AA_k than Ruj's DACC scheme, which consists of the secret keys for all the attributes. The public parameters contribute the main storage overhead on the owner. Besides the public parameters, Ruj's DACC scheme requires the owner to re-encrypt the ciphertexts, thus 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. The storage overhead on each user in DAC-MACS comes from the global secret key issued by the CA and the secret keys issued by all the *AA*s. However, in Ruj's DACC scheme, the storage overhead on each user consists of both the security keys issued by all the *AA*s and the ciphertext components that associated with the revoked attribute. That is because when the ciphertext is re-encrypted, some of its components related to the revoked attributes should be send to each non-revoked user who holds the revoked attributes. 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).

TABLE II
COMPARISON OF STORAGE OVERHEAD

Entity	Ruj's DACC [16]	Our DAC-MACS
AA_k	$2n_{a,k}\|p\|$	$(n_{a,k}+3)\|p\|$
Owner	$(n_c+2\sum_{k=1}^{N_A} n_{a,k})\|p\|$	$(3N_A+1+\sum_{k=1}^{N_A} n_{a,k})\|p\|$
User	$(n_{c,x}+\sum_{k=1}^{N_A} n_{a,k,uid})\|p\|$	$(2N_A+1+\sum_{k=1}^{N_A} n_{a,k,uid})\|p\|$
Server	$(3l+1)\|p\|$	$(3l+2)\|p\|$

n_c: total number of ciphertexts stored on the cloud server; $n_{c,x}$:number of ciphertexts contain the revoked attribute x; l: total number of attributes that appeared in the ciphertext.

2) Communication Cost

The communication cost of the normal access control is almost the same between our DAC-MACS and Ruj's DACC scheme. Here, we only compare the communication cost of attribute revocation, as shown in Table III. It is easily to find that the communication cost of attribute revocation in Ruj's scheme is linear to the number of ciphertexts which contain the revoked attributes. Due to the large number of ciphertext in cloud storage system, Ruj's scheme incurs a heavy communication cost for attribute revocation.

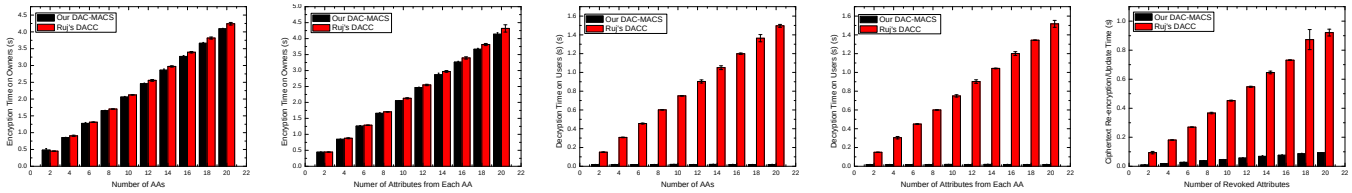
TABLE III
COMPARISON OF COMMUNICATION COST FOR ATTRIBUTE REVOCATION

Operation	Ruj's DACC [16]	Our DAC-MACS
Key Update	N/A	$n_{non,x}\|p\|$
Ciphertext Update	$(n_{c,x}\cdot n_{non,x}+1)\|p\|$	$\|p\|$

$n_{non,x}$ is the number of non-revoked users who hold the revoked attribute x; $n_{c,x}$ is the number of ciphertexts which contain the revoked attribute x.

3) Computation Cost

We simulate the computation time of encryption, decryption and ciphertext re-encryption/update in both our DAC-MACS and Ruj's DACC scheme. We do the simulation on a Linux system with an Intel Core 2 Duo CPU at 3.16GHz and 4.00GB RAM. The code uses the Pairing-Based Cryptography (PBC) library version 0.5.12 to simulate 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.



(a) Encryption (b) Encryption (c) Decryption (d) Decryption (e) Re-encryption/Update

Fig. 2. Comparison of Encryption, Decryption and Ciphertext Re-encryption/Update Time

We compare the computation efficiency of both encryption and decryption in two criteria: the number of authorities and the number of attributes per authority, as shown in Fig. 2. Fig.2(a) describes the comparison of encryption time versus the number of *AA*s, where the involved number of attributes from each *AA* is set to be 10. Fig.2(b) gives the encryption time comparison versus the number of attributes from each *AA*, where the involved number of *AA*s is set to be 10. Suppose the user has the same number of attributes from each *AA*. Fig.2(c) shows the comparison of decryption time versus the number of *AA*s, where the number of attributes the user holds from each *AA* is set to be 10. Fig.2(d) describes the decryption time comparison versus the number of attributes the user holds from each *AA*. In Fig.2(d), the number of authority for the user is fixed to be 10. Fig.2(e) gives the comparison of ciphertext re-encryption/update versus the number of revoked attributes appeared in the ciphertext.

The simulation results show that our DAC-MACS incurs less computation cost on the encryption of owners, the decryption of users and the re-encryption of ciphertexts.

V. Related Work

Cryptographic techniques are well applied to access control for remote storage systems [17]–[19]. Traditional public key encryption (PKE) based schemes [20], [21] either incurs complicated key management or produces multiples copies of encrypted data with different user's keys. Some methods [22], [23] deliver the key management and distribution from the data owners to the remote server under the assumption that the server is trusted or semi-trusted. However, the server is not fully trusted in cloud storage systems and thus these methods cannot be applied to access control for cloud storage systems.

Attribute-based Encryption (ABE) is a promising technique that is designed for access control of encrypted data. After Sahai and Waters introduced the first ABE scheme [24], Goyal *et al.* [25] formulated the ABE into two complimentary forms: Key-Policy ABE (KP-ABE) and Ciphertext-Policy ABE (CP-ABE). There are a number of works used ABE to realize fine-grained access control for outsourced data [15], [26], [27]. In these schemes, a trusted single authority is used to manage the attributes and issue keys. However, in real storage systems, the authority can fail or be corrupted, which may leak out the data since the authority can decrypt all the encrypted data. Moreover, the authority may become the performance bottleneck in the large scale cloud storage systems.

Some new cryptographic methods are proposed to the multi-authority ABE problem [7]–[10], [28], [29]. Chase [7] proposed a solution that introduced a global identifier to tie users' keys together. The proposed scheme also relies on a central authority to provide a final secret key to integrate the secret keys from different attribute authorities. However, since the central authority would be able to decrypt all the ciphertext in the Chase's scheme, the central authority would be a vulnerable point for security attacks and a performance bottleneck for large scale systems. Another limitation of Chase's scheme is that it can only express a strict "AND" policy over a pre-determined set of authorities. To improve the Chase's scheme, Muller *et al.* [8] proposed a multi-authority ABE scheme that can handle any expressions in LSSS access policy, but it also requires a centralized authority. Chase *et al.* [9] also proposed a method to remove the central authority by using a distributed PRF (pseudo-random function). But it has the same limitation to strict "AND" policy of pre-determined authorities. Lin *et al.* [28] proposed a decentralized scheme based on threshold mechanism. In this scheme, the set of authorities is pre-determined and it requires the interaction among the authorities during the system setup. This scheme can tolerate collusion attacks for up to m colluding users, where m is a system parameter chosen at setup time. In [10], Lewko *et al.* proposed a new comprehensive scheme, which does not require any central authority. It is secure against any collusion attacks and it can process the access policy expressed in any Boolean formula over attributes. However, their method is constructed in composite order bilinear groups that incurs heavy computation cost. They also proposed a multi-authority CP-ABE scheme constructed in prime order group, but they did not consider attribute revocation.

There are a number of works about the revocation in ABE systems in the cryptography literature [2]–[6]. However, these methods either only support the user level revocation or rely on the server to conduct the attribute revocation. Moreover, these attribute revocation methods are designed only for ABE systems with single authority. Ruj *et al.* [16] designed a DACC scheme and proposed an attribute revocation method for the Lewko and Waters' decentralized ABE scheme. However, their attribute revocation method incurs a heavy communication cost since it requires the data owner to transmit a new ciphertext component to every non-revoked user. Li et al. [30] proposed an attribute revocation method for multi-authority ABE systems, but their methods is only for KP-ABE systems.

Green *et al.* [14] proposed two ABE schemes that outsource the decryption to the server. In their schemes, the authority separate the traditional secret key into a user secret key and a transformation key. However, their schemes are designed only for the single authority systems and do not support for the multi-authority systems. That is because each authority may generate different user's secret key, such that the transformation keys cannot be combined together to transform the ciphertext into a correct intermediate value.

VI. Conclusion

In this paper, we proposed an effective data access control scheme for multi-authority cloud storage systems, DAC-MACS. We also construct a new multi-authority CP-ABE scheme, in which the main computation of decryption is outsourced to the server. We further designed an efficient attribute revocation method that can achieve both forward security and backward security. Our attribute revocation methods incurs less communication cost and less computation cost of the revocation, where only those components associated with the revoked attribute in the secret keys and the ciphertext need to be updated. Through the analysis and the simulation, we showed that our DAC-MACS is provably secure in the random oracle model and incurs less storage overhead, communication cost and computation cost, compared to existing schemes.

References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Tech. Rep., 2009.

[2] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy (S&P'07)*. IEEE Computer Society, 2007, pp. 321–334.

[3] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proceedings of the 4th International Conference on Practice and Theory in Public Key Cryptography (PKC'11)*. Springer, 2011, pp. 53–70.

[4] V. Goyal, A. Jain, O. Pandey, and A. Sahai, "Bounded ciphertext policy attribute based encryption," in *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*. Springer, 2008, pp. 579–591.

[5] R. Ostrovsky, A. Sahai, and B. Waters, "Attribute-based encryption with non-monotonic access structures," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07)*. ACM, 2007, pp. 195–203.

[6] 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 *Proceedings of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology - EUROCRYPT'10*. Springer, 2010, pp. 62–91.

[7] M. Chase, "Multi-authority attribute based encryption," in *Proceedings of the 4th Theory of Cryptography Conference on Theory of Cryptography (TCC'07)*. Springer, 2007, pp. 515–534.

[8] S. Müller, S. Katzenbeisser, and C. Eckert, "Distributed attribute-based encryption," in *Proceedings of the 11th International Conference on Information Security and Cryptology (ICISC'08)*. Springer, 2008, pp. 20–36.

[9] M. Chase and S. S. M. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09)*. ACM, 2009, pp. 121–130.

[10] A. B. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proceedings of the 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology - EUROCRYPT'11*. Springer, 2011, pp. 568–588.

[11] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proceedings of the 4st Annual International Cryptology Conference: Advances in Cryptology - CRYPTO'84*. Springer, 1984, pp. 47–53.

[12] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *Proceedings of the 21st Annual International Cryptology Conference: Advances in Cryptology - CRYPTO'01*. Springer, 2001, pp. 213–229.

[13] C. Cocks, "An identity based encryption scheme based on quadratic residues," in *Proceedings of the 8th IMA International Conference on Cryptography and Coding*. Springer, 2001, pp. 360–363.

[14] M. Green, S. Hohenberger, and B. Waters, "Outsourcing the decryption of abe ciphertexts," in *Proceedings of the 20th USENIX Security Symposium*. USENIX Association, 2011.

[15] J. Hur and D. K. Noh, "Attribute-based access control with efficient revocation in data outsourcing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 7, pp. 1214–1221, 2011.

[16] S. Ruj, A. Nayak, and I. Stojmenovic, "DACC: Distributed Access Control in Clouds," in *Proceeding of the 10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom'11)*. IEEE, 2011, pp. 91–98.

[17] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03)*. USENIX, 2003.

[18] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius: Securing remote untrusted storage," in *Proceedings of the Network and Distributed System Security Symposium (NDSS'03)*. The Internet Society, 2003.

[19] D. Naor, M. Naor, and J. Lotspiech, "Revocation and tracing schemes for stateless receivers," *Electronic Colloquium on Computational Complexity (ECCC)*, no. 043, 2002.

[20] J. Benaloh, M. Chase, E. Horvitz, and K. Lauter, "Patient controlled encryption: ensuring privacy of electronic medical records," in *Proceedings of the first ACM Cloud Computing Security Workshop (CCSW'09)*. ACM, 2009, pp. 103–114.

[21] C. Dong, G. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," *Journal of Computer Security*, vol. 19, no. 3, pp. 367–397, 2011.

[22] E. Damiani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Key management for multi-user encrypted databases," in *Proceedings of the 2005 ACM Workshop On Storage Security And Survivability (StorageSS'05)*. ACM, 2005, pp. 74–83.

[23] W. Wang, Z. Li, R. Owens, and B. K. Bhargava, "Secure and efficient access to outsourced data," in *Proceedings of the first ACM Cloud Computing Security Workshop (CCSW'09)*. ACM, 2009, pp. 55–66.

[24] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology - EUROCRYPT'05*. Springer, 2005, pp. 457–473.

[25] 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 (CCS'06)*. ACM, 2006, pp. 89–98.

[26] S. Yu, C. Wang, K. Ren, and W. Lou, "Attribute based data sharing with attribute revocation," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS'10)*. ACM, 2010, pp. 261–270.

[27] S. Jahid, P. Mittal, and N. Borisov, "Easier: encryption-based access control in social networks with efficient revocation," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS'11)*. ACM, 2011, pp. 411–415.

[28] H. Lin, Z. Cao, X. Liang, and J. Shao, "Secure threshold multi authority attribute based encryption without a central authority," *Inf. Sci.*, vol. 180, no. 13, pp. 2618–2632, 2010.

[29] J. Li, Q. Huang, X. Chen, S. S. M. Chow, D. S. Wong, and D. Xie, "Multi-authority ciphertext-policy attribute-based encryption with accountability," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS'11)*. ACM, 2011, pp. 386–390.

[30] 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 Transactions on Parallel and Distributed Systems*, 2012.

APPENDIX A
BACKGROUND

We give some formal definitions for access structures, Linear Secret Sharing Schemes (LSSS) and the background information on Bilinear Pairings.

A. Access Structures

Definition 3 (Access Structure)**.** *Let $\{P_1, P_2, \cdots, P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{\{P_1, P_2, \cdots, P_n\}}$ is monotone if $\forall B, C$ if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\{P_1, P_2, \cdots, P_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{P_1, P_2, \cdots, P_n\}} \setminus \{\emptyset\}$. The sets in $\mathbb{A}$ are called the authorized sets, and the sets not in $\mathbb{A}$ are called the unauthorized sets.*

In our proposed scheme, the role of the parties is taken by the attributes. Thus, the access structure $\mathbb{A}$ will contain the authorized sets of attributes. We restrict our attention to monotone access structures. From now on, unless stated otherwise, by an access structure we mean a monotone access structure.

B. Linear Secret Sharing Schemes

We give our definitions of Linear Secret Sharing Schemes (LSSS) as

Definition 4 (Linear Secret-Sharing Schemes (LSSS))**.** *A secret-sharing scheme Π over a set of parties $\mathcal{P}$ is called linear (over $\mathbb{Z}_p$) if*

1) *The shares for each party form a vector over $\mathbb{Z}_p$.*
2) *There exists a matrix M called the share-generating matrix for Π. The matrix M has l rows and n columns. For all $i = 1, \cdots, l$, the i-th row of M 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 $s \in \mathbb{Z}_p$ is the secret to be shared and $r_2, \cdots, r_n \in \mathbb{Z}_p$ 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 $\rho(i)$.*

Every linear secret sharing-scheme according to the above definition also enjoys the *linear reconstruction* property: Suppose that Π is a LSSS for the access structure $\mathbb{A}$. Let $S \in \mathbb{A}$ be any authorized set, and let $I \subset \{1, 2, \cdots, l\}$ be defined as $I = \{i : \rho(i) \in S\}$. Then, there exist constants $\{w \in \mathbb{Z}_p\}_{i \in I}$ such that, for any valid shares $\{\lambda_i\}$ of a secret s according to Π, we have $\sum_{i \in I} w_i \lambda_i = s$. These constants $\{w_i\}$ can be found in time polynomial in the size of the share-generating matrix M. We note that for unauthorized sets, no such constants $\{w_i\}$ exist.

C. Bilinear Pairing

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three multiplicative groups with the same prime order p. A bilinear map is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

1) Bilinearity: $e(u^a, v^b) = e(u, v)^{ab}$ for all $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$.
2) Non-degeneracy: There exist $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ such that $e(u, v) \neq I$, where I is the identity element of $\mathbb{G}_T$.
3) Computability: e can be computed in an efficient way.

Such a bilinear map is called a bilinear pairing.

D. 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 $a, s \in \mathbb{Z}_p$ be chosen at random and g be a generator of $\mathbb{G}$. If an adversary is given

$$\begin{aligned}\vec{y} = (\ & g, g^s, g^{1/z}, g^{a/z}, \cdots, g^{(a^q/z)}, \\ & g^a, g^{(a^q)}, \ , g^{(a^{q+1})}, \cdots, g^{(a^{2q})}, \\ & \forall_{1 \le j \le q} \quad g^{s \cdot b_j}, \ g^{a/b_j}, \cdots, g^{(a^q/b_j)}, \ , g^{(a^{2q}/b_j)}, \\ & \forall_{1 \le j,k \le q, k \neq j} \quad g^{a \cdot s \cdot b_k / b_j}, \cdots, g^{(a^q \cdot s \cdot b_k / b_j)} \ ),\end{aligned}$$

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

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

Definition 5. *The decisional q-parallel BDHE assumption holds if no polynomial time algorithm has a non-negligible advantage in solving the q-parallel BDHE problem.*

E. Security Proof of Our Proposed Multi-authority CP-ABE

We prove that our multi-authority access control is secure under security model we defined, which can be summarized as in the following theorem.

Theorem 3. *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 $n^* \le q$.*

Proof: Suppose we have an adversary $\mathcal{A}$ with non-negligible advantage $\varepsilon = Adv_{\mathcal{A}}$ in the selective security game against our construction and suppose it chooses a challenge matrix M^* with the dimension at most $q - 1$ columns. In the security game, the adversary can query any secret keys that cannot be used for decryption in combination with any keys it can obtain from the corrupted AAs. With these constraints, the security game in multi-authority systems can be treated equally to the one in single authority systems. Therefore, we can build a simulator $\mathcal{B}$ that plays the decisional q-parallel BDHE problem with non-negligible advantage as follows.

Init. The simulator takes in the q-parallel BDHE challenge $\vec{y}$, T. The adversary gives the algorithm the challenge access structure M^*, ρ^*, where M^* has n^* columns.

Setup. The simulator runs the CASetup and AASetup algorithm, and gives g to the adversary. The adversary chooses a set of $S'_A \subset S_A$ of corrupted authorities, and reveals these to the simulator. For each uncorrupted authority $AA_k (k \in S_A - S'_A)$,

the simulator randomly chooses $\alpha'_k \in \mathbb{Z}_p (k \in S_A - S'_A)$ and implicitly sets $\alpha_k l = \alpha'_k + a^{q+1}$ by letting

$$e(g,g)^{\alpha_k} = e(g^a, g^{a^q}) e(g,g)^{\alpha'_k}. \quad (3)$$

Then, we describe how the simulator programs the random oracle H by building a table. Consider a call to $H(x)$, if $H(x)$ was already defined in the table, then the oracle returns the same answer as before. Otherwise, begin by choosing a random value d_x. Let X denote the set of indices i, such that $\rho^*(i) = x$. In other words, all the row indices in the set X match the same attribute x. The simulator programs the oracle as

$$H(x) = g^{d_x} \prod_{i \in X} g^{aM^*_{i,1}/b_i} \cdot g^{a^2 M^*_{i,2}/b_i} \cdots g^{a^n M^*_{i,n}/b_i}. \quad (4)$$

Note that if $X = \emptyset$ then we have $H(x) = g^{d_x}$. Also note that the response from the oracle are distributed randomly due to the g^{d_x} value.

The simulator also randomly chooses two numbers $\beta_k, \gamma_k \in \mathbb{Z}_p$. Then, it generate the public key of each uncorrupted authority AA_k as

$$\mathsf{PK}_k = \left(e(g,g)^{\alpha_k}, g^{\frac{1}{\beta_k}}, g^{\frac{\gamma_k}{\beta_k}} \right).$$

The public attribute keys PK_{x_k} can be simulated by randomly choosing a version number $v_{x_k} \in \mathbb{Z}_p$ as

$$\mathsf{PK}_{x_k} = (g^{v_{x_k} + d_{x_k}} \prod_{i \in X} g^{aM^*_{i,1}/b_i} \cdot g^{a^2 M^*_{i,2}/b_i} \cdots g^{a^n M^*_{i,n}/b_i})^{\gamma_k}.$$

Secret Key Query Phase 1. In this phase, the simulator answers secret key queries from the adversary. Suppose the adversary makes secret key queries by submitting pairs $(Sig_{sk_{CA}}(uid), S_k)$ to the simulator, where S_k is a set of attributes belonging to an uncorrupted authority AA_k and $Sig_{sk_{CA}}(uid)$ is the user certificate issued by the CA. Suppose S_k does not satisfy M^* together with any keys that can obtain from corrupted authorities.

The simulator first authenticate the user's certificate by using the public key of the CA, then the simulator chooses a random number $r \in \mathbb{Z}_p$. It then finds a vector $\vec{w} = (w_1, w_2, \cdots, w_{n^*}) \in \mathbb{Z}_p^{n^*}$, such that $w_1 = -1$ and for all i where $\rho^*(i) \in S_k$ we have that $\vec{w} \cdot M^*_i = 0$. By the definition of a LSSS, such a vector must exist, since S_k does not satisfy M^*.

The simulator then implicitly defines u as

$$u = \frac{1}{z}(r + w_1 a^q + w_2 a^{q-1} + \cdots + w_{n^*} a^{q-n^*+1})$$

by setting

$$\mathsf{GPK}_{uid} = (g^{\frac{1}{z}})^r \prod_{i=1,\ldots,n^*} (g^{\frac{a^{q+1-i}}{z}})^{w_i} = g^u.$$

The simulator constructs $L_{uid,k}$ as

$$L_{uid,k} = g^{\frac{\beta_k}{z}}.$$

From the definition of g^u, we find that g^{au} contains a term of $g^{a^{q+1}}$, which will cancel out with the unknown term in $(g^{\frac{1}{z}})^{\alpha_k}$ when creating $K_{uid,k}$. The simulator can calculate $K_{uid,k}$ as

$$K_{uid,k} = g^{\frac{\alpha'_k}{z}} (g^{\frac{a}{z}})^r \prod_{i=2,\ldots,n^*} \left(g^{\frac{a^{q+2-i}}{z}} \right)^{w_i}.$$

For the calculation of $K_{x_k,uid,k} (\forall x_k \in S_k)$, if x is used in the access structure, we must make sure that there are no terms of the form g^{a^{q+1}/b_i} that we cannot simulate. However, we have that $\vec{w} \cdot M^*_i = 0$. Therefore, all of these terms cancel out.

The simulator computes K_{x_k,uid,S_k} as follows.

$$K_{x_k,uid,S_k} = g^{\frac{\beta_k \gamma_k}{z}} \cdot (\mathsf{GPK}_{uid})^{\beta_k \gamma_k (v_{x_k} + d_{x_k})} \cdot \prod_{i \in X} \prod_{j=1,\ldots,n^*} \left((g^r)^{\beta_k \gamma_k} \prod_{k=1,\ldots,n^*, k \neq j} \left(g^{a^{q+1+j-k}/b_i} \right)^{\beta_k \gamma_k w_k} \right)^{M^*_{i,j}}$$

If the attribute $x \in S_{AID}$ is not used in the access structure. That is there is no i such that $\rho^*(i) = x$. For those attributes, we can let

$$K_{x_k,uid,k} = g^{\frac{\beta_k \gamma_k}{z}} \cdot (\mathsf{GPK}_{uid})^{\beta_k \gamma_k (v_{x_k} + d_{x_k})}.$$

Challenge. In this phase, the simulator programs the challenge ciphertext. The adversary gives two messages m_0, m_1 to the simulator. The simulator flips a coin b. It creates

$$C = m_b T \cdot \prod_{k \in I_A} e(g^s, g^{\alpha'_{AID_k}})$$

and $C' = g^s$.

The difficult part is to simulate the C_i values since this contains terms that must be canceled out. However, the simulator can choose the secret splitting, such that these can be canceled out. Intuitively, the simulator will choose random $y'_2, \cdots, y'_{n^*}$ and share the secret s using the vector

$$\vec{v} = (s, sa + y'_2, sa^2 + y'_3, \cdots, sa^{n^*-1} + y'_{n^*}) \in \mathbb{Z}_p^{n^*}.$$

It also chooses random values $r'_1, \cdots, r'_\ell$.

For $i = 1, \ldots, n^*$, let R_i be the set of all $k \neq i$ such that $\rho^*(i) = \rho^*(k)$. That is the set of all other row indices that have the same attribute as row i. The challenge ciphertext components can be generated as

$$D_{1,i} = \left(g^{r'_i} g^{sb_i} \right)^{\frac{1}{\beta_k}}$$

and

$$D_{2,i} = \left(g^{r'_i} g^{sb_i} \right)^{\frac{-\gamma_k}{\beta_k}}.$$

From the vector $\vec{v}$, we can construct the share of the secret as

$$\lambda_i = s \cdot M^*_{i,1} + \sum_{j=2,\ldots,n^*} (sa^{j-1} + y'_j) M^*_{i,j}$$

Then, we can simulate the C_i as

$$C_i = (g^{v_{\rho^*(i)}} \cdot H(\rho^*(i)))^{\gamma_k r'_i} \cdot \left(\prod_{j=1,\ldots,n^*} g^{aM_{i,j} y_j} \right) \cdot \left(g^{b_i s} \right)^{-\gamma_k (v_{\rho^*(i)} + d_{\rho^*(i)})} \cdot \left(\prod_{k \in R_i} \prod_{j=1,\ldots,n^*} (g^{a^j s (b_i/b_k)})^{\gamma_k M^*_{k,j}} \right).$$

Secret Key Query Phase 2. Same as Phase 1.

Guess. The adversary will eventually output a guess b' of b. If $b' = b$, the simulator then outputs 0 to show that $T = e(g,g)^{a^{q+1}s}$; otherwise, it outputs 1 to indicate that it believes T is a random group element in $\mathbb{G}_T$.

When T is a tuple, the simulator $\mathcal{B}$ gives a perfect simulation so we have that

$$Pr[\mathcal{B}(\vec{y}, T = e(g,g)^{a^{q+1}s}) = 0] = \frac{1}{2} + Adv_{\mathcal{A}}.$$

When T is a random group element the message m_b is completely hidden from the adversary and we have at

$$Pr[\mathcal{B}(\vec{y}, T = e(g,g)^{a^{q+1}s}) = 0] = \frac{1}{2}.$$

Therefore, $\mathcal{B}$ can play the decisional q-parallel BDHE game with non-negligible advantage. ■