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

Kan Yang, *Associate Member, IEEE*, Xiaohua Jia, *Fellow, IEEE*, Kui Ren, *Senior Member, IEEE*, Bo Zhang, *Member, IEEE*, and Ruitao Xie, *Student Member, IEEE*

***Abstract*—Data access control is an effective way to ensure 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 Existing access control schemes are no longer applicable to cloud storage systems, because they either produce multiple encrypted copies of the same data or require a fully trusted cloud server.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. However, due to the inefficiency of decryption and revocation, existing CP-ABE schemes cannot be directly applied to construct a data access control scheme for multiauthority cloud storage systems, where users may hold attributes from multiple authorities. In this paper, we propose data access control for multiauthority cloud storage (DAC-MACS), an effective and secure data access control scheme with efficient decryption and revocation. Specifically, we construct a new multiauthority CP-ABE scheme with efficient decryption, and also design an efficient attribute revocation method that can achieve both forward security and backward security. We further propose an extensive data access control scheme (EDAC-MACS), which is secure under weaker security assumptions.**

***Index Terms*—Access control, attribute revocation, CP-ABE, decryption outsourcing, multiauthority cloud.**

I. Introduction

CLOUD storage is an important service of cloud computing [1]. It allows data owners to host their data in the cloud and rely on cloud servers to provide "24/7/365" data access to users (data consumers). Data access control is an effective way to ensure the data security in the cloud. However, due to the data outsourcing, the cloud server cannot be fully trusted to provide data access control service, which means existing server-based access control methods are no longer applicable to cloud storage systems. To achieve data access control on untrusted servers, traditional methods usually encrypt the data and only users holding valid keys are able to decrypt. Although these methods can provide secure data access control, the key management is very complicated when more users are in the system. Data owners also have to stay online all the time to deliver keys to new users. Moreover, for each data, there are multiple copies of ciphertexts for users with different keys, which will incur high storage overhead on the server.

Ciphertext-Policy Attribute-based Encryption (CP-ABE) [2]–[6] 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 ciphertexts only when its attributes satisfy the access policies.

In cloud storage systems, a user may hold attributes issued by multiple authorities and the owner may share data with users administrated to different authorities. For instance, in an E-health system, the medical data may be shared 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. Some CP-ABE schemes [7]–[10] have been proposed for such multiauthority systems. However, due to the *inefficiency of computation*, they cannot be directly applied to construct the data access control scheme. Basically, there are two operations in access control that require efficient computation, namely *decryption* and *revocation*.

Revocation Efficiency: Data access in cloud storage systems is not static, as employees are hired/fired or promoted/demoted, it will be necessary to change the attributes of users. To guarantee the security of attribute revocation, there are two requirements: 1) *Backward Security*: The revoked user (whose attributes are revoked) cannot decrypt new ciphertexts that require the revoked attributes for decryption; 2) *Forward Security*: The newly joined users who have sufficient attributes are also able to decrypt the previously published ciphertexts. To achieve these two requirements, a trivial method is to re-encrypt all the data. But it incurs a high computation overhead as the amount of data is massive. This motivates us to develop a new

Manuscript received January 17, 2013; revised May 12, 2013 and August 06, 2013; accepted August 16, 2013. Date of publication August 23, 2013; date of current version October 09, 2013. This work was supported in part by the Research Grants Council of Hong Kong under Project CityU 114112, and in part by the U.S. National Science Foundation under Grants CNS-1262277 and CNS-1116939. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Jiangtao Li.

K. Yang is with the Department of Computer Science, University of Science and Technology of China, Hefei 230027, China, and also with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong (e-mail: kan.yang@my.cityu.edu.hk).

X. Jia, B. Zhang, and R. Xie are with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong (e-mail: csjia@cityu.edu.hk; Bo.Zhang@my.cityu.edu.hk; ruitaoxie2-c@my.cityu.edu.hk).

K. Ren is with the Department of Computer Science and Engineering, University at Buffalo, The Sate University of New York, Buffalo, NY 14260 USA (e-mail: kuiren@buffalo.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2013.2279531

method that can efficiently deal with the attribute revocation of users.

Decryption Efficiency: In CP-ABE systems, the users need to decrypt the data by using their secret keys. However, nowadays, users usually use their mobile devices (e.g., smart phones, tablets etc.) to access the cloud data, and the computation abilities of mobile devices are not as powerful as the one of PCs. This motivates us to outsource the main computation of decryption into the cloud server, while still keep the data privacy against the cloud server.

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

1) We propose DAC-MACS (Data Access Control for Multiauthority Cloud Storage), an effective and secure data access control scheme for multiauthority cloud storage systems, which is secure in the random oracle model and has better performance than existing schemes.
2) We construct a new multiauthority 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 multiauthority CP-ABE scheme that achieves both forward security and backward security. Moreover, it incurs less communication cost and computation cost during the attribute revocation.

Compared with the previous conference version [11], we highly improve the security of DAC-MACS and make it more practical for multiauthority cloud storage systems. Specifically, we mainly address the security weakness caused by the collusion between nonrevoked users and the corrupted AA. We first give a straightforward solution by making a security assumption that all the nonrevoked users will not send their received update keys to the revoked user. We further remove this assumption and propose an extensive data access control scheme (EDAC-MACS) that can achieve the same security goal. In EDAC-MACS, the revoked user will not be able to get illegal data access even with the help of any corrupted AA and nonrevoked user. Without such assumption, EDAC-MACS is more practical to be implemented in multiauthority cloud storage systems. We also provide the security analysis of EDAC-MACS and prove that it is secure under weaker security assumptions.

The remaining of this paper is organized as follows. We first define the system model, framework and security model in Section II. Then, we propose a new multiauthority CP-ABE scheme with efficient decryption and revocation, and then apply it to construct DAC-MACS in Section III. In Section IV, we analyze DAC-MACS in terms of both the security and the performance. We further extended our DAC-MACS to be secure under weaker assumptions in Section V. Section VI gives the related work. Finally, the conclusion is given in Section VII and the detailed security proof is described in the Appendix.

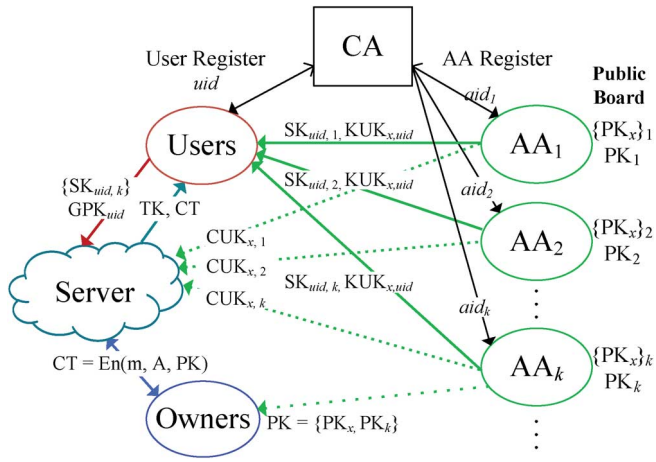


Fig. 1. System model of DAC-MACS.

II. System Model and Security Model

A. System Model

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), attribute authorities (AAs), cloud server (server), data owners (owners) and data consumers (users).

CA. 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, the CA assigns a global unique user identity to it and also generates a global secret/public key pair for this user. However, the CA is *not* involved in any attribute management and any generation of secret keys that are associated with attributes.

AA. Every AA is an independent attribute authority that is responsible for issuing, revoking and updating user's attributes according to its role or identity. Each AA is responsible for generating a public attribute key for each attribute it manages and a secret key for each user reflecting their attributes.

Server. The cloud server stores owners' data and provides data access service to users. It also helps users decrypt ciphertexts by generating decryption tokens and helps owners update ciphertexts when an attribute revocation happens.

Owners. Before outsourcing the data, each owner first encrypts the data with content keys by using symmetric encryption techniques. Then, the owner defines the access policies over attributes from multiple AAs and encrypts content keys under the policies. They do not trust on the server to do data access control. Instead, they assume that the server may give the data to all the 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 is able to decrypt the ciphertext.

Users. Each user is assigned with a global user identity from the CA and can freely query ciphertexts from the server. To decrypt a ciphertext, each user may submit their secret keys issued by some AAs together with its global public key to the server and ask for a decryption token. The user then uses the received decryption token to decrypt the ciphertext along with 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 further decryption token generation.

B. Framework

Definition 1 (DAC-MACS): The framework of DAC-MACS contains the following phases:

Phase 1: System Initialization: This phase consists of the following algorithms:

- $\mathbf{CASetup}(1^\lambda) \rightarrow (\mathsf{MSK}, \mathsf{SP}, (sk_{CA}, vk_{CA}))$. The CA setup algorithm takes no input other than the implicit security parameter λ. It outputs the master key MSK, the system parameter SP, a pair of signature and verification key (sk_{CA}, vk_{CA}) of the CA.
- $\mathbf{UserReg}(\mathsf{SP}, sk_{CA}, Info_u) \rightarrow (uid, \mathsf{GPK}_{uid}, \mathsf{GSK}_{uid}, Cert(uid))$. The user registration algorithm takes the system parameter SP, the CA's signature key sk_{CA} and the user information $Info_u$ (e.g., name, birthday etc.) as inputs. It authenticates the user and assigns a global unique user identity uid to the user. It outputs the user identity uid, a pair of global public/secret key $(\mathsf{GPK}_{uid}, \mathsf{GSK}_{uid})$ and a certificate $Cert(uid)$ which is signed by the CA.
- $\mathbf{AAReg}(Info_{AA}) \rightarrow (aid)$. The attribute authority registration algorithm takes the information of an attribute authority $Info_{AA}$ as input. It authenticates the AA and outputs a global authority identity aid for this AA.
- $\mathbf{AASetup}(\mathsf{SP}, aid) \rightarrow (\mathsf{SK}_{aid}, \mathsf{PK}_{aid}, \{\mathsf{VK}_{x_{aid}}, \mathsf{PK}_{x_{aid}}\})$. The attribute authority setup algorithm takes the system parameter SP and the global authority identity aid as inputs. It outputs a pair of secret/public authority key $(\mathsf{SK}_{aid}, \mathsf{PK}_{aid})$, the set of version keys and public attribute keys $\{\mathsf{VK}_{x_{aid}}, \mathsf{PK}_{x_{aid}}\}$ for each attributes x.

Phase 2: Secret Key Generation:

- $\mathbf{SKeyGen}(\mathsf{SK}_{aid}, \mathsf{SP}, \{\mathsf{PK}_{x_{aid}}\}, S_{uid,aid}, Cert(uid)) \rightarrow \mathsf{SK}_{uid,aid}$. The secret key generation algorithm takes as inputs the secret authority key SK_{aid}, the system parameter SP, the set of public attribute keys $\{\mathsf{PK}_{x_{aid}}\}$, a set of attributes $S_{uid,aid}$ that describes the secret key, and the certificate of user uid. It outputs a secret key $\mathsf{SK}_{uid,aid}$ for the user uid.

Phase 3: Data Encryption:

- $\mathbf{Encrypt}(\mathsf{SP}, \{\mathsf{PK}_k\}_{k\in I_A}, \{\mathsf{PK}_{x_k}\}_{x_k\in S_{A_k}}^{k\in I_A}, m, \mathbb{A}) \rightarrow \mathsf{CT}$. The encryption algorithm takes as inputs the system parameter SP, 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 $\{\mathsf{PK}_{x_k}\}_{x_k\in S_{A_k}}^{k\in I_A}$, the data m and an access structure $\mathbb{A}$ over all the selected attributes from the involved AAs. The algorithm first encrypts the data m by using symmetric encryption methods with a content key κ. Then, it encrypts the content key κ under the access structure $\mathbb{A}$ and outputs a ciphertext CT. We will assume that the ciphertext implicitly contains the access structure $\mathbb{A}$.

Phase 4: Data Decryption: The data decryption phase consists of Decryption Token Generation by cloud servers and Data Decryption by users with the following algorithms:

- $\mathbf{TKGen}(\mathsf{CT}, \mathsf{GPK}_{uid}, \{\mathsf{SK}_{uid,k}\}_{k\in I_A}) \rightarrow \mathsf{TK}$. The decryption token generation algorithm takes as inputs 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 user uid holds sufficient attributes that satisfy the access structure $\mathbb{A}$, the algorithm can successfully compute the correct decryption token TK for the ciphertext CT.
- $\mathbf{Decrypt}(\mathsf{CT}, \mathsf{TK}, \mathsf{GSK}_{uid}) \rightarrow m$. The decryption algorithm takes as inputs the ciphertext CT, the decryption token TK and the user's global secret key GSK_{uid}. It first decrypts the content key and further uses the content key to decrypt the data. It outputs the data m.

Phase 5: Attribute Revocation: This phase contains three steps: Update Key Generation by AAs, Secret Key Update by Nonrevoked Users[1] and Ciphertext Update by Servers.

- $\mathbf{UKeyGen}(\mathsf{SK}_{aid}, \{u_j\}, \mathsf{VK}_{\tilde{x}_{aid}}) \rightarrow (\mathsf{KUK}_{j,\tilde{x}_{aid}}, \mathsf{CUK}_{\tilde{x}_{aid}})$. The update key generation algorithm takes as inputs the secret authority 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's Key Update Key $\mathsf{KUK}_{j,\tilde{x}_{aid}} (j \in S_U, j \neq \mu, \tilde{x}_{aid} \in S_{j,aid})$ and the Ciphertext Update Key $\mathsf{CUK}_{\tilde{x}_{aid}}$.
- $\mathbf{SKUpdate}(\mathsf{SK}_{uid,aid}, \mathsf{KUK}_{uid,\tilde{x}_{aid}}) \rightarrow \mathsf{SK}'_{uid,aid}$. The user's secret key update algorithm takes as inputs the current secret key $\mathsf{SK}_{uid,aid}$ and its key update key $\mathsf{KUK}_{uid,\tilde{x}_{aid}}$. It outputs a new secret key $\mathsf{SK}'_{uid,aid}$.
- $\mathbf{CTUpdate}(\mathsf{CT}, \mathsf{CUK}_{\tilde{x}_{aid}}) \rightarrow \mathsf{CT}'$. 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'.

C. Security Assumption of Each Entity

In DAC-MACS, we have the following assumptions:

- The CA is trusted, but it is not allowed to decrypt any ciphertexts.
- Each AA is also trusted, but it can be corrupted by the adversary.
- The server is semitrusted (curious but honest). It will not deny service to any authorized users, and will correctly execute the tasks assigned by the AA. But it is curious about the data content or the received messages.
- Users are dishonest and may collude to obtain unauthorized access to data.
- All the nonrevoked users will not give the received update keys to the revoked user.[2]

[1]Users who hold the revoked attribute but have not been revoked.

[2]We will remove this assumption in the extensive data access control scheme (EDAC-MACS) in Section V.

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 [5] as follows. Let $a, s, b_1, \cdots, b_q \in \mathbb{Z}_p$ be chosen at random and g be a generator of $\mathbb{G}$. If an adversary is given

$$\vec{y} = \Big(g, g^s, g^{\frac{1}{z}}, g^{\frac{a}{z}}, \cdots, g^{\left(\frac{a^q}{z}\right)}, g^a, \cdots, g^{(a^q)}, , g^{(a^{q+2})}, \cdots, g^{(a^{2q})}, \forall_{1\le j\le q}\ g^{s\cdot b_j}, g^{\frac{a}{b_j}}, \cdots, g^{\left(\frac{a^q}{b_j}\right)}, , g^{\left(\frac{a^{q+2}}{b_j}\right)}, \cdots, g^{\left(\frac{a^{2q}}{b_j}\right)}, \forall_{1\le j,k\le q, k\ne j}\ g^{a\cdot s\cdot b_k/b_j}, \cdots, g^{(a^q\cdot s\cdot b_k/b_j)}\Big),$$

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

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

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

E. Security Model

We now describe the security model of DAC-MACS by the following game between a challenger and an adversary. The security model allows the adversary to query for any secret keys and update keys that cannot be used to decrypt the challenge ciphertext. Similar to [10], we assume that the adversaries can corrupt authorities only statically, 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 system parameters are generated by running the CA setup algorithm. The adversary specifies a set of corrupted attribute authorities $S'_A \subset S_A$. The challenger generates the public keys by querying the AA setup oracle, and generates the secret keys by querying the secret key generation oracle. 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 public keys and secret keys to the adversary.

Phase 1: The adversary makes secret key queries by submitting pairs (uid, S_{uid}) to the challenger, where $S_{uid} = \{S_{uid,k}\}_{k\in S_A - S'_A}$ is a set of attributes belonging to several uncorrupted AAs. The challenger gives the corresponding secret keys $\{\mathsf{SK}_{uid,k}\}$ 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. We let V denote the subset of rows of M^* labeled by attributes controlled by corrupted AAs. 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,. . .,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 b, and encrypts m_b 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)[3] 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 b' of b.

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

Definition 3: DAC-MACS is secure against static corruption of authorities if all polynomial time adversaries have at most a negligible advantage in the above security game.

Definition 4: DAC-MACS is collusion resilience if no polynomial time adversaries can decrypt the data by combining attributes of different users together, when each individual user cannot decrypt the data only with its own attributes.

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

This section first gives an overview of our scheme. Then, we describe DAC-MACS which consists of five phases.

A. Overview

Although the existing multiauthority CP-ABE scheme [10] proposed by Lewko and Waters has high policy expressiveness and has been extended to support attribute revocation in [12], it still cannot be applied to access control for multiauthority cloud storage systems due to the inefficiency of decryption and revocation. Thus, the main challenge is to construct a new underlying multiauthority CP-ABE scheme that supports efficient decryption and revocation.

To design a multiauthority CP-ABE scheme, the most challenging issue is how to tie different secret keys together but still prevent the collusion attack. Similar to [7], in DAC-MACS, we separate the authority into a global certificate authority (CA) and multiple attribute authorities (AAs). The CA sets up the system and assigns a global user identity uid to each user and a global authority identity aid to each attribute authority. The global unique uid can tie secret keys issued by different AAs together for decryption, and the global unique aid can distinguish attributes issued by different AAs. Thus, by using uid and aid, the collusion attack can be resisted. However, different from [7], the CA in DAC-MACS is *not* involved in any attribute management and the creation of secret keys reflecting the user's attributes. DAC-MACS also requires all the AAs to generate their own public keys which can be used to encrypt data together with the global public parameters, instead of only using the system unique public key for data encryption. This solves the security drawback in [7], i.e., it prevents the CA from decrypting the ciphertexts.

[3]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.

To achieve efficient decryption on the user, we propose a token-based decryption outsourcing method. We apply the decryption outsourcing idea from [12] 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 AAs to the server and asks the server to compute a decryption token for the ciphertext. The user can 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 for each attribute revocation, only those components associated with the revoked attribute in secret keys and ciphertexts need to be updated. When an attribute is revoked from a user, the corresponding AA will generate a new version key for this revoked attribute, and computes an update key containing a Ciphertext Update Key (CUK) and several user's Key Update Keys (KUKs). With the KUKs, each nonrevoked user can update its secret key to the current version, while the revoked user cannot update its secret key even using other users' update keys, since each KUK is associated with the uid (Backward Security). The ciphertexts can also be updated to the current version with the CUK, such that the newly joined users who have sufficient attribute are also able to decrypt the previous published data (Forward Security). Moreover, all the users only need to hold the latest secret key, rather than all the previous secret keys. To improve the efficiency, we delegate the workload of ciphertext update to the server by using the proxy re-encryption method.

B. System Initialization

This phase consists two steps: CA Setup and AA Setup.

1) CA Setup: 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 $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be the bilinear map. Let g be the generator of $\mathbb{G}$. Let $H : \{0,1\}^* \rightarrow \mathbb{G}$ be a hash function such that the security is in the random oracle.

The CA runs the CA setup algorithm, which takes a security parameter as input. The CA first generates a pair of signature and verification key (sk_{CA}, vk_{CA}). Then, it chooses a random number $a \in \mathbb{Z}_p$ as the master key MSK of the system and compute the system parameter as

$$\mathsf{SP} = (g, g^a, G, G_T, H).$$

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

- *User Registration*
  Every user should register itself to the CA during the system initialization. The CA runs the user registration algorithm $\mathsf{UserReg}$ which takes the system parameter and the user information as inputs. If the user is legal in the system, it assigns a global user identity uid to this user, and 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 $Cert(uid)$ which contains an item $Sign_{sk_{CA}}(uid, u_{uid}, g^{1/z_{uid}})$. Then, the CA sends the global public/secret key pair $(\mathsf{GPK}_{uid}, \mathsf{GSK}_{uid})$ and the certificate $Cert(uid)$ to user uid.
- *AA Registration*
  Each AA should also register itself to the CA during the system initialization. The CA runs the AA registration algorithm AAReg by taking the information of AA as input. If the AA is a legal authority in the system, the CA first assigns a global authority identity aid to it. Then, the CA sends both its verification key vk_{CA} and the system parameter SP to this AA.

2) AA Setup: Each $AA_k (k \in S_A)$ runs the AA 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 secret authority 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 key as $\mathsf{VK}_{x_k} = v_{x_k}$. The AA_k also computes the public authority key as

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

All the public attribute keys and public authority keys are published on the public bulletin board of AA_k.

C. Secret Key Generation by AAs

For every 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 its certificate by using the verification key vk_{CA}. If the user is not legal, it aborts. Otherwise, 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 runs the secret key generation algorithm $\mathsf{SKeyGen}$ to generate the user's secret key $SK_{j,k}$.

The algorithm takes as inputs the secret authority key SK_{aid}, the system parameter SP, the set of public attribute keys $\{\mathsf{PK}_{x_{aid}}\}$, a set of attributes $S_{uid,aid}$ that describes the secret key, and the certificate of user uid. It chooses a random number $t_{j,k} \in \mathbb{Z}_p$ and computes $SK_{j,k}$ as

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

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

D. Data Encryption by Owners

Before outsourcing data m into the cloud, the owner encrypts the data by running the data encryption algorithm $\mathsf{Encrypt}$. It takes as inputs the system parameter SP, 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 $\{\mathsf{PK}_{x_k}\}_{x_k \in S_{A_k}}^{k \in I_A}$, the data m and an access structure (M, ρ) over all the selected attributes from the involved AAs. Let M be a $l \times n$ matrix, where l denotes the total number of all the attributes. The function ρ associates rows of M to attributes.

The algorithm first divides the data into several data components as $m = \{m_1, \cdots, m_n\}$ according to the logic granularities. For example, the personal data may be divided into {name, address, security number, employer, salary}. It then encrypts data components with different symmetric content keys $\{\kappa_1, \cdots, \kappa_n\}$ by using symmetric encryption methods, where κ_i is used to encrypt $m_i (i = 1, \cdots, n)$.

Then, it defines an access structure M_i and encrypts each content key $\kappa_i (i = 1, \cdots, n)$ under this access structure. For simplicity, the rest of this paper only considers one component m and one content key κ. The encryption algorithm 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 $r_1, r_2, \cdots, r_l \in \mathbb{Z}_p$ and computes the ciphertext as

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

In real systems, if the data m is divided into n components, the ciphertext CT also consists of n components $\mathsf{CT} = \{\mathsf{CT}_1, \cdots, \mathsf{CT}_n\}$.

E. Data Decryption by Users (With the Help of Cloud)

All the legal users in the system can freely query any interested ciphertexts from the cloud server. But only when the user's attributes satisfy the access structure embedded in the ciphertext, he/she is able to decrypt the content keys and further use them to decrypt the data. This phase consists of two steps: *Token Generation by Cloud Server* and *Data Decryption by Users*

1) Token Generation by Cloud Server: 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 for a decryption token for the ciphertext CT. Only when the attributes the user U_j possesses satisfy the access structure defined in the ciphertext CT, the server can successfully compute the correct decryption token TK.

The server runs the token generation algorithm TKGen, which takes as inputs the ciphertext CT (which contains an access structure $\mathbb{A}$), user's global public key GPK_j and a set of user's secret keys $\{SK_{j,k}\}_{k \in I_A}$. Let $I = \{I_{A_k}\}_{k \in I_A}$ be the whole index set of all the attributes involved in the ciphertext, where $I_{A_k} \subset \{1, \cdots, l\}$ is the index subset of the attributes from the AA_k, defined as $I_{A_k} = \{i : \rho(i) \in S_{A_k}\}$. Let $N_A = |I_A|$ be the number of AAs 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 algorithm computes the decryption token TK as

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

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

2) Data Decryption by Users: Upon receiving this decryption token TK, the user U_j can use it to decrypt the ciphertext together with its global secret key $\mathsf{GSK}_j = z_j$ as

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

Then, the user can use the content key κ to further decrypt the data as

$$m = Dec_\kappa \left(Enc_\kappa(m) \right).$$

F. 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 by AAs*, *Secret Key Update by Nonrevoked Users* and *Ciphertext Update by Cloud Server*. The secret key update can prevent the revoked user from decrypting the new ciphertexts which are encrypted by the new public attribute keys (Backward Security). The ciphertext update can also guarantee that the newly joined user who have sufficient attributes can still access the previous published data (Forward Security).

1) Update Key Generation by $\mathbf{AA}$*s:* The corresponding authority AA_k runs the update key generation algorithm $\mathsf{UKeyGen}$ to compute the update keys. The algorithm takes as inputs the secret authority key SK_k, the current attribute version key $v_{\tilde{x}_k}$ and the user's global public keys GPK_j. It generates a new attribute version key $\mathsf{VK}'_{\tilde{x}_k} = v'_{\tilde{x}_k}$. It first calculates the Attribute Update Key as $\mathsf{AUK}_{\tilde{x}_k} = \gamma_k (v'_{\tilde{x}_k} - v_{\tilde{x}_k})$, and then applies it to compute the user's Key Update Key as

$$\mathsf{KUK}_{j,\tilde{x}_k} = g^{u_j \beta_k \cdot \mathsf{AUK}_{\tilde{x}_k}}$$

and the Ciphertext Update Key as

$$\mathsf{CUK}_{\tilde{x}_k} = \beta_k \cdot \mathsf{AUK}_{\tilde{x}_k} / \gamma_k.$$

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 $\tilde{x}_k$ is updated. Then, all the owners can get the new public attribute key from the public board of AA_k.

2) Secret Key Update by Nonrevoked Users: For each nonrevoked user $U_j (j \in S_U, j \neq \mu)$ who holds the revoked attribute $\tilde{x}_k$, the AA_k sends the corresponding user's key update key $\mathsf{KUK}_{j,\tilde{x}_k}$ to it. Upon receiving $\mathsf{KUK}_{j,\tilde{x}_k}$, the user U_j runs the key update algorithm $\mathsf{SKUpdate}$ to update its secret key as

$$\mathsf{SK}'_{j,k} = (K'_{j,k} = K_{j,k}, L'_{j,k} = L_{j,k}, R'_{j,k} = R_{j,k},$$
$$K'_{j,\tilde{x}_k} = K_{j,\tilde{x}_k} \cdot \mathsf{KUK}_{j,\tilde{x}_k}, \; \forall x \in S_u, x \neq \tilde{x} : K'_{j,k} = K_{j,k}).$$

Note that each $\mathsf{KUK}_{j,\tilde{x}_k}$ is associated with the uid, so that they are distinguishable for different nonrevoked users. Thus, the revoked user U_μ cannot use any other user's update keys to update its secret key.

3) Ciphertext Update by Cloud Server: 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 $\mathsf{CTUpdate}$ to update all the ciphertexts which are associated

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

Scheme	Authority	Computation		Revocation	Revocation Security		Revocation	Ciphertext
		Encrypt	Decrypt*	Message ($\|p\|$)	Backward	Forward	Controller	Updater
Hur's [13]	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 [14]	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 semitrusted.

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 several components of the ciphertext, which are associated with the $\tilde{x}_k$. The new ciphertext CT' is published as

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

DAC-MACS requires to update only a few components which are associated with the revoked attribute, while the other components are not changed. This can greatly improve the efficiency of attribute revocation.

The ciphertext update not only can guarantee the forward security of the attribute revocation, but also can reduce the storage overhead on users (i.e., all the users only need to hold the latest secret key, rather than to keep records on all the previous secret keys).

IV. ANALYSIS OF DAC-MACS

This section provides a comprehensive analysis of DAC-MACS, followed by security 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 nonrevoked 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, all of which relied on the ciphertext re-encryption to achieve the attribute revocation. 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 semitrusted server, which can greatly reduce the workload on owners. For the security of attribute revocation, DAC-MACS can achieve both forward security and backward security. The cloud server in our system is required to be semitrusted. Even if the cloud server is not semitrusted 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, i.e., the revoked user cannot decrypt new ciphertexts that requires the revoked attributes for decryption.

B. Security Analysis

Under the security model we defined in Section II, we prove that DAC-MACS is provable secure and collusion resilience, as concluded in the following theorems:

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

Proof: Suppose we have an adversary $\mathcal{A}$ with nonnegligible advantage $\epsilon = Adv_{\mathcal{A}}$ 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 and update 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 multiauthority 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 nonnegligible advantage. The detailed proof is described in the full version of this paper [15].■

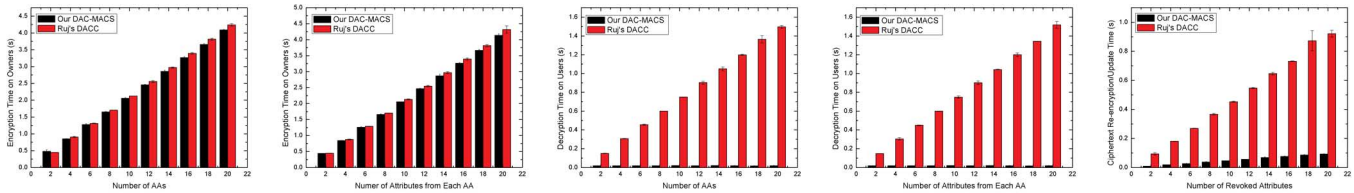
Theorem 2: DAC-MACS is secure against the collusion attack of users.

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 AAs are associated with 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 AAs 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 Ruj's DACC scheme under the metrics



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

Fig. 2. Comparison of encryption, decryption, and ciphertext re-encryption/update time.

TABLE II
COMPARISON OF STORAGE OVERHEAD

Entity	DACC [14]	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	$(n_{c,x}+\sum_{k=1}^{N_A} n_{a,k,uid})\|p\|$	$(3N_A+1+\sum_{k=1}^{N_A} n_{a,k,uid})\|p\|$
Server	$(3t_c+1)\|p\|$	$(3t_c+3)\|p\|$

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

TABLE III
COMPARISON OF COMMUNICATION COST FOR ATTRIBUTE REVOCATION

Operation	DACC [14]	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 nonrevoked users holding x;
$n_{c,x}$ is the number of ciphertexts containing x.

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 AAs 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 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 secret authority key, while in DACC it consists of secret keys for all the attributes. The public parameters contribute the main storage overhead on the owner. Besides, DACC also requires the owner to hold the encryption secret for every ciphertext in the system, because the owner is required to re-encrypt the ciphertexts. This incurs a heavy storage overhead on the owner, especially when the number of ciphertext is large in the system. 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 AAs. However, in DACC, 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. That is because when the ciphertext is re-encrypted, some of its components related to the revoked attributes should be sent to each nonrevoked user who holds the revoked attributes. The ciphertexts contribute the main storage overhead on the server (here we do not consider the component of data encrypted by symmetric content keys).

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.

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

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 on the owner versus the number of AAs, where the involved number of attributes from each AA is set to be 10. Fig. 2(b) gives the comparison of encryption time on the owner versus the number of attributes from each AA, where the involved number of AAs 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 on the user versus the number of AAs, where the number of attributes the user holds from each AA is set to be 10. Fig. 2(d) describes the comparison of decryption time on the user versus the number of attributes the user holds from each AA, where 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. EXTENSIVE DAC-MACS

In DAC-MACS, there is a security assumption that all the nonrevoked users will not give the received update keys to the revoked user. However, this is a strong assumption, and in practical the revoked user may collude with other users to obtain the update keys. This section first proposes an extensive data access control scheme (EDAC-MACS), and then give the security

analysis to show that EDAC-MACS can achieve the same security goal without this assumption.

A. EDAC-MACS

If we remove this assumption, the backward security in DAC-MACS will no longer be guaranteed. That is when the adversary μ (the revoked user) corrupted any AA, he/she could obtain all the users' secrets $\{u_{uid}\}$, and use it to transfer the other user's key update key $\mathsf{KUK}_{j,\tilde{x}_k}$ to its own one as

$$\mathsf{KUK}_{\mathcal{A},\tilde{x}_k} = (\mathsf{KUK}_{j,\tilde{x}_k})^{\frac{u_\mu}{u_j}}.$$

Then, the adversary can use it to update his secret key to the latest version by running the secret key update algorithm $\mathsf{SKUpdate}$.

To address this security issue, we modify the secret key generation algorithm $\mathsf{SKeyGen}$ by padding a new piece to the K_{j,x_k}. It generates the user's secret key as

$$\mathsf{SK}_{j,k} = \Bigg(K_{j,k} = g^{\frac{\alpha_k}{z_j}} \cdot g^{au_j} \cdot g^{\frac{a}{\beta_k} t_{j,k}}, L_{j,k} = g^{\frac{\beta_k}{z_j} t_{j,k}}, R_{j,k} = g^{a t_{j,k}}$$
$$\forall x_k \in S_{j,k} : K_{j,x_k} = g^{\frac{\beta_k \gamma_k t_{j,k}}{z_j}} \cdot \left(g^{v_{x_k}} \cdot H(x_k)\right)^{\left(\gamma_k \beta_k u_j + \gamma_k^2\right)} \Bigg),$$

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

The encryption algorithm $\mathsf{Encrypt}$ is the same as DAC-MACS, but during the data decryption, the decryption token TK is generated by the new decryption token generation algorithm TKGen as

$$\mathsf{TK} = \prod_{k \in I_A} \frac{e(C', K_{j,k}) \cdot e(R_{j,k}, C'')^{-1}}{\prod_{i \in I_{A_k}} \left(e(C_i, \mathsf{GPK}_j) e\left(D_{1,i}, K_{j,\rho(i)}\right) e\left(D_{2,i}, L_{j,k} \mathsf{PK}_{\rho(i)}\right)\right)^{w_i N_A}}$$
$$= \prod_{k \in I_A} e(g,g)^{\frac{\alpha_k}{z_j} s}.$$

Correctness: We observe that

$$\prod_{k \in I_A} e\left(C', K_{j,k}\right) e\left(R_{j,k}^{-1}, C''\right) = \prod_{k \in I_A} e\left(g^s, g^{\frac{\alpha_k}{z_j}} g^{au_j} g^{\frac{a t_{j,k}}{\beta_k}}\right) \times e\left(g^{a t_{j,k}}, g^{-\frac{s}{\beta_k}}\right)$$
$$= e(g,g)^{s a u_j N_A} \cdot \prod_{k \in I_A} e(g,g)^{s \frac{\alpha_k}{z_j}}.$$

For each $i \in I$, suppose $\rho(i) \in S_k$, it computes

$$e(C_i, \mathsf{GPK}_j) \cdot e\left(D_{1,i}, K_{j,\rho(i)}\right) \cdot e\left(D_{2,i}, L_{j,k} \cdot \mathsf{PK}_{\rho(i)}\right)$$
$$= e\left(g^{a\lambda_i} \cdot \left(g^{v_{\rho(i)}} H\left(\rho(i)\right)\right)^{-\gamma_k r_i}, g^{u_j}\right) \cdot e\left(g^{\frac{r_i}{\beta_k}}, g^{\frac{\beta_k \gamma_k t_{j,k}}{z_j}} \cdot \left(g^{v_{\rho(i)}} H\left(\rho(i)\right)\right)^{\left(\gamma_k \beta_k u_j + \gamma_k^2\right)}\right) \cdot e\left(g^{-\frac{\gamma_k}{\beta_k r_i}}, g^{\frac{\beta_k t_{j,k}}{z_j}} \left(g^{v_{\rho(i)}} H\left(\rho(i)\right)\right)^{\gamma_k}\right)$$
$$= e(g,g)^{a u_j \lambda_i}.$$

Then, it computes

$$\prod_{k \in I_A} \prod_{i \in I_{A_k}} \left(e(g,g)^{a u_j \lambda_i}\right)^{w_i N_A} = e(g,g)^{a u_j N_A s}.$$

The decryption algorithm is the same as DAC-MACS.

During the attribute revocation, the authority also needs to first generate the update keys. The ciphertext update key $\mathsf{CUK}_{\tilde{x}_k}$ is the same as the one in DAC-MACS. However, the user's key update key is generated as

$$\mathsf{KUK}_{j,\tilde{x}_k} = g^{(u_j \beta_k + \gamma_k) \cdot \mathsf{AUK}_{\tilde{x}_k}}.$$

B. Security Analysis

We conclude the security of EDAC-MACS as the following two theorems:

Theorem 3: In EDAC-MACS, the revoked user has no chance to update its secret key, even if it can corrupt some AAs (not the AA corresponding to the revoked attribute) and collude some nonrevoked users.

Proof: In EDAC-MACS, each key update key is associated with the user's identity uid. And the item $\left(g^{v_{x_k}} H(x_k)\right)^{\gamma_k}$ in the secret key prevents users from updating their secret keys with the other user's update key, since γ_k is only known by the AA_k and kept secret to all the users. ∎

Theorem 4: When the decisional q-parallel BDHE assumption holds, no polynomial time adversary can selectively break EDAC-MACS with a challenge matrix of size $l^* \times n^*$, where $n^* < q$.

Proof: Based on the Theorem 3, the security proof of EDAC-MACS is similar to DAC-MACS as Theorem 1. The main difference is how to simulate the new secret keys in EDAC-MACS. The detailed security proof is described in Appendix A. ∎

VI. Related Work

Cryptographic techniques are well applied to access control for remote storage systems [16]–[18]. To prevent the untrusted servers from accessing sensitive data, traditional methods [19], [20] usually encrypt the data and only the users who hold valid keys can decrypt. Then, the data access control becomes the matter of key distribution. Although these methods can provide secure data access control, the key management is very complicated when more users are in the system. Data owners also have to stay online all the time to deliver keys to new users. Moreover, for each data, there are multiple copies of ciphertexts for users with different keys, which will incur high storage overhead on the server. Some methods [21] deliver the key management and distribution from data owners to the remote server under the assumption that the server is trusted. However, the server is not fully trusted in cloud storage systems and thus these methods cannot be applied to data 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 [22], Goyal *et al.* [23] 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 [13], [24], [25]. These schemes require a trusted authority to manage all the attributes in the system and issue secret keys to users. Since the authority can decrypt all the encrypted data, it becomes a vulnerable security point for the system. Moreover, the authority may become the performance bottleneck in the large scale cloud storage systems.

Some cryptographic methods are proposed for the multiauthority ABE problem [7]–[10], [26], [27], where there are multiple authorities coexist and users may have attributes from multiple authorities. However, some of them [7], [8] still require a global authority. Lin *et al.* [26] proposed a decentralized scheme based on threshold mechanism. In this scheme, the set of authorities is predetermined and it requires the interaction among the authorities during the system setup. In [10], Lewko *et al.* proposed a new comprehensive scheme, which does not require any central authority. However, they did not consider attribute revocation problem.

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.* [14] 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 nonrevoked user. Li *et al.* [28] proposed an attribute revocation method for multiauthority ABE systems, but their methods is only for KP-ABE systems.

Green *et al.* [12] 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 multiauthority 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.

VII. Conclusion

In this paper, we proposed an effective data access control scheme for multiauthority cloud storage systems, DAC-MACS. We also constructed a new multiauthority 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.

We also removed the security assumption that all the nonrevoked users will not reveal their received key update keys to the revoked user. We further proposed an extensive DAC-MACS, which is secure under weaker security assumptions. Although this work is for multiauthority cloud storage systems, the techniques designed in this paper can be applied into other applications, such as any remote storage systems, online social networks etc.

Appendix
Proof of Theorem 4

Proof: Suppose we have an adversary $\mathcal{A}$ with nonnegligible advantage $\epsilon = Adv_{\mathcal{A}}$ 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 and update 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 multiauthority 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 nonnegligible 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 $\mathsf{CASetup}$ and $\mathsf{AASetup}$ 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 $\alpha'_k, \beta_k, \gamma_k \in \mathbb{Z}_p (k \in S_A - S'_A)$ and implicitly sets $\alpha_k = \alpha'_k + a^{q+1}$ by letting

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

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, 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

$$H(x) = g^{d_x} \prod_{i \in X} g^{\frac{a^2 M^*_{i,1}}{b_i}} \cdot g^{\frac{a^3 M^*_{i,2}}{b_i}} \cdots g^{\frac{a^{n^*+1} M^*_{i,n}}{b_i}}.$$

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 generates 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

$$\mathsf{PK}_{x_k} = \left(g^{v_{x_k}+d_{x_k}} \prod_{i \in X} g^{\frac{a^2 M^*_{i,1}}{b_i}} \cdot g^{\frac{a^3 M^*_{i,2}}{b_i}} \cdots g^{\frac{a^{n+1} M^*_{i,n}}{b_i}}\right)^{\gamma_k}.$$

The simulator assigns a user identity uid to the adversary and chooses two random numbers $u'_{uid}, z_{uid} \in \mathbb{Z}_p$. Then, it sets $\mathsf{GSK}_{uid} = z_{uid}$ and implicitly sets $u_{uid} = u'_{uid} - (a^q / z_{uid})$ by setting

$$\mathsf{GPK}_{uid} = g^{u'_{uid}} \left(g^{a^q}\right)^{-\frac{1}{z_{uid}}}$$

The simulator then sends the global public/secret key pairs $(\mathsf{GPK}_{uid}, \mathsf{GSK}_{uid})$ to the adversary.

Phase 1: In this phase, the simulator answers secret key queries and update key queries from the adversary. Suppose the adversary makes secret key queries by submitting pairs (uid, S_k) to the simulator, where S_k is a set of attributes belonging to an uncorrupted authority AA_k. Suppose S_k does not satisfy M^* together with any keys that can obtain from corrupted authorities.

The simulator 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 t by randomly choosing a number $r \in \mathbb{Z}_p$ as

$$t_{uid,k} = r + w_1 a^{q-1} + w_2 a^{q-2} + \cdots + w_{n^*} a^{q-n^*}$$

by setting

$$L_{uid,k} = \left(g^{\frac{\beta_k}{z_{uid}}}\right)^r \prod_{i=1,\ldots,n^*} \left(g^{a^{q-i}}\right)^{w_i \frac{\beta_k}{z_{uid}}}.$$

The simulator then constructs $R_{uid,k}$ as

$$R_{uid,k} = g^{ar} \cdot \prod_{i=1,\ldots,n^*} \left(g^{a^{q+1-i}}\right)^{w_i}.$$

From the definition of $g^{u_{uid}}$, we find that $g^{au_{uid}}$ contains a term of $g^{a^{q+1}/z_{uid}}$, which will cancel out with the unknown term in $g^{\alpha_k/z_{uid}}$ when creating $K_{uid,k}$. The simulator can calculate

$$K_{uid,k} = g^{\frac{\alpha_k'}{z_{uid}}} g^{au_{uid}'} g^{\frac{ar}{\beta_k}} \cdot \prod_{i=1,\ldots,n^*} \left(g^{a^{q+1-i}}\right)^{\frac{w_i}{\beta_k}}.$$

In EDAC-MACS, a new piece is padded to the component $K_{x_k,uid,k}$ in the secret key. Fortunately, the new piece can be easily simulated as $(\mathsf{PK}_{x_k})^{\gamma_k}$. Thus, for the calculation of $K_{x_k,uid,k}(\forall x_k \in S_k)$, if x_k is used in the access structure, the simulator computes K_{x_k,uid,S_k} as follows.

$$\begin{aligned} K_{uid,x_k} &= (L_{uid,k})^{\gamma_k} \cdot (\mathsf{PK}_{x_k})^{\beta_k u_{uid}' + \gamma_k} \\ &\quad \cdot \left(g^{a^q}\right)^{-\beta_k \gamma_k \left(v_{x_k} + d_{x_k}\right)/z_{uid}} \\ &\quad \cdot \prod_{i \in X} \prod_{j=1,\ldots,n^*} \left(g^{\frac{a^{q+1+j}}{b_i}}\right)^{-\beta_k \gamma_k M_{i,j}^*} \end{aligned}$$

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

$$K_{uid,x_k} = (L_{uid,k})^{\gamma_k} \cdot (\mathsf{GPK}_{uid})^{\beta_k \gamma_k \left(v_{x_k} + d_{x_k}\right)} \cdot g^{\gamma_k^2 \left(v_{x_k} + d_{x_k}\right)}.$$

Towards update key queries, suppose the adversary submits pairs of $\{(uid, x_k)\}$. If the attribute x_k has a new version number v_{x_k}', and uid is an nonrevoked users, it then sends back the key update key as

$$\mathsf{KUK}_{uid,x_k} = g^{u_j \beta_k \gamma_k \left(v_{x_k}' - v_{x_k}\right)}.$$

Otherwise, it responses "$\perp$".

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

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

and $C' = g^s$, $C'' = g^{s/\beta_k}$.

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

$$\vec{v} = \left(s, sa + y_2', sa^2 + y_3', \cdots, sa^{n^*-1} + y_{n^*}'\right) \in \mathbb{Z}_p^{n^*}.$$

It also chooses random values $r_1', \cdots, r_l'$.

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

$$D_{1,i} = \left(g^{r_i'} g^{sb_i}\right)^{\frac{1}{\beta_k}}, \quad D_{2,i} = \left(g^{r_i'} g^{sb_i}\right)^{\frac{-\gamma_k}{\beta_k}}.$$

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

$$\lambda_i = s \cdot M_{i,1} + \sum_{j=2,\ldots,n^*} \left(sa^{j-1} + y_j'\right) M_{i,j}^*$$

Then, we can simulate the C_i as

$$\begin{aligned} C_i &= \left(g^{v_{\rho^*(i)}} \cdot H\left(\rho^*(i)\right)\right)^{\gamma_k r_i'} \cdot \left(\prod_{j=1,\ldots,n^*} g^{aM_{i,j} y_j}\right) \\ &\quad \cdot \left(g^{b_i s}\right)^{-\gamma_k \left(v_{\rho^*(i)} + d_{\rho^*(i)}\right)} \cdot \left(\prod_{k \in R_i} \prod_{j=1,\ldots,n^*} \left(g^{a^j s \left(\frac{b_i}{b_k}\right)}\right)^{\gamma_k M_{k,j}^*}\right). \end{aligned}$$

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 $Pr[\mathcal{B}(\vec{y}, T = e(g,g)^{a^{q+1}s}) = 0] = (1/2) + Adv_{\mathcal{A}}$. When T is a random group element the message m_b is completely hidden from the adversary and we have at $Pr[\mathcal{B}(\vec{y}, T = e(g,g)^{a^{q+1}s}) = 0] = 1/2$.

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

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 *Proc. S&P'07*, 2007, pp. 321–334, IEEE Computer Society.

[3] V. Goyal, A. Jain, O. Pandey, and A. Sahai, "Bounded ciphertext policy attribute based encryption," in *Proc. ICALP'08*, 2008, pp. 579–591, Springer.

[4] R. Ostrovsky, A. Sahai, and B. Waters, "Attribute-based encryption with non-monotonic access structures," in *Proc. CCS'07*, 2007, pp. 195–203, ACM.

[5] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proc. PKC'11*, 2011, pp. 53–70, Springer.

[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 *Proc. EUROCRYPT'10*, 2010, pp. 62–91, Springer.

[7] M. Chase, "Multi-authority attribute based encryption," in *Proc. TCC'07*, 2007, pp. 515–534, Springer.

[8] S. Müller, S. Katzenbeisser, and C. Eckert, "Distributed attribute-based encryption," in *Proc. 11th Int. Conf. Information Security and Cryptology*, 2008, pp. 20–36, Springer.

[9] M. Chase and S. S. M. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *Proc. CCS'09*, 2009, pp. 121–130, ACM.

[10] A. B. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proc. EUROCRYPT'11*, 2011, pp. 568–588, Springer.

[11] K. Yang, X. Jia, and K. Ren, "DAC-MACS: Effective Data Access Control for Multi-Authority Cloud Storage Systems," in *Proc. INFOCOM'13*, 2013, pp. 2995–3003, IEEE.

[12] M. Green, S. Hohenberger, and B. Waters, "Outsourcing the decryption of abe ciphertexts," in *Proc. 20th USENIX Security Symp.*, 2011, pp. 1–16, USENIX Association.

[13] 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, Jul. 2011.

[14] S. Ruj, A. Nayak, and I. Stojmenovic, "DACC: Distributed Access Control in Clouds," in *Proc. TrustCom'11*, 2011, pp. 91–98, IEEE.

[15] K. Yang, X. Jia, and K. Ren, "DAC-MACS: Effective Data Access Control for Multi-Authority Cloud Storage Systems," *IACR Cryptology ePrint Archive*, vol. 419, pp. 1–12, 2012.

[16] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Plutus: Scalable secure file sharing on untrusted storage," in *Proc. FAST'03*, San Francisco, CA, USA, 2003, pp. 29–42, USENIX.

[17] E.-J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "Sirius: Securing remote untrusted storage," in *Proc. NDSS'03*, 2003, pp. 131–145, The Internet Society.

[18] D. Naor, M. Naor, and J. Lotspiech, "Revocation and tracing schemes for stateless receivers," in *Proc. Electronic Colloquium on Computational Complexity (ECCC)*, 2002, pp. 41–62.

[19] J. Benaloh, M. Chase, E. Horvitz, and K. Lauter, "Patient controlled encryption: Ensuring privacy of electronic medical records," in *Proc. CCSW'09*, 2009, pp. 103–114, ACM.

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

[21] W. Wang, Z. Li, R. Owens, and B. K. Bhargava, "Secure and efficient access to outsourced data," in *Proc. CCSW'09*, 2009, pp. 55–66, ACM.

[22] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proc. EUROCRYPT'05*, 2005, pp. 457–473, Springer.

[23] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. CCS'06*, 2006, pp. 89–98, ACM.

[24] S. Yu, C. Wang, K. Ren, and W. Lou, "Attribute based data sharing with attribute revocation," in *Proc. ASIACCS'10*, 2010, pp. 261–270, ACM.

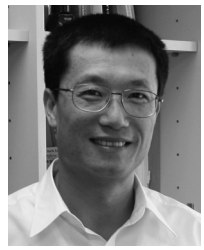
[25] S. Jahid, P. Mittal, and N. Borisov, "Easier: Encryption-based access control in social networks with efficient revocation," in *Proc. ASIACCS'11*, 2011, pp. 411–415, ACM.

[26] 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.

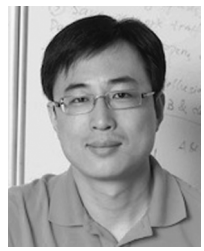
[27] J. Li, Q. Huang, X. Chen, S. S. M. Chow, D. S. Wong, and D. Xie, "Multi-authority ciphertext-policy attribute-based encryption with accountability," in *Proc. ASIACCS'11*, 2011, pp. 386–390, ACM.

[28] 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 Distrib. Syst.*, vol. 24, no. 1, pp. 131–143, Jan. 2013.

Kan Yang (S'10–A'13) received the B.Eng. degree from the University of Science and Technology of China, in 2008, and the Ph.D. degree from the City University of Hong Kong, in August 2013. He was a visiting scholar with the 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, distributed systems, etc.



Xiaohua Jia (A'00–SM'01–F'13) received the B.Sc. (1984) and M.Eng. (1987) degrees from the University of Science and Technology of China, and the D.Sc. (1991) degree in information science from the University of Tokyo. He is currently Chair Professor with 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. He 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 Cochair of IEEE MASS 2009, Area-Chair of IEEE INFOCOM 2010, TPC Cochair of IEEE GlobeCom 2010 Ad Hoc and Sensor Networking Symp, and Panel Cochair of IEEE INFOCOM 2011.



Kui Ren (A'07–M'07–SM'11) is currently an associate professor with the Computer Science and Engineering Department, State University of New York at Buffalo. In the past, he has been an associate/assistant professor with the Electrical and Computer Engineering Department, Illinois Institute of Technology. He received the Ph.D. degree from Worcester Polytechnic Institute. His research interests include cloud computing security, wireless and smartphone security, crowdsourcing systems, and smart grid security. His research has been supported by NSF, DoE, AFRL, and Amazon.

Dr. Ren is a recipient of National Science Foundation Faculty Early Career Development (CAREER) Award in 2011. He received the Best Paper Award from IEEE ICNP 2011. He serves as an Associate Editor for IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE WIRELESS COMMUNICATIONS, IEEE TRANSACTIONS ON SMART GRID, *Internet of Things Journal*, and *Journal of Communications and Networks*. He is a member of ACM and a past board member of Internet Privacy Task Force, State of Illinois.

Bo Zhang (S'10–M'13) received the B.Sc. (2009) degree in electronic information science and technology from Sun Yat-sen University, China, and the Ph.D. (2013) degree from City University of Hong Kong. He is currently a Postdoctoral Fellow with the Department of Computer Science, City University of Hong Kong. His research interests include wireless ad-hoc networks, collaborative relay networks, and MIMO systems.

Ruitao Xie (S'11) received the B.Eng. degree from Beijing University of Posts and Telecommunications in 2008. She is currently a Ph.D. candidate in the Department of Computer Science, City University of Hong Kong. Her research interests include wireless sensor networks, cloud computing, and distributed systems.