TMACS: A Robust and Verifiable Threshold Multi-Authority Access Control System in Public Cloud Storage

Wei Li, Kaiping Xue, Yingjie Xue, and Jianan Hong

Abstract—Attribute-based Encryption (ABE) is regarded as a promising cryptographic conducting tool to guarantee data owners' direct control over their data in public cloud storage. The earlier ABE schemes involve only one authority to maintain the whole attribute set, which can bring a single-point bottleneck on both security and performance. Subsequently, some multi-authority schemes are proposed, in which multiple authorities separately maintain disjoint attribute subsets. However, the single-point bottleneck problem remains unsolved. In this paper, from another perspective, we conduct a threshold multi-authority CP-ABE access control scheme for public cloud storage, named TMACS, in which multiple authorities jointly manage a uniform attribute set. In TMACS, taking advantage of (t, n) threshold secret sharing, the master key can be shared among multiple authorities, and a legal user can generate his/her secret key by interacting with any t authorities. Security and performance analysis results show that TMACS is not only verifiable secure when less than t authorities are compromised, but also robust when no less than t authorities are alive in the system. Furthermore, by efficiently combining the traditional multi-authority scheme with TMACS, we construct a hybrid one, which satisfies the scenario of attributes coming from different authorities as well as achieving security and system-level robustness.

Index Terms—CP-ABE, (*t,n*) threshold secret sharing, multi-authority, public cloud storage, access control

1 Introduction

To satisfy requirements of data storage and high performance computation, cloud computing has drawn extensive attentions from both academic and industry. Cloud storage is an important service of cloud computing [1], which provides services for data owners to outsource data to store in cloud via Internet.

Despite many advantages of cloud storage, there still remain various challenging obstacles, among which, privacy and security of users' data have become major issues, especially in public cloud storage [2], [3]. Traditionally, a data owner stores his/her data in trusted servers, which are generally controlled by a fully trusted administrator. However, in public cloud storage systems, the cloud is usually maintained and managed by a semi-trusted third party (the cloud provider). Data is no longer in data owner's trusted domains and the data owner cannot trust on the cloud server to conduct secure data access control. Therefore, the secure access control problem has become a critical challenging issue in public cloud storage, in which traditional security technologies cannot be directly applied.

Attribute-based Encryption (ABE) [4], [5], [6] is regarded as one of the most suitable schemes to conduct data access control in public clouds for it can guarantee data owners' direct control over their data and provide a fine-grained access control service. Till now, there are many ABE schemes proposed, which can be divided into two categories: Key-Policy Attribute-based Encryption (KP-ABE), such as [7], [8], and Ciphertext-Policy Attribute-based Encryption (CP-ABE), such as [9], [10], [11], [12]. In KP-ABE schemes, decrypt keys are associated with access structures while ciphertexts are only labeled with special attribute sets. On the contrary, in CP-ABE schemes, data owners can define an access policy for each file based on users' attributes, which can guarantee owners' more direct control over their data. Therefore, compared with KP-ABE, CP-ABE is a preferred choice for designing access control for public cloud storage.

In most existing CP-ABE schemes [9], [10], [11], [12], there is only one authority responsible for attribute management and key distribution. This only-one-authority scenario can bring a single-point bottleneck on both security and performance. Once the authority is compromised, an adversary can easily obtain the only-one-authority's master key, then he/she can generate private keys of any attribute subset to decrypt the specific encrypted data. Moreover, once the only-one-authority is crashed, the system completely cannot work well. Therefore, these CP-ABE schemes are still far from being widely used for access control in public cloud storage. Although some multi-authority CP-ABE schemes [13], [14], [15] have been proposed, they still cannot deal with the problem of single-point bottleneck on both security and performance mentioned above. In these multi-authority CP-ABE schemes, the whole attribute set is divided into multiple disjoint subsets and each attribute subset is still maintained by only one authority. Although the adversary cannot gain private keys of all attributes if he/she hasn't compromised all authorities, compromising one or more

• *The authors are with the Department of Electrical Engineering and Computer Science, University of Science and Technology of China, Hefei, China 230027.*
E-mail: {lwz159, xyj1108, hongjn}@mail.ustc.edu.cn, kpxue@ustc.edu.cn.

Manuscript received 5 Dec. 2014; revised 12 May 2015; accepted 16 June 2015. Date of publication 21 June 2015; date of current version 13 Apr. 2016. Recommended for acceptance by G. Wang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2448095

authorities would make the adversary have more privileges than he/she should have. Moreover, the adversary can obtain private keys of specific attributes by compromising specific one or more authorities. In addition, the single-point bottleneck on performance is not yet solved in these multi-authority CP-ABE schemes. Crash or offline of a specific authority will make that private keys of all attributes in attribute subset maintained by this authority cannot be generated and distributed, which will still influence the whole system's effective operation.

In this paper, we propose a robust and verifiable threshold multi-authority CP-ABE access control scheme, named TMACS, to deal with the single-point bottleneck on both security and performance in most existing schemes. In TMACS, multiple authorities jointly manage the whole attribute set but no one has full control of any specific attribute. Since in CP-ABE schemes, there is always a secret key (SK) used to generate attribute private keys, we introduce (t, n) threshold secret sharing [16] into our scheme to share the secret key among authorities. In TMACS, we redefine the secret key in the traditional CP-ABE schemes as master key. The introduction of (t, n) threshold secret sharing guarantees that the master key cannot be obtained by any authority alone. TMACS is not only verifiable secure when less than t authorities are compromised, but also robust when no less than t authorities are alive in the system. To the best of our knowledge, this paper is the first try to address the single-point bottleneck on both security and performance in CP-ABE access control schemes in public cloud storage.

Main contributions of this work can be summarized as follows:

- In existing access control systems for public cloud storage, there brings a single-point bottleneck on both security and performance against the single authority for any specific attribute. To the best of our knowledge, we are the first to design a multi-authority access control architecture to deal with the problem.
- By introducing the combining of (t, n) threshold secret sharing and multi-authority CP-ABE scheme, we propose and realize a robust and verifiable multi-authority access control system in public cloud storage, in which multiple authorities jointly manage a uniform attribute set.
- Furthermore, by efficiently combining the traditional multi-authority scheme with ours, we construct a hybrid one, which can satisfy the scenario of attributes coming from different authorities as well as achieving security and system-level robustness.

The rest of this paper is organized as follows. In Section 2, technical preliminaries are presented. Following the definitions of system model and security model in Section 3, we give our proposed multi-authority CP-ABE and relative access control scheme for public cloud storage in detail in Section 4. In Section 5, we analyze our proposed scheme in terms of security and performance. In Section 6, we briefly introduce the construction of a hybrid multi-authority system. We give some related work about ABE and some existing multi-authority extension schemes in cloud storage in Section 7. Finally, the conclusion is given in Section 8.

2 Preliminary

In this section, we first give a brief review of background information on bilinear maps and the security assumption defined on it, then we briefly describe (t, n) threshold secret sharing introduced in TMACS.

2.1 Bilinear Maps

Let $\mathbb{G}$, $\mathbb{G}_T$ be two multiplicative cyclic groups with the same prime order p and g be a generator of $\mathbb{G}$. A bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ defined on $\mathbb{G}$ has the following three properties:

1) *Bilinearity*: $\forall a, b \in \mathbb{Z}_p$ and $g_1, g_2 \in \mathbb{G}$, we have $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
2) *Non-degeneracy*: There exists $g_1, g_2 \in \mathbb{G}$ such that $e(g_1, g_2) \neq 1$, which means the map does not send all pairs in $\mathbb{G} \times \mathbb{G}$ to the identity in $\mathbb{G}_{\mathbb{T}}$.
3) *Computability*: There is an efficient algorithm to compute $e(g_1, g_2)$ for all $g_1, g_2 \in \mathbb{G}$.

Definition 1. *Decisional q-parallel Bilinear Diffie-Hellman Exponent Assumption(decisional q-BDHE): The decisional q-BDHE problem is that, in a group $\mathbb{G}$ of prime order p, give a, s, $b_1, b_2, \ldots, b_q \in \mathbb{Z}_p$, if an adversary is given:*

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

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

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

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

We say that the decisional q-BDHE assumption holds in $\mathbb{G}$ if no polynomial-time algorithm has a non-negligible advantage in solving the decisional q-BDHE problem.

2.2 (t, n) Threshold Secret Sharing

Secret sharing is a technique used to share a secret among a group of participants, each of whom is allocated partial information about the secret, which is named a share of the secret. The secret can be reconstructed only when a sufficient number of partial shares are combined together. Individual shares are of no use on their own. There are several types of secret sharing schemes, among which, the most basic types are the so-called (t, n) threshold schemes. After Shamir has proposed the first (t, n) threshold secret sharing [17], many more practical schemes [16], [18], [19] have been proposed. In TMACS, to avoid any entity being the security bottleneck, we adopt the scheme proposed by Pedersen [16], in which there is not any trusted third party.

Here, we give a simple description of (t, n) threshold secret sharing. Assume that there are n participants in the system, denoted as $P = \{P_1, P_2, \ldots, P_n\}$. And we set t $(t \leq n)$ as the threshold value. Define a finite field $K = GF(q)$. Each participant's identification in public can be

denoted as $x_1, x_2, \ldots, x_n \in GF(q)$, $(x_i \neq x_j$, when $i \neq j)$. First, each participant P_i selects a random element $S_i \in GF(q)$ as his/her sub-secret, then the master secret can be set as S=$\sum_{i=1}^{n} S_i$, which cannot be known by any participant. Each participant P_i separately generates a random polynomial $f_i(x)$ of degree $t-1$ that satisfies the formula $f_i(0) = S_i$. Subsequently, for each of the other $n-1$ participants P_j $(j = 1, 2, \ldots, i-1, i+1, \ldots, n)$, the participant P_i separately calculates sub-shares: $s_{ij} = f_i(x_j)$, and sends the sub-share s_{ij} to P_j securely. Meanwhile, P_i generates $s_{ii} = f_i(x_i)$ for himself/herself. After receiving messages from all of the other $n-1$ participants, each participant P_i obtains n sub-shares $s_{ji}, (j = 1, 2, \ldots, n)$. Thus, P_i can calculate his/her own master share as $s_i = \sum_{j=1}^{n} s_{ji} = \sum_{j=1}^{n} f_j(x_i)$. After that the sharing master secret S can be reconstructed by any t of the n participants. Assume that there is a function $F(x) = \sum_{j=1}^{n} f_j(x)$. For each participant's master share s_i, $i = 1, 2, \ldots, n$, we can know that $s_i = \sum_{j=1}^{n} s_{ji} = \sum_{j=1}^{n} f_j(x_i) = F(x_i)$. Therefore, $F(x)$ can be reconstructed using the Lagrange interpolating formula by any t participants' master shares, and the sharing master secret $S = F(0)$ can be calculated.

3 System Model and Security Model

In this section, we give the definitions of the system model and the security model in robust multi-authority public cloud storage systems.

3.1 System Model

In robust multi-authority public cloud storage systems, there exist five entities: a global *certificate authority (CA)*, multiple *attribute authorities (AAs)*, *data owners (Owners)*, *data consumers (Users)*, and *the cloud server*.

1) *The certificate authority* is a global trusted entity in the system that is responsible for the construction of the system by setting up system parameters and attribute public key (PK) of each attribute in the whole attribute set. *CA* accepts users and *AA*s' registration requests by assigning a unique uid for each legal user and a unique aid for each *AA*. *CA* also decides the parameter t about the threshold of *AA*s that are involved in users' secret key generation for each time. However, *CA* is not involved in *AA*s' master key sharing and users' secret key generation. Therefore, for example, *CA* can be government organizations or enterprise departments which are responsible for the registration [20], [21], [22], [23], [24], [25].
2) *The attribute authorities* focus on the task of attribute management and key generation. Besides, *AA*s take part of the responsibility to construct the system, and they can be the administrators or the managers of the application system. Different from other existing multi-authority CP-ABE systems, all *AA*s jointly manage the whole attribute set, however, any one of *AA*s cannot assign users' secret keys alone for the master key is shared by all *AA*s. All *AA*s cooperate with each other to share the master key. By this means, each *AA* can gain a piece of master key share as its private key, then each *AA* sends its corresponding public key to *CA* to generate one of the system public keys. When it comes to generate users' secret key, each *AA* only should generate its corresponding secret key independently. That is to say, no communication among *AA*s is needed in the phase of users' secret key generation.
3) *The data owner (Owner)* encrypts his/her file and defines access policy about who can get access to his/her data. First of all, each owner encrypts his/her data with a symmetric encryption algorithm like *AES* and *DES*. Then the owner formulates access policy over an attribute set and encrypts the symmetric key under the policy according to attribute public keys gained from *CA*. Here, the symmetric key is the key used in the former process of symmetric encryption. After that, the owner sends the whole encrypted data and the encrypted symmetric key to store in the cloud server. However, the owner doesn't rely on the cloud server to conduct data access control. Data stored in the cloud server can be gained by any data consumer. Despite all this, no data consumer can gain the plaintext without the attribute set satisfying the access policy.
4) *The data consumer (User)* is assigned with a global user identity uid from *CA*, and applies for his/her secret keys from *AA*s with his/her identification. The user can freely get the ciphertexts that he/she is interested in from the cloud server. He/She can decrypt the encrypted data if and only if his/her attribute set satisfies the access policy hidden inside the encrypted data.
5) *The cloud server* does nothing but provide a platform for owners storing and sharing their encrypted data. The cloud server doesn't conduct data access control for owners. The encrypted data stored in the cloud server can be downloaded freely by any data consumer.

3.2 Security Assumption and Security Model

In this section, we first give the security assumption of each entity in the robust multi-authority public cloud storage systems, then we define the corresponding security model.

3.2.1 Security Assumption

In multi-authority public cloud storage systems, the security assumption of the five roles is assumed as follows. The cloud server is always online and managed by the cloud provider. Usually, the cloud server and its provider is assumed "honest-but-curious". In actually using this model, there exist different assumptions about whether the cloud server can collude with the malicious users. Thus in order to eliminate the ambiguous, in this paper, we assume that the cloud server can still collude with some malicious users to gain the content of encrypted data when it is highly beneficial. Moreover, if treating this as a relative strong security assumption, the scheme meeting the security requirements can also apply to the scenario with a relative weak security assumption that the cloud server will never collude with malicious users.

CA is assumed to be trusted, but it can also be compromised by an adversary, so as to *AA*s. Although a user can freely get ciphertexts from the cloud server, he/she can't decrypt the encrypted data only unless the user's attributes satisfy the access policy hidden inside the encrypted data. Therefore, some malicious users are assumed to be dishonest and curious, who may collude with other entities except data owners(even compromising *AA*s) to obtain the access permission beyond their privileges. As a comparison, owners can be fully trusted.

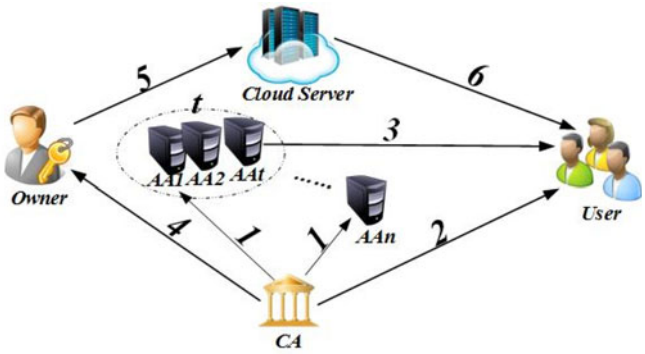
3.2.2 Security Model

Here, we introduce the universal security model in multi-authority public cloud storage systems, which can be divided into two phases. In the first phase, the malicious user (denoted as an adversary in the following) compromises *AA*s to gain *AA*s' master key. In the second phase, the adversary attempts to decrypt a ciphertext with the secret keys that can't satisfy the access policy inside the ciphertext. In this sub-process, the security model is defined similar to Waters's scheme [10]. In this security model, there is an adversary and a challenger. The adversary can query for any attributes keys as long as they can not be applied directly to decrypt the ciphertext. The ciphertext is provided by the challenger and encrypted under an access structure with attribute public keys. The challenger is responsible for the secret key generation and hides its details from the adversary. Now the security game is described as follows: the adversary is challenged by the ciphertext encrypted under the access structure $\mathcal{M}^*$. He/She can query private keys of any attribute set S that can't satisfy $\mathcal{M}^*$. The formal security game is described as follows:

1) *Setup*. The challenger runs the *System Initialization* operation in our system to generate system parameters.
2) *Secret Key Query Phase I*. The adversary makes private key queries for attribute set S not satisfying the access structure $\mathcal{M}^*$ to the challenger.
3) *Challenge*. The adversary submits two messages with equal length, M_0 and M_1, to the challenger. In addition, the challenger gains the access structure $\mathcal{M}^*$ from the adversary. The access structure $\mathcal{M}^*$ cannot be satisfied by the attribute set S in phase I. The challenger randomly chooses one message marked as M_β from the two messages submitted by the adversary, then he/she encrypts the message M_β under the access structure $\mathcal{M}^*$. The ciphertext is given to the adversary.
4) *Secret Key Query Phase II*. The adversary can repeat *phase I* to ask more private keys of the attribute set S as long as S doesn't satisfy the access structure $\mathcal{M}^*$.
5) *Guess*. The adversary outputs a guess β' of β.

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

Definition 2. *Our multi-authority CP-ABE scheme is secure if all polynomial time adversaries have at most a negligible advantage in the above game.*



(1) *AA* registers to *CA* to gain $(aid, aid.cert)$;
(2) User registers to *CA* to gain $(uid, uid.cert)$;
(3) User gains his/her *SK* from any t out of n *AA*s;
(4) Owners gain *PK* from *CA*;
(5) Owners upload (*CT*) to the cloud server;
(6) Users download (*CT*) from the cloud server.

Fig. 1. Framework and basic protocol flow.

4 Our Proposed Threshold Multi-Authority CP-ABE Data Access Control System

In this section, we first give an overview of TMACS, including the scheme structure and the challenging issues in the design of TMACS. In the following, we detailed describe TMACS, which mainly consists of four phases: *System Initialization*, *Secret Key Generation*, *Encryption*, and *Decryption*.

4.1 Overview of Our Scheme

To address the problem of single-point bottleneck, we introduce (t, n) threshold secret sharing, based on redundant multiple *AA*s, then propose a threshold multi-authority CP-ABE and the relevant access control scheme TMACS in public cloud storage.

In TMACS, the overall framework of the system, described in Fig. 1, is similar to DAC-MACS [20] proposed by Yang et al. The main difference is: In DAC-MACS, the whole attribute set is divided into multiple disjoint subsets and each one of the multiple authorities maintains one attribute subset. By contrast, in TMACS, multiple authorities jointly manage the whole attribute set but no one has full control of any specific attribute. In TMACS, a global certificate authority is responsible for the construction of the system, which avoids the extra overhead caused by *AA*s' negotiation of system parameters. *CA* is also responsible for the registration of users, which avoids *AA*s synchronized maintaining a list of users. However, *CA* is not involved in *AA*s' master key sharing and users' secret key generation, which avoids *CA* becoming the security vulnerability and performance bottleneck.

The scheme structure of TMACS can also be summarised in Fig. 1. In TMACS, *AA*s must first register to *CA* to gain the corresponding identity and certificate (aid, aid.cert). Then *AA*s will be involved in the construction of the system, assisting *CA* to finish the establishment of system parameters. *CA* accepts users' registration and issues the certificate (uid, uid.cert) to each legal user. With the certificate, the user can contract with any t *AA*s one by one to gain his/her secret key. Owners who want share their data in the cloud can gain the public key from *CA*. Then the owner can

encrypt his/her data under predefined access policy and upload the ciphertext (*CT*) to the cloud server. User can freely download the ciphertexts that he/she is interested in from the cloud server. However, he/she can't decrypt the ciphertext unless his/her attributes satisfy the access policy hidden inside the ciphertexts.

One challenging issue in design of TMACS is reusing of the master key shared among multiple attribute authorities. In traditional (t, n) threshold secret sharing, once the secret is reconstructed among multiple participants, someone can actually gain its value. Similarly, in CP-ABE schemes, the only-one-authority knows the master key and uses it to generate each user's secret key according to a specific attribute set. In this case, if the *AA* is compromised by an adversary, it will become the security vulnerability. To avoid this, by means of (t, n) threshold secret sharing, the master key cannot be individually reconstructed and gained by any entity in TMACS. In TMACS, the master key α is implicitly reconstructed by reconstructing g^{α} and $e(g,g)^{\alpha}$. It is a discrete logarithm problem to calculate the master key α from g^{α} or $e(g,g)^{\alpha}$, which means that the master key α is actually secure. By this means, we solve the problem of reusing of the master key.

How to guarantee the flexibility of the system in users' secret key generation is another challenging issue. In traditional (t, n) threshold secret sharing, the secret can be reconstructed unless there are at least t participants cooperating with each other. This means that, if just simply introducing traditional (t, n) threshold secret sharing into our multi-authority CP-ABE design, the user should contact with t *AA*s during the secret key generation for each time, and the chosen t *AA*s also have to contact with each other to implicitly reconstruct the master key. This will bring too much communication overhead, which is not flexible for system performing.

To reduce the trivial communication overhead, in TMACS, rather than the master key, the entire secret key is reconstructed by collecting t secret key shares generated by *AA*s. Furthermore, the reconstructed process can be done by the user rather than the specific t *AA*s. By this means, the user can contact with the t *AA*s one by one, which is suit for real application scenarios, enhances the flexibility of the system, avoids the extra communication overhead and synchronization issues among *AA*s.

4.2 Details of Our Data Access Control Scheme

4.2.1 System Initialization

The operation of *System Initialization* is divided into three sub-processes: *CASetup1*, *AASetup*, and *CASetup2*. The operation of *CASetup1* is mainly responsible for establishment of system parameters and accepting registration of users and *AA*s. *AA*s cooperate with each other to share the master key in *AASetup*, while the corresponding public key is generated by *CA* in *CASetup2*.

1) *CASetup1*: The operation of *CASetup1* is run by *CA*. First, *CA* chooses two multiplicative cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$ with the same prime order p, then defines a binary map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ on $\mathbb{G}$. *CA* chooses a random $a \in \mathbb{Z}_p$ as the master key, and then calculates the relevant public key part g^a. Here, the parameter g is a generator of $\mathbb{G}$. *CA* generates a pair of keys (sk_{CA}, vk_{CA}) to sign and verify, in which, vk_{CA} is publicly known by each entity in the system. *CA* also generates public keys for each attribute $Att_i, (i = 1, 2, \ldots, U)$: $h_1, h_2, \ldots, h_U \in \mathbb{G}$. To distinguish each user and *AA*, the following two processes are described separately:
   - *User registration*: Each user sends a registration request to *CA* during the phase of *System Initialization*. *CA* authenticates the user, then assigns an identification uid to him/her. The identification uid is a random element in $\mathbb{Z}_p$. *CA* signs a certificate $uid.Cert$ with sk_{CA} for the user to verify his/her identity.
   - AA*registration*: Each *AA* also sends a registration request to *CA* during the *System Initialization*. For each legal authority, *CA* assigns a unique identity $aid \in \mathbb{Z}_p$ and generates a certificate $aid.Cert$.

According to the total number (marked as n) of *AA*s involved in the system, *CA* decides the threshold number (marked as t) of *AA*s that participate in user's secret key generation for each time.

1) *AASetup*: The operation of *AASetup* is run by each one of all n *AA*s. These n *AA*s cooperate with each other to call (t, n) threshold secret sharing as follows:
   - Each *AA* $(AA_i, i = 1, 2, \ldots, n)$ selects a random number $\alpha_i \in \mathbb{Z}_p$ as its sub-secret, in this way, the master key α is implicitly decided: $\alpha = \sum_{i=1}^{n} \alpha_i$. The value of α shouldn't be gained by any entity alone. Then each *AA* $(AA_i, i = 1, 2, \ldots, n)$ separately generates a random polynomial $f_i(x)$ of degree $t-1$ that satisfies the formula $\alpha_i = f_i(0)$. AA_i calculates the sub-share $s_{ij} = f_i(aid_j)$ for each of the other *AA* $(AA_j,\ j = 1, 2, \ldots, i-1, i+1, \ldots, n)$, and sends the sub-share s_{ij} to AA_j securely. Meanwhile, AA_i calculates $s_{ii} = f_i(aid_i)$ for itself.
   - After receiving $n-1$ sub-shares s_{ji} $(j = 1, 2, \ldots, i-1, i+1, \ldots, n)$ from all of the other $n-1$ *AA*s $(AA_j, j = 1, 2, \ldots, i-1, i+1, \ldots, n)$, AA_i calculates its master key share: $sk_i = \sum_{j=1}^{n} s_{ji}$, then AA_i futher calculates its relevant public key share: $pk_i = e(g,g)^{sk_i}$.

After finishing the operation of *AASetup*, each *AA* $(AA_i, i = 1,2, \ldots, n)$ gains a pair of keys (sk_i, pk_i). Here, the public key share pk_i can be shared with any other entities, including *CA*.

1) *CASetup2*: The operation of *CASetup2* is run by *CA*. To calculate the global public key, *CA* randomly chooses t out of n *AA*s' public key shares, denoted as pk_i, $i = 1,2, \ldots, t$. Then *CA* calculates the global public key as follows:

$$e(g,g)^{\alpha} = e(g,g)^{\sum_{i=1}^{t} \left(sk_i \prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i} \right)} = \prod_{i=1}^{t} e(g,g)^{sk_i \prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i}} = \prod_{i=1}^{t} pk_i^{\prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i}}.$$

Description	$E_\kappa(M)$	$C, C', \{C_i, D_i\}_{i=1\ to\ l}$

Fig. 2. Ciphertext format.

Now, the operation of *System Initialization* is finished, and it returns the public key PK:

$$PK = (g,\ g^a,\ e(g,g)^\alpha, n, t, h_1, \ldots, h_U).$$

Meanwhile, $MK = (a, \alpha)$ implicitly exists in the system, which doesn't need to be obtained by any entity.

4.2.2 Encryption

The operation of *Encryption* is implemented by a specific data owner independently. To improve the system's performance, the owner first chooses a random number $\kappa \in \mathbb{Z}_p$ as the symmetric key and encrypts the plaintext message M using κ with the symmetric encryption algorithm, such as *AES*. The encrypted data can be denoted as $E_\kappa(M)$, then the owner encrypts the symmetric key κ using CP-ABE under an access policy defined by himself/herself. The owner first defines an easy expressed monotone boolean formula. By following the method defined in [26], he/she can turn it to a LSSS access structure, which can be denoted as $(\mathcal{M}, \rho)$. $\mathcal{M}$ is a $l \times k$ matrix, where l is the scale of a specific attribute set and k is variable that is depend on the monotone boolean formula definition and the LSSS turning method. The function ρ maps each row of $\mathcal{M}$ to a specific attribute, marked as $\rho(i)(\in \{Att_1, Att_2, \ldots, Att_U\})$. A random secret parameter s is chosen to encrypt the symmetric key κ. To hide the parameter s, a random vector $\vec{v} = (s, y_2, y_3, \ldots, y_k) \in \mathbb{Z}_p^n$ is selected, where $y_2, y_3, \ldots, y_k$ are randomly chosen and used to share the perameter s. Each $\lambda_i = \mathcal{M}_i \vec{v}^\top$ is calculated for $i = 1, 2, \ldots, l$, where $\mathcal{M}_i$ denotes the ith row of the matrix $\mathcal{M}$. The owner randomly selects $r_1, r_2, \ldots, r_l \in \mathbb{Z}_p$ and calculates the ciphertext CT using the public keys gained from *CA*:

$$CT = (C = \kappa e(g,g)^{\alpha s},\ C' = g^s,$$
$$\forall i = 1\ to\ l,\ C_i = (g^a)^{\lambda_i} \cdot {h_{\rho(i)}}^{-r_i},\ D_i = g^{r_i}).$$

Finally, the owner sends the encrypted data $E_\kappa(M)$ encrypted by the symmetric key κ together with CT to the cloud server as the format in Fig. 2.

4.2.3 Secret Key Generation

The *Secret Key Generation* operation is run by one user and any t out of n *AA*s. Less than t *AA*s, user's secret key cannot be generated. In this operation, there is no interaction between any two of t *AA*s, so the user can select t *AA*s according to his/her own preference, and then separately contact with each of these t *AA*s to get the secret key share. After getting t secret key shares separately from t *AA*s, the user can generate his/her secret key.

To gain the secret key share from AA_i, the user uid_j first sends his/her signed request including his/her identity and his/her certificate to AA_i. After receiving the request, AA_i verifies uid_j's certificate by using *CA*'s public verification key vk_{CA}, then authenticates the user by verifying his/her signature over the request. If the user is an illegal one, the operation aborts. Otherwise, AA_i assigns an attribute set S to the user according to the role he/she plays in the domain[1] and generates the secret key share for him/her. AA_i first chooses a random number $b_i \in \mathbb{Z}_p$ and then generates the secret key share as follows:

$$K_i = g^{sk_i} \cdot g^{a \cdot b_i},\ L_i = g^{b_i},\ \forall Att_i \in S : K_{Att_i} = {h_{Att_i}}^{b_i}.$$

After collecting t secret key shares generated by t different *AA*s, the user can generate his/her secret key as follows:

$$K = \prod_{i=1}^{t} K_i^{\prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i}},\quad L = \prod_{i=1}^{t} L_i^{\prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i}},$$
$$\forall Att \in S :\ K_{Att} = \prod_{i=1}^{t} K_{Att_i}^{\prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i}}.$$

Furthermore, the formulas of the user's secret key can be expressed as follows:

$$\begin{aligned}
K &= \prod_{i=1}^{t} \left(g^{sk_i} \cdot g^{a \cdot b_i}\right)^{\prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i}} \\
&= \prod_{i=1}^{t} \left(\left(g^{sk_i}\right)^{\prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i}} \cdot \left(g^{a \cdot b_i}\right)^{\prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i}} \right) \\
&= g^{\sum_{i=1}^{t} \left(sk_i \cdot \prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i} \right)} \cdot g^{\sum_{i=1}^{t} \left(a \cdot b_i \cdot \prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i} \right)} \\
&= g^{\alpha} \cdot g^{a \cdot \sum_{i=1}^{t} \left(b_i \cdot \prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i} \right)}, \\
L &= g^{\sum_{i=1}^{t} \left(b_i \cdot \prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i} \right)}, \\
\forall Att \in S : K_{Att} &= h_{Att}^{\sum_{i=1}^{t} \left(b_i \cdot \prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i} \right)}.
\end{aligned}$$

To simplify the formulas, a parameter d is introduced:

$$d = \sum_{i=1}^{t} \left(b_i \cdot \prod_{j=1, j\neq i}^{t} \frac{aid_j}{aid_j - aid_i} \right).$$

Therefore, the formulas of user's secret key can be simply denoted as:

$$K = g^\alpha \cdot g^{a \cdot d},\ L = g^d,\ \forall Att \in S : K_{Att} = {h_{Att}}^d.$$

4.2.4 Decryption

The *Decryption* operation is run by each user. The user can freely query and download any encrypted data that he/she is interested in from the cloud server. However, he/she can't decrypted the data unless his/her attribute set satisfies the access structure hidden inside the ciphertext. For the user U, let $\mathcal{M}_U$ be a sub-matrix of $\mathcal{M}$, where each row of $\mathcal{M}_U$ corresponds to a specific attribute in U's attribute set S_U. Let $I \subset \{1, 2, \ldots, l\}$ denote $\{i : \rho(i) \in S_U\}$, and $\mathcal{M}_i$ denote the ith row of $\mathcal{M}$.

If the user U's attribute set S_U satisfies the access structure $(\mathcal{M}, \rho)$, the vector $\vec{e} = (1, 0, \ldots, 0)$ is in the span of matrix $\mathcal{M}_U$, which means an appropriate parameter $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ can be found to satisfy $\vec{e} = (\omega_1, \omega_2, \ldots, \omega_{|I|})\mathcal{M}_U$.

1. The assignment operation should accord the same standard by all *AA*s , so the assignment result from different *AA*s are consistent.

The parameter $\{\omega_i\}_{i\in I}$ can further help the user to find the hidden secret parameter s:

$$\begin{aligned} s &= (1,0,\ldots,0)\cdot(s,y_2,\ldots,y_n)^\top \\ &= \vec{e}\cdot\vec{v}^\top = (\omega_1,\omega_2,\ldots,\omega_{|I|})\mathcal{M}_U\cdot\vec{v}^\top \\ &= (\omega_1,\omega_2,\ldots,\omega_{|I|})\cdot\vec{\lambda_I}^\top = \sum_{i\in I}\omega_i\cdot\lambda_i. \end{aligned}$$

Here, $\vec{\lambda_I}$ denotes the sub-vector of $(\lambda_1,\lambda_2,\ldots,\lambda_l)$, then with the help of parameter $\{\omega_i\}_{i\in I}$, the user computes:

$$\begin{aligned} C_U &= \frac{e(C',K)}{\prod_{i\in I}\big(e(C_i,L)\cdot e\big(D_i,K_{\rho(i)}\big)\big)^{\omega_i}} \\ &= \frac{e\big(g^s,g^\alpha\cdot g^{a\cdot d}\big)}{\prod_{i\in I}\Big(e\Big((g^a)^{\lambda_i}\cdot {h_{\rho(i)}}^{-r_i},g^d\Big)\cdot e\big(g^{r_i},{h_{\rho(i)}}^d\big)\Big)^{\omega_i}} \\ &= \frac{e(g,g)^{\alpha\cdot s}\cdot e(g,g)^{a\cdot s\cdot d}}{\prod_{i\in I}e(g,g)^{d\cdot a\cdot\lambda_i\cdot\omega_i}} = e(g,g)^{\alpha s}. \end{aligned}$$

The user can further get the symmetric key κ by dividing out the above value C_U from C:

$$\kappa = C/C_U = C/e(g,g)^{\alpha s}.$$

With the computed symmetric key κ, the user can decrypt the ciphertext to get the final plaintext data M.

5 Security and Performance Analysis of TMACS

5.1 Security Analysis

We analyze the security properties of TMACS according to the security model defined in Section 3.

5.1.1 Security of Access Policy

To prove the security of access policy in TMACS, we use the following theorem:

Theorem 1. *When the decisional q-BDHE assumption holds, no adversary can use a polynomial-time algorithm to selectively break TMACS with a challenge matrix of size $l^*\times n^*$, where $l^*, n^*\le q$.*

Proof. Suppose we have an adversary $\mathcal{A}$ with non-negligible advantage $\epsilon = Adv\mathcal{A}$ in the selective security game defined in Section 3 against our construction. Moreover, suppose he/she chooses a challenge matrix $\mathcal{M}^*$ where each of both dimensions is at most q. In the security game, the adversary $\mathcal{A}$ can query any secret keys that can't be directly used for decrypting the ciphertext from the challenger. The challenger hides details of AAs generating secret key shares and generates the integral secret key for the adversary $\mathcal{A}$. In this way, the security game can be treated as same as the single authority one. We can also build a simulator $\mathcal{B}$, which plays the decisional q-BDHE problem. The detailed proof is described as follows:

1) **Setup**. The simulator $\mathcal{B}$ chooses a random number $\alpha'\in\mathbb{Z}_p$ and implicitly sets $\alpha=\alpha'+a^{q+1}$ by letting $e(g,g)^\alpha = e(g^a,g^{a^q})e(g,g)^{\alpha'}$. Here, we describe how the simulator "programs" the group elements $h_1,\ldots,h_U$. For each $x\in[1,U]$, choose a random value z_x. Let I^* denote the set

$$\{i:\forall x\in[1,U],\rho^*(i)=x\}.$$

The simulator programs h_x as:

$$h_x = g^{z_x}\cdot\prod_{i\in I^*} g^{a\cdot M^*_{i,1}/b_i}\cdot g^{a^2\cdot M^*_{i,2}/b_i}\ldots g^{a^{n^*}\cdot M^*_{i,n^*}/b_i}.$$

Note that if $I^*=\phi$ we have $h_x=g^{z_x}$.

2) **Secret Key Query Phase I**. In this phase the simulator $\mathcal{B}$ answers private key queries. Suppose the simulator $\mathcal{B}$ is given a private key query for a attribute set S where S does not satisfy $\mathcal{M}^*$. The simulator $\mathcal{B}$ first chooses a random $r\in\mathbb{Z}_p$, and then finds a vector $\vec{\omega}^*=(\omega_1^*,\omega_2^*,\ldots,\omega_{n^*}^*)\in\mathbb{Z}_p^{n^*}$, such that $\omega_1^*=-1$ and for all $i\in I^*$, we have $\vec{\omega}^*\cdot M_i^*=0$. By the definition of a LSSS such a vector $\vec{\omega}^*$ must exist. Note that if such a vector does not exist, the vector $(1,0,0,\ldots,0)$ will be in the span of S. The simulator $\mathcal{B}$ implicitly defines d as

$$d = r+\omega_1^*\cdot a^q+\omega_2^*\cdot a^{q-1}+\cdots+\omega_{n^*}^*\cdot a^{q-n^*+1}.$$

It performs this by setting

$$L = g^r\cdot\prod_{i=1}^{n^*}(g^{a^{q+1-i}})^{\omega_i^*} = g^d.$$

We observe that by the definition of d, $g^{a\cdot d}$ contains a term of $g^{-a^{q+1}}$, which will cancel out with the unknown term in g^α when creating K. The simulator $\mathcal{B}$ can compute K as:

$$K = g^{\alpha'}\cdot g^{ar}\cdot\prod_{i=2}^{n^*}(g^{a^{q+2-i}})^{\omega_i^*}.$$

Now we further calculate $K_x,\forall x\in S$. First, we consider $x\in S$ for which there is no i such that $\rho^*(i)=x$. We can simply set $K_x=L^{z_x}$.

The more difficult task is to create key components K_x for attributes $x\in S$, where x is used in the access structure. For these keys we must make sure that we can simulate all terms of the form g^{a^{q+1}/b_i} or cancel out them. Fortunately, we have that $M_i^*\cdot\vec{\omega}^*=0$; therefore, all of these terms cancel.

Furthermore, let I^* be the set of all i: $\{i:\rho^*(i)=x,\forall x\in S\}$. The simulator $\mathcal{B}$ creates K_x as follows:

$$K_x = L^{z_x}\cdot\prod_{i\in I^*}\prod_{j=1}^{n^*}\left(g^{(a^j/b_i)\cdot r}\cdot\prod_{k=1,k\neq j}^{n^*}(g^{a^{q+1+j-k}/b_i})^{\omega_k^*}\right)^{M^*_{i,j}}.$$

3) **Challenge**. In this phase, we build the challenge ciphertext. The adversary $\mathcal{A}$ gives two messages M_0,M_1 to the simulator $\mathcal{B}$. The simulator $\mathcal{B}$ flips a

coin β. It creates $C = M_\beta Te(g^s, g^{\alpha'})$ and $C' = g^s$. The tricky part here is to simulate the value of C_i $(i = 1, 2, \ldots, n^*)$ since it contains terms that we must cancel out. However, the simulator $\mathcal{B}$ can choose the secret share, such that these can be canceled out. Intuitively, the simulator $\mathcal{B}$ will choose random $y'_2, \ldots, y'_{n^*}$ and share the secret using the vector

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

In addition, it chooses random values $r'_1, \ldots, r'_l$.

For all $i \in [1, n^*]$, we define R_i as the set of all $k \neq i$ such that $\rho^*(i) = \rho^*(k)$. In other words, the set of all other row indices that have the same attribute as row i. The challenge ciphertext components are then generated as:

$$D_i = g^{-r'_i} g^{-sb_i}$$

$$C_i = h_{\rho^*(i)}^{r'_i} \left(\prod_{j=2}^{n^*} (g^a)^{M^*_{i,j} \cdot y'_j} \right) \cdot (g^{b_i \cdot s})^{-z_{\rho^*(i)}} \cdot \left(\prod_{k \in R_i} \prod_{j=1}^{n^*} \left(g^{a^j \cdot s \cdot (b_i / b_k)} \right)^{M^*_{k,j}} \right)$$

4) **Secret Key Query Phase II**. This phase is the same as **Secret Key Query Phase I**.

5) **Guess**. In this phase, the adversary $\mathcal{A}$ outputs a guess β' of β. If $\beta' = \beta$, the simulator $\mathcal{B}$ outputs "0" to guess that $T = e(g, g)^{a^{q+1}s}$; otherwise, it outputs "1" to indicate that it believes T is a random number in $\mathbb{G}_T$. When T is a tuple the simulator $\mathcal{B}$ gives a perfect simulation so that we have

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

When T is a random number in $\mathbb{G}_T$, the message M_β is completely hidden from the adversary $\mathcal{A}$ and we have

$$Pr[\mathcal{B}(\vec{y}, T = R) = 0] = \frac{1}{2}.$$

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

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

This is against the Definition 1, thus we can say that there is not an adversary that can have an non-negligible advantage in the selective security game defined in Section 3. According to the Definition 2, we can conclude that our multi-authority CP-ABE scheme is secure. □

5.1.2 Security Against Collusion Attack

When some malicious users collude with each other, they may share their secret keys to gain more privilege. However, they will be disappointed for the existence of the random element $d = \sum_{i=1}^{t} (b_i \prod_{j=1, j \neq i}^{t} \frac{aid_j}{aid_j - aid_i})$ in the secret key. Due to the existence of the random element, each component associated with the same attribute in different users' secret keys is distinguishable. So that they can't gain more privilege by combining their secret keys.

5.1.3 Confidentiality Guarantee

The cloud server can be seen as an adversary for it is "honest and curious", but it doesn't have any advantages compared with malicious users. The cloud server doesn't participate in *AA*s' master key sharing or any users' secret key generation. All it does is only storing the ciphertext which makes no difference for it to gain the plaintext, even if it colludes with the malicious users.

5.1.4 Soundness and Completeness

As defined in Section 3.2, the malicious users (adversary) can compromise *AA*s to gain more privilege. On one hand, the adversary can compromise *AA*s to gain the master key to generate secret key for himself/herself; on the other hand, he/she can deceive *AA*s to generate his/her secret key shares so that he/she can reconstruct his/her secret key.

With regard to the former possible threaten, we introduce the concept of soundness and completeness. In our scheme, the soundness means that the adversary can't gain any benefit if there is only less than t *AA*s comprised by him/her; the completeness means that as long as there are more than t *AA*s operation normally, the system can work properly. The soundness and completeness can be guaranteed by (t,n) threshold secret sharing.

In our scheme, we introduce (t, n) threshold secret sharing in [16] to share the master key among multiple *AA*s, which uses Shamir's (t, n) threshold secret sharing [17] as a building block. Shamir's (t, n) threshold secret sharing holds the following properties in information-theoretically secure degree: (1) knowledge of any t or more secret pieces makes the secret easily computable; (2) knowledge of any $t-1$ or fewer secret pieces leaves the secret completely undermined (in the sense that all its possible values are equally likely). Thus, based on the two properties, our scheme can guarantee that (1) any t or more *AA*s master key shares can easily reconstruct *AA*s master key; (2) less than t *AA*s master key shares can't learn anything about the master key. In our scheme, users' secret key can't be generated unless the master key can be reconstructed. Thus, our scheme actually have achieved soundness and completeness.

The soundness guarantees that the adversary has to compromise no less than t *AA*s to obtain illegal benefits. The ability of our scheme against this attack will be detailed discussed in the following section. The completeness guarantees that our scheme can be robust against *AA*s' crash or offline. The detailed analysis will be discussed in Section 5.2.4.

5.1.5 Security Against Compromising AAs

Here, we give the security analysis against the attack that the adversary compromises *AA*s. The soundness guarantees that the adversary has to compromise more than t *AA*s to assign secret key shares for him/her. Now let the parameter

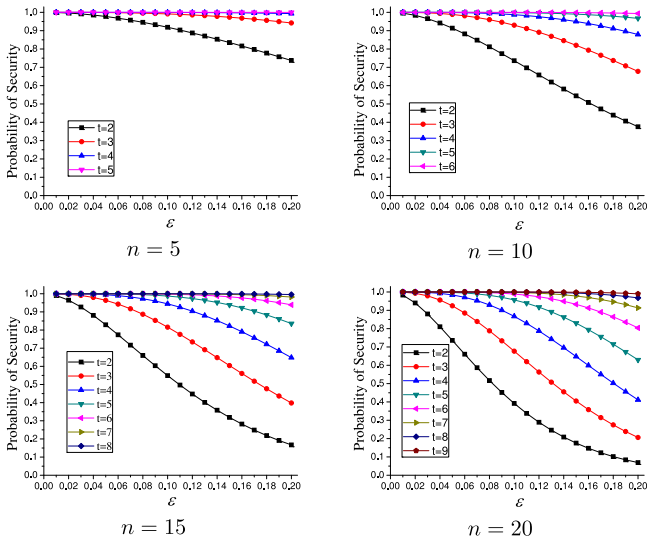


Fig. 3. Probability of security against *AA* compromise.

ε denote the probability that an illegal user gains one piece secret key share from one *AA*. Then the probability of TMACS security is:

$$\sum_{i=0}^{t-1}\binom{i}{n}\varepsilon^i(1-\varepsilon)^{n-i}.$$

To have an intuitive understanding for the security of TMACS against this attack, we draw the probability map in Fig. 3, which shows the probability of system security versus the number of *AA*s n and the probability ε. From the figure, we can see that for any number of authorities, we can increase the value of t to make TMACS secure against the above attack as a high probability. But we can also notice that the overlarge value of t will only bring extra overhead rather than increase the security effectively. For example, the system can be secure with probability close to 1, when we set the value of t only equal to 5 rather than a larger value in a system with 10 authorities, even the adversary can device the authority as a high probability 0.2. We can say, TMACS can be secure against the above attack with an appropriate threshold value t.

5.2 Performance Analysis

This section numerically evaluates the performance of TMACS in terms of storage, communication, and computation overhead. At the same time, we give a detailed analysis about how we can achieve system robustness to deal with authority crash or offline.

TABLE 1
Storage Overhead

Entity	TMACS	Waters's Scheme
CA	$(2U+2N_u+n+10)\|p\|$	0
AA	$(6+2U)\|p\|$ $(each\ AA)$	$(2U+2N_u+9)\|p\|$
Owner	$(2U+4)\|p\|$	$(2U+4)\|p\|$
User	$(5+2N_{uid})\|p\|$	$(4+2N_{uid})\|p\|$

For a better understanding, we conduct performance analysis between TMACS and Waters's scheme [10]. Both schemes are based on the same security assumption decisional q-BDHE assumption and support the same LSSS access structure. Besides, TMACS and existing multi-authority CP-ABE schemes are based on different system models, so that there is no comparable between them. Thus we choose Waters's scheme to conduct the performance analysis.

To conduct performance analysis, we have the following definitions: Let $|p|$ be the size of element in the groups with prime order p. Let n be the number of *AA*s and t be the threshold value. Let U be the total number of attributes. Let N_u be the number of users in the system and N_o be the number of owners. Let N_{uid} denote the average number of attributes owned by user uid.

5.2.1 Storage Overhead

Storage overhead on *CA*, each *AA*, user, and owner of Waters's scheme and TMACS is shown in Table 1.

In Waters's scheme, there is not *CA*. But the authority (*AA*) in their scheme bears the same responsibility as *CA* in TMACS. Comparing the storage overhead on *CA* in TMACS with that on *AA* in Waters's scheme, there is no much difference except n *AA*s' aid stored on *CA* in TMACS. To relieve communication pressure on *CA*, the public keys are backed up in *AA*s and owners. The main storage overhead on each *AA* in TMACS is the backup of the public keys. On owners and users, there is no much difference about storage overhead between TMACS and Waters's scheme. The analysis above shows that no additional storage overhead is introduced in TMACS but the backup of attributes on *AA*s and the n aid on *CA*.

5.2.2 Communication Overhead

Table 2 shows comparison result about communication overhead on *CA*, each *AA*, owner, and user of TMACS and Waters's scheme. It's clear that the communication overhead of TMACS is larger than Waters's scheme, which can be anticipated for the introduction of *AA*s. However, most

TABLE 2
Communication Overhead

Process	Setup				Key Generation	
Entity	*CA*	*AA*	Owner	User	*AA*	User
TMACS	$(10n+4N_u+2UN_o+4N_o+2nU)\|p\|$	$(2n+2U+8)\|p\|(each\ AA)$	$(2U+4)\|p\|$	$4\|p\|$	$(4+2N_{uid})\|p\|(each\ AA)$	$(2N_{uid}+4)t\|p\|$
Waters's scheme	0	$(2N_u+2UN_o+4N_o)\|p\|$	$(2U+4)\|p\|$	$2\|p\|$	$(4+2N_{uid})\|p\|$	$(2N_{uid}+4)\|p\|$

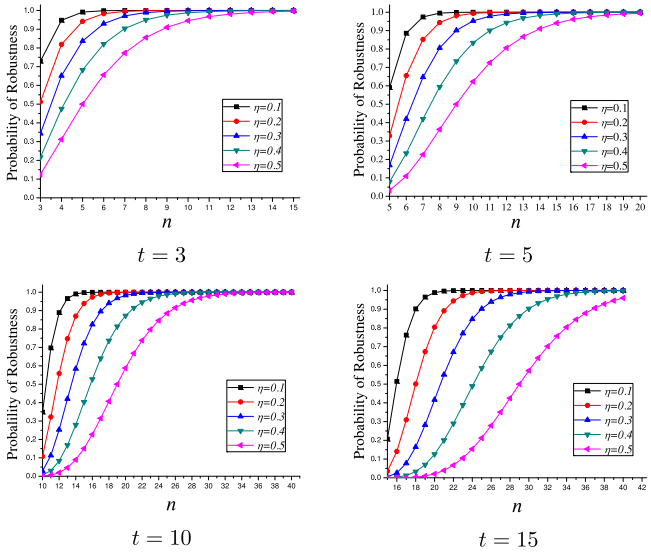


Fig. 4. Probability of robustness against *AA* crash or offline.

communication overhead is only needed in the process of construction of the system. After the system is constructed, the main communication overhead is only users' secret key generation. We can see that, compared with Waters's scheme, there is no much communication overhead addition on each entity except user in TMACS. But the addition communication overhead on user cannot be avoid for the introduction of the multiple authorities, which is no different between TMACS and existing multi-authority schemes.

5.2.3 Computation Overhead

Here, we give a simple analysis about the additional computation overhead introduced by the n *AA*s. The introduction of the n *AA*s mainly increases users' computation overhead during the process of users' secret key generation, while there is no difference about the computation overhead on encryption and decryption compared with Waters's scheme. Users must calculate the integral secret key from the t secret key shares generated by *AA*s, which makes it inevitable for the increase of the computation overhead.

5.2.4 Robustness

As we have defined in Section 5.1.4, the completeness guarantee that our scheme can work properly with no less than t *AA*s. That means, the crash or offline of no more than $n-t$ *AA*s doesn't do harm to the normal operation of the system. In the following, we analyze the robustness of TMACS from the point of probability.

Let η denote the probability of a single *AA* crash or offline. The probability of system normal operation is 1-η in the single authority scheme. In TMACS, the system can operate normally as long as there are t *AA*s in good condition. So the probability of system normal operation in TMACS is:

$$1-\sum_{i=0}^{t-1}\binom{i}{n}(1-\eta)^i\eta^{n-i}.$$

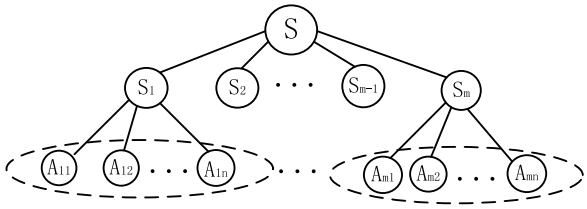


Fig. 5. Attribute management model.

In Fig. 4, we draw the probability map versus the number of *AA*s n and the threshold value t. From the figure, we can see that TMACS can be robust against authorities crashing or offline because of existence of redundant authorities. For example, even if the authority is crashed or offline as a probability of 50 percent, we can still set an appropriate proportion, like $t=10, n=30$, to make TMACS work properly. By the way, a lager n, like 40, has no use for system robustness, but only increases system overhead.

Combing the security analysis against compromising *AA*s with the performance analysis, we can find that the choice of t is a trade-off between security and robustness. But from the analysis, we can be sure that the appropriate value of (t,n) can still be found easily. For example, we can set the whole number of authorities as 15 and the value of t as 5. In this way, even the adversary can compromise authorities as a probability of 0.2, and the authorities may be crashed or offline as a high probability of 50 percent, the system can still achieve both security and robustness as a high probability.

Restriction. From the performance analysis, while enhancing robustness and security, the introduced redundancy among *AA*s will increase the overhead of computing, communication and storage. Thus, the efficiency seems as the main restriction of our scheme. However, compared with the benefit gained from the aspect of secrecy and robustness, the extra overhead is not very high, which is worthy and can be ignored.

6 The Enhanced Scheme

Usually two different multi-authority scenarios both exist in the real complex environment, where attributes come from different authority-sets and multiple authorities in an authority-set jointly maintain a subset of the whole attribute set. To satisfy this hybrid scenario, we conduct a hybrid multi-authority access control scheme, by combining the traditional multi-authority scheme [14] with our proposed TMACS. In the enhanced scheme, the whole attribute set is divided into disjoint subsets to be maintained by different authority-sets. Each attribute subset is managed by n *AA*s in the same authority-set jointly. The attribute management model is shown in Fig. 5. This model has both advantages: On one hand, it satisfies the scenario of attributes from different *AA*s; on the other hand, it can achieve security and system-level robustness.

We first assume the threshold and the total number of *AA*s are equal in different authority-sets, which is not necessary just because of easy description. Now, we give a brief introduction of our enhanced scheme to meet this model.

1) *Global setup*: The global public parameters are published publicly, which include a bilinear group $\mathbb{G}$ of prime order p, a generator g of $\mathbb{G}$, and a function H

mapping user's global identity uid to one element of $\mathbb{G}$.

2) *Authority setup*: For each attribute i maintained by the authority-set AA_j, the authorities in the authority-set AA_j cooperate with each other to call (t, n) threshold secret sharing to generate the attribute private key (α_i, y_i). After that, each AA (AA_{jk}) keeps a private key share $(sk_{\alpha_i,jk}, sk_{y_i,jk})$ as its secret key. Then these AAs cooperate with each other to calculate the public key of attribute i: $(e(g,g)^{\alpha_i}, g^{y_i})$.

3) *KeyGen*: To create a key to the user with uid for attribute i maintained by the authority-set AA_j, the authority AA_{jk} computes:

$$K_{i,uid,jk} = g^{sk_{\alpha_i,jk}} H(uid)^{sk_{y_i,jk}}.$$

The user uid gains t secret key shares for attribute i from any t authorities in authority-set AA_j, then he/she can generate the secret key through Lagrange interpolating formula:

$$K_{i,uid} = g^{\alpha_i} H(uid)^{y_i}.$$

4) *Encrypt*: The access structure is similar to the description in Section 4. Besides, we choose a random vector $\vec{c} \in \mathbb{Z}_p^l$ with 0 as its first entry. Let c_x denote $\mathcal{M}_x \cdot \vec{c}$. The ciphertext is computed as:

$$C_0 = M \cdot e(g,g)^s, C_{1,x} = e(g,g)^{\lambda_x} \cdot e(g,g)^{\alpha_{\rho(x)} r_x},$$
$$C_{2,x} = g^{r_x}, C_{3,x} = g^{y_{\rho(x)} r_x} g^{c_x} \ \forall x.$$

5) *Decrypt*: For each row of $\mathcal{M}_U$ defined in Section 4, denoted as x, the user computes:

$$C_{1,x} \cdot e\big(H(uid), C_{3,x}\big) / e\big(K_{\rho(x),uid}, C_{2,x}\big) = e(g,g)^{\lambda_x} \cdot e(H(uid), g)^{c_x}.$$

The user then chooses constants $\omega_x \in \mathbb{Z}_p$ such that $\sum_x \omega_x \cdot \mathcal{M}_x = (1, 0, \ldots, 0)$ and computes:

$$\prod_x \big(e(g,g)^{\lambda_x} e(H(uid), g)^{c_x}\big)^{\omega_x} = e(g,g)^s.$$

The message can be obtained as: $M = C_0 / e(g,g)^s$.

7 Related Work

How to provide a fine-grained access control is a critical challenging issue in public cloud storage system [2], [3], while the access control can be easily and efficiently achieved in private cloud [2], [27]. For ABE is one of the most suitable schemes, Yu et al. [28] have introduced KP-ABE [7] into public cloud storage to conduct fine-grained data access control. After that, more data access control schemes based on single-authority ABE, such as [24], [29], [30], have been proposed. However, in real complex scenario, it seems impossible to find only-one-authority to manage all attributes, a user usually holds attributes issued by multiple authorities. How to make ABE satisfy the scenario where attributes come from multiple authorities has been proposed as an open problem by Sahai and Waters in [4]. Based on the basic ABE [4] scheme, Chase has proposed the first multi-authority ABE scheme [13], in which a global certification authority (*CA*) is introduced. However, in this scheme, *CA* may become security vulnerability and performance bottleneck of the system. Besides, the access structure is not flexible enough to satisfy complex environments. Subsequently, much effort has been made to deal with the disadvantages in the early schemes. Among them, some multi-authority ABE schemes without *CA* have been proposed, such as [14], [15], [31]. In [32], Li et al. have conducted a multi-authority KP-ABE data access control scheme for securing personal health records in public cloud storage.

Since the first construction of CP-ABE [9], many works [33], [34], [35], [36], have been proposed for more expressive, flexible and practical versions of this technique. Based on CP-ABE, Lewko and Waters [14] have proposed a none-*CA* multi-authority scheme, in which there is no global coordination other than the creation of an initial set of common reference parameters. Subsequently, some more practical multi-authority CP-ABE schemes for data access control in public cloud storage have been successively proposed, such as [20], [21], [22], [23], [25]. In Yang et al.'s scheme [20], [22], [23], there is still a global *CA*, which is only responsible for the construction initialization of the system rather than generating users' secret keys, so it can avoid *CA* becoming a security vulnerability of the system. To get rid of *CA*, Ruj at al. [21] have introduced Lewko and Waters's scheme [14] to conduct a distributed access control in public clouds.

In these multi-authority access control schemes, the whole attribute set is divided into multiple disjoint subsets and maintained by multiple authorities, but each attribute subset is still maintained by only one authority, which makes the problem of single-point bottleneck on both security and performance still exist in the system. As far as we know, in existing single-authority and multi-authority ABE schemes, no one has paid special attention to this problem. Although Jung et al. [25] have noted that the compromised authorities may harm the security of the system, since they are able to issue valid attribute keys for which they are in charge of, they have not proposed an appropriate solution. Instead, they rely on the assumption that the probability of compromising an authority is very low.

8 Conclusion

In this paper, we propose a new threshold multi-authority CP-ABE access control scheme, named TMACS, in public cloud storage, in which all *AA*s jointly manage the whole attribute set and share the master key α. Taking advantage of (t, n) threshold secret sharing, by interacting with any t *AA*s, a legal user can generate his/her secret key. Thus, TMACS avoids any one *AA* being a single-point bottleneck on both security and performance. The analysis results show that our access control scheme is robust and secure. We can easily find appropriate values of (t, n) to make TMACS not only secure when less than t authorities are compromised, but also robust when no less than t authorities are alive in the system. Furthermore, based on efficiently combining the traditional multi-authority scheme with TMACS, we also construct a hybrid scheme that is more suitable for the real scenario, in which attributes come

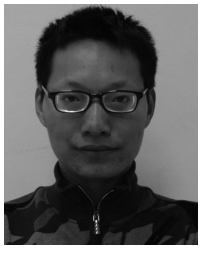
from different authority-sets and multiple authorities in an authority-set jointly maintain a subset of the whole attribute set. This enhanced scheme addresses not only attributes coming from different authorities but also security and system-level robustness. How to reasonably select the values of (t, n) in theory and design optimized interaction protocols will be addressed in our future work.

Acknowledgments

The authors sincerely thank the anonymous referees for their invaluable suggestions that have led to the present improved version of the original manuscript. This work is supported by the National Natural Science Foundation of China under Grant Nos. 61379129 and 61170231. K. Xue is the corresponding author.

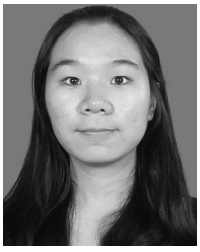
References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," *Nat. Instit. Standards Technol.*, vol. 53, no. 6, p. 50, 2009.
[2] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Proc. 14th Financial Cryptography Data Security*, 2010, pp. 136–149.
[3] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Comput.*, vol. 16, no. 1, pp. 69–73, Jan.-Feb. 2012.
[4] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Proc. 24th Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2005, pp. 457–473.
[5] R. Ostrovsky, A. Sahai, and B. Waters, "Attribute-based encryption with non-monotonic access structures," in *Proc. 14th ACM Conf. Comput. Commun. Security*, 2014, pp. 195–203.
[6] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *Proc. 29th Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2010, pp. 62–91.
[7] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proc. 13th ACM Conf. Comput. Commun. Security*, 2006, pp. 89–98.
[8] N. Attrapadung, B. Libert, and E. Panafieu, "Expressive key-policy attribute-based encryption with constant-size ciphertexts," in *Proc. 14th Int. Conf. Practice Theory Public Key Cryptography*, 2011, pp. 90–108.
[9] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Security Privacy*, 2007, pp. 321–334.
[10] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proc. 14th Int. Conf. Practice Theory Public Key Cryptography*, 2011, pp. 53–70.
[11] V. Goyal, A. Jain, O. Pandey, and A. Sahai, "Bounded ciphertext policy attribute based encryption," in *Proc. 35th Int. Colloquium Automata, Lang. Programm.*, 2008, pp. 579–591.
[12] R. Bobba, H. Khurana, and M. Prabhakaran, "Attribute-sets: A practically motivated enhancement to attribute-based encryption," in *Proc. 14th Eur. Symp. Res. Comput. Security*, 2009, pp. 587–604.
[13] M. Chase, "Multi-authority attribute based encryption," in *Proc. 4th Theory Cryptography Conf.*, 2007, pp. 515–534.
[14] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proc. 30th Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2011, pp. 568–588.
[15] 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.
[16] T. Pedersen, "A threshold cryptosystem without a trusted party," in *Proc. 10th Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 1991, pp. 522–526.
[17] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[18] M. Ito, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," *Electron. Commun. Japan (Part III: Fundam. Electron. Sci.)*, vol. 72, no. 9, pp. 56–64, 1989.
[19] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Proc. 28th Annu. Symp. Found. Comput. Sci.*, 1987, pp. 427–438.
[20] K. Yang, X. Jia, and K. Ren, "DAC-MACS: Effective data access control for multi-authority cloud storage systems," in *Proc. 32nd IEEE Int. Conf. Comput. Commun.*, 2013, pp. 2895–2903.
[21] S. Ruj, A. Nayak, and I. Stojmenovic, "Dacc: Distributed access control in clouds," in *Proc. 10th IEEE Int. Conf. Trust, Security Privacy Comput. Commun.*, 2011, pp. 91–98.
[22] K. Yang and X. Jia, "Expressive, efficient and revocable data access control for multi-authority cloud storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 7, pp. 1735–1744, Jul. 2013.
[23] K. Yang and X. Jia, "Attributed-based access control for multi-authority systems in cloud storage," in *Proc. IEEE 32nd Int. Conf. Distrib. Comput. Syst.*, 2012, pp. 536–545.
[24] M. Barua, X. Liang, R. Lu, and X. Shen, "ESPAC: Enabling security and patient-centric access control for ehealth in cloud computing," *Int. J. Security Netw.*, vol. 6, no. 2, pp. 67–76, 2011.
[25] T. Jung, X. Li, Z. Wan, and M. Wan, "Privacy preserving cloud data access with multi-authorities," in *Proc. 32nd IEEE Int. Conf. Comput. Commun.*, 2013, pp. 2625–2633.
[26] Z. Liu and Z. Cao, "On efficiently transferring the linear secret-sharing scheme matrix in ciphertext-policy attribute-based encryption," *IACR Cryptol. ePrint Archive*, vol. 2010, p. 374, 2010.
[27] S. Patil, P. Vhatkar, and J. Gajwani, "Towards secure and dependable storage services in cloud computing," *Int. J. Innovative Res. Adv. Eng.*, vol. 1, no. 9, pp. 57–64, 2014.
[28] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. 29th IEEE Int. Conf. Comput. Commun.*, 2010, pp. 1–9.
[29] S. Zarandioon, D. Yao, and V. Ganapathy, "K2c: Cryptographic cloud storage with lazy revocation and anonymous access," in *Proc. 8th Int. ICST Conf. Security Privacy Commun. Netw.*, 2012, pp. 59–76.
[30] J. Li, X. Huang, J. Li, X. Chen, and Y. Xiang, "Securely outsourcing attribute-based encryption with checkability," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 8, pp. 2201–2210, Aug. 2014.
[31] M. Chase and S. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *Proc. 16th ACM Conf. Comput. Commun. Security*, 2009, pp. 121–130.
[32] 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.
[33] L. Cheung and C. Newport, "Provably secure ciphertext policy abe," in *Proc. 14th ACM Conf. Comput. Commun. Security*, 2007, pp. 456–465.
[34] Y. Wu, Z. Wei, and H. Deng, "Attribute-based access to scalable media in cloud-assisted content sharing," *IEEE Trans. Multimedia*, vol. 15, no. 4, pp. 778–788, Jun. 2013.
[35] J. Hur, "Improving security and efficiency in attribute-based data sharing," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 10, pp. 2271–2282, Oct. 2013.
[36] Z. Wan, J. Liu, and R. Deng, "Hasbe: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 2, pp. 743–754, Apr. 2012.

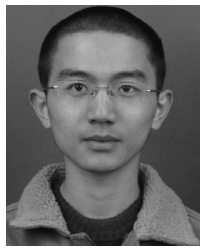


Wei Li received the BS degree from the Department of Information Security, University of Science and Technology of China (USTC), in 2014. He is currently a graduated student in communication and information system from the Department of Electronic Engineering and Information Science (EEIS), USTC. His research interests include network security protocol design and analysis.

Kaiping Xue received the BS degree from the Department of Information Security, in 2003 and the PhD degree from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 2007. Currently, he is an associate professor in the Department of Information Security and Department of EEIS, USTC. His research interests include next-generation Internet, distributed networks, and network security.



Yingjie Xue will receive the BS degree from the Department of Information Security, University of Science and Technology of China (USTC) in July, 2015. She will be a graduated student in communication and information system from the Department of Electronic Engineering and Information Science (EEIS), USTC. Her research interests include network security and cryptography.



Jianan Hong received the BS degree from the Department of Information Security, University of Science and Technology of China (USTC), in 2012. He is currently working towards the PhD degree in information security from the Department of Electronic Engineering and Information Science (EEIS), USTC. His research interests include secure cloud computing and mobile network security.