Secure, Efficient and Fine-Grained Data Access Control Mechanism for P2P Storage Cloud

Heng He, Ruixuan Li, *Member, IEEE*, Xinhua Dong, and Zhao Zhang, *Member, IEEE*

Abstract—By combining cloud computing and Peer-to-Peer computing, a P2P storage cloud can be formed to offer highly available storage services, lowering the economic cost by exploiting the storage space of participating users. However, since cloud severs and users are usually outside the trusted domain of data owners, P2P storage cloud brings forth new challenges for data security and access control when data owners store sensitive data for sharing in the trusted domain. Moreover, there are no mechanisms for access control in P2P storage cloud. To address this issue, we design a ciphertext-policy attribute-based encryption (ABE) scheme and a proxy re-encryption scheme. Based on them, we further propose a secure, efficient and fine-grained data Access Control mechanism for P2P storage Cloud named ACPC. We enforce access policies based on user attributes, and integrate P2P reputation system in ACPC. ACPC enables data owners to delegate most of the laborious user revocation tasks to cloud servers and reputable system peers. Our security analysis demonstrates that ACPC is provably secure. The performance evaluation shows that ACPC is highly efficient under practical settings, and it significantly reduces the computation overheads brought to data owners and cloud servers during user revocation, compared with other state-of-the-art revocable ABE schemes.

Index Terms—Cloud computing, peer-to-peer computing, access control, attribute-based encryption

1. Introduction

Cloud computing and Peer-to-Peer (P2P) computing are two of the Internet trends of the last decade, both of which are a form of large-scale distributed systems and have gained popularity in both research and industrial communities. Cloud computing [1] is a promising computing paradigm in which resources in the computing infrastructure are provided as services over the Internet by cloud service providers. The resources could be software, hardware, data storage, etc. With broadband Internet access, users are able to acquire these services for application. Cloud computing relies on large data-centers consisting of thousands of servers and all application processing and resources are centralized in data-centers. P2P computing [2] is a highly decentralized computing paradigm that leverages resources of participating users to support large-scale decentralized applications effectively. There are important differences between the two paradigms: the cloud provides highly available resources, but at an economic cost; P2P resources are almost free, but their availability is shaky. Several academic and commercial projects have proposed P2P storage cloud systems [3], [4], [5], [6] to combine the benefits of both paradigms, offering highly available storage services based on the cloud while lowering the cost by pooling the storage space of all participating users to provide a substantially storage. Potential applications include storage and backup systems [7], [8], [9], content distribution [10], [11], music or video streaming [12], [13], [14], and online gaming [15]. In recent years, there are several real-world P2P storage cloud systems that have been used worldwide, such as Symform [8], Wuala [9] and Spotify [12].

Security issues have been the top concerns in cloud computing [16]. When users store sensitive data in the cloud, maintaining confidentiality and privacy of the data becomes a challenging task. Since cloud servers are operated by commercial providers who are usually outside of the trusted domain of users, they are not entitled to access the confidential data. Furthermore, there are cases in which users (i.e. data owners) publish data in the cloud and need fine-grained data access control in terms of which users (i.e. data consumers) have the access privilege to which types of data. To achieve secure and flexible data sharing among a large number of users in P2P storage cloud, the security requirements are usually more complex. Data owners need to keep data confidential against cloud servers and all the users (i.e. peers) that contribute storage space to P2P cloud; and due to the dynamics of users, the access policies may have to be more complex. For example, in a real-world application scenario, the data owners can be individual users or organizations and may publish their various articles, books and magazines online through P2P cloud. The data consumers may subscribe to one or more kinds of these publications from the data owners. Since cloud servers do not have copyright on the publications, they should not be allowed to access the publications. Since users are enormous and may subscribe or unsubscribe at any time, it is very challenging to achieve data access control efficiently in such environment.

- *H. He, R. Li, and X. Dong are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China and H. He is also with the School of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, China. E-mail: heheng@wust.edu.cn, {rxli, xhdong}@hust.edu.cn.*
- *Z. Zhang is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011, USA. E-mail: zzhang@iastate.edu.*

Manuscript received 23 July 2014; revised 20 Oct. 2014; accepted 11 Nov. 2014. Date of publication 29 Dec. 2014; date of current version 30 Jan. 2015. Recommended for acceptance by D. Wei.
For information on obtaining reprints of this article, please send e-mail to: reprints.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2014.2378788

Data access control has been well studied and various techniques have been developed to specify differential access rights for individual users. Traditional access control models assume that the servers are fully trusted by data owners and let the servers enforce all access control policies. However, this assumption no longer holds in cloud computing. To achieve access control of data stored on untrusted cloud servers, a feasible solution would be storing encrypted data and disclosing decryption keys only to authorized users. In the literature, related mechanisms can be found in the areas of cryptographic file systems and access control of outsourced data. However, these mechanisms either lack fine-grainedness [17] or scalability [18] or do not provide adequate proof of data confidentiality [19], which makes them unsuitable for P2P storage cloud.

A recently proposed access control model, called attribute-based access control, defines access control policies based on the attributes of the user, environment, or the data. ABE [20] is a public-key cryptography primitive that was proposed to achieve the attribute-based access control on untrusted storage. Compared to previous works, ABE can achieve the fine-grainedness of data access control and meanwhile be efficient. In ABE, the complexity of encryption and key management are independent from the number of system users, and is just related to the number of system attributes. User revocation is an important issue in access control systems. However, it is hard to execute user revocation efficiently in ABE schemes since each attribute is usually shared by multiple users. To revoke attributes from a user, the data owner has to re-encrypt all the files associated with revoked attributes and update the secret keys for all the remaining users who share these attributes. Those operations would introduce heavy computation overheads on the data owner and may also require the data owner to be always online.

Related works have proposed schemes to exploit ABE technique to achieve attribute-based access control in data outsourcing systems [21], [22], [23]. To realize user revocation, Yu et al. [21], [22] integrated Proxy Re-Encryption (PRE) technique in their schemes, where the data owner can delegate most of laborious revocation tasks to cloud servers. Hur et al. [23] combined group key management algorithm based on a binary tree with ABE. However, these schemes are not designed for P2P cloud. When being applied directly in P2P cloud, they will be inefficient and bring heavy computation overheads to cloud servers because of the high dynamic of users and complex access policy, requiring an increase of the data owner's investment. To the best of our knowledge, there are still no relevant research works that address the issue of data access control in P2P storage cloud.

In this paper, we propose a secure, efficient and fine-grained data Access Control mechanism for P2P storage Cloud, namely ACPC. As [22], [23] do, we also exploit Ciphertext-Policy ABE (CP-ABE) technique for fine-grained data access control. However, unlike those schemes, in ACPC we design a novel CP-ABE scheme, namely PCCP-ABE. The new scheme is motivated by the CP-ABE scheme proposed by Ibraimi et al. [24], which is in turn inspired by [20], [25]. PCCP-ABE can support expressive access policy and is more efficient than state-of-the-art CP-ABE schemes. To efficiently address the important yet challenging issue of user revocation in P2P cloud, we further design a PRE scheme, namely PCPRE, which is an extension of PCCP-ABE with Proxy Re-Encryption, and integrate it into ACPC. By combining the schemes and a P2P reputation system [26], we enable the data owner to delegate file re-encryption to cloud servers and delegate user secret key update, the most computation intensive task, to reputable system peers picked up by P2P reputation system, without disclosing data contents and valid user secret keys. Consequently, the computation overheads brought to the data owner and cloud servers for user revocation are significantly reduced, which makes ACPC much more scalable than the previous revocable ABE schemes [21], [22], [23]. Furthermore, ACPC is provably secure under the standard security model, and can resist collusion attacks and protect user access privilege information effectively in the P2P cloud environment.

The rest of this paper is organized as follows. Section 2 gives some preliminaries related to ACPC. Section 3 describes ACPC in detail. Section 4 analyzes the security of ACPC and Section 5 evaluates its performance. Section 6 discusses related works and finally Section 7 concludes the paper.

2 Preliminaries

In this section, we first briefly present the technique preliminaries closely related to ACPC, and then present the system model and some security assumptions of ACPC.

2.1 CP-ABE

CP-ABE is a type of ABE schemes. ABE is an appealing technique in which data is encrypted under an access policy over a set of descriptive attributes. Since Bethencourt et al. [20] proposed the first CP-ABE scheme, more improved CP-ABE schemes have been proposed, e.g. [24], [25], [27] and [28]. In this paper, we consider an access policy that can be represented by an expressive access tree over attributes, similar to the one in [20], where each interior node is a threshold gate and the leaf nodes are associated with attributes. A user's secret key is associated with a set of these attributes, and one can decrypt the ciphertext with one's secret key only if the associated attributes satisfy the access tree. We give the definition of the access tree as follows.

Let T be an access tree representing an access policy. Each non-leaf node of T represents a threshold gate, described by its children and a threshold value. Let num_x be the number of children of a node x and k_x be its threshold value, then $0 < k_x \leq num_x$. When $k_x = 1$, the threshold gate is an OR gate and when $k_x = num_x$, it is an AND gate. Each leaf node x of T is described by an attribute and a threshold value $k_x = 1$. To facilitate working with the tree, we denote the parent of the node x in T by $parent(x)$. T also defines an ordering between the children of every node, that is, the children of a node are numbered from 1 to num. The function $index(x)$ returns such a number associated with the node x.

Let node ro be the root of T and T_x be the subtree of T rooted at the node x. Hence T is the same as T_{ro}. If a set S of attributes satisfies the access tree T_x, we denote it as $T_x(S) = 1$. We compute $T_x(S)$ as follows. If x is a non-leaf

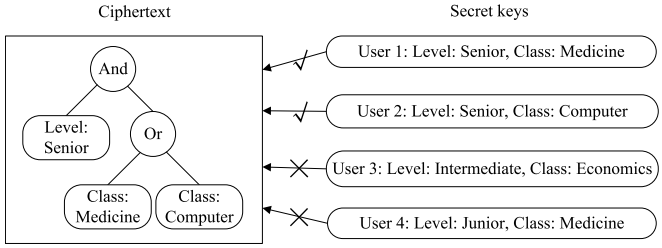


Fig. 1. An example of the access tree in CP-ABE.

node, evaluate $T_{x'}(S)$ for all children x' of x. $T_x(S)$ returns 1 if and only if at least k_x children return 1. If x is a leaf node, then $T_x(S)$ returns 1 if and only if the associated attribute $a_j \in S$. Thus, if T is satisfied by S, $T_{ro}(S) = 1$.

Fig. 1 presents an example that describes the access tree in CP-ABE. In the figure, only user 1 and user 2 can decrypt the ciphertext with their secret keys since the attributes associated with the secret keys satisfy the access tree of the ciphertext.

2.2 PRE

PRE [29] is a cryptographic primitive in which a semi-trusted proxy with the proxy re-encryption key $rk_{a\leftrightarrow b}$ can translate a ciphertext under public key PK_a into another ciphertext of the same plaintext under public key PK_b without knowing the plaintext, and vise versa.

2.3 Bilinear Pairing

Let G_0 and G_1 be two multiplicative cyclic groups of prime order p. Let g be a generator of G_0 and e be a bilinear pairing, $e : G_0 \times G_0 \rightarrow G_1$, which has the following properties:

1) Bilinear: for all $u, v \in G_0$, $a, b \in Z_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
2) Non-degeneracy: $e(g, g) \neq 1$.

We say that G_0 is a bilinear group if the group operation in G_0 and e are both efficiently computable. In a general implementation, G_0 is the group of points on an elliptic curve and G_1 denotes a multiplicative subgroup of a finite field. Please see Ref. [30] for more comprehensive description on how these groups and pairing are defined.

2.4 Secret Sharing

In Shamir's secret sharing scheme [31], a secret s is divided into m shares in such a way that any subset of k shares, where $k \leq m$, can together reconstruct the secret; no subset smaller than k can reconstruct the secret. The scheme utilizes a random polynomial q of degree $k-1$, where $q(l) \in Z_p$ and $q(0) = s$. Given any k shares $(l_0, q(l_0)),\ldots, (l_{k-1}, q(l_{k-1}))$, one can use Lagrange interpolation formulas as follows to recover $q(0)$:

$$q(0) = \sum_{i=0}^{k-1} \lambda_{l_i,L} q(l_i), \text{ where}$$

$$\lambda_{l_i,L} = \prod_{l_j \neq l_i, l_j \in L} \frac{l_j}{l_j - l_i}, \; L = \{l_0, \ldots, l_{k-1}\}.$$

2.5 P2P Reputation System

P2P reputation systems [26] are playing critical roles in P2P networks and they are mainly divided into three categories: peer reputation system, object reputation system and hybrid reputation system. In a peer reputation system, peers assign reputations to other peers according to their service quality and malicious peers with low reputation will be identified. In an object reputation system, peers assign reputations to the objects (files) they download regarding their authenticity and the object reputation is then used to decide whether the object should be downloaded. A hybrid reputation system combines the benefits of both strategies, by maintaining integrated reputation information of peers or objects, to determine which peers are the most secure and reliable at offering the best quality resources.

2.6 System Model and Assumptions

We utilize the architecture design of the P2P storage cloud systems proposed in the related works [3], [5], [13], which is composed of cloud servers and many system peers. The peers are participating users, which can be individual users or organizations. Every peer hosts a cloud peer fabric service built upon the Distributed Hash Table (DHT) based self-organizing routing structure, and thus they form a DHT overlay network [5]. Data storage, discovery and message routing are using the DHT overlay. Cloud servers maintain the meta-data of data files, including the identity, size and ciphertext of the encryption key of each file, and the current status of system peers, including their reputation ratings and performance parameters (i.e. computing power, contributing storage space, and online time). Cloud servers also perform the security and management mechanisms of the system. All the functions of cloud servers are achieved using service combination of IaaS and SaaS [1]. Since this paper focuses on the data access control mechanism for the P2P storage cloud based on this architecture, the details of the system design are omitted due to the space limit. More design details can be found in [3], [5] and [13].

Cloud servers are assumed always online and users can join or leave the system arbitrarily. We assume that cloud servers are honest but curious as [21], [23] do, that is, cloud servers honestly perform our proposed mechanism, but may actively attempt to gain file contents based on their inputs. An arbitrary number of users may collude to access files that they should not have access to. The communication channels between cloud servers and users are assumed secured under existing security protocols.

In P2P cloud, a peer's comprehensive reputation value is computed by its reputation rating from other peers and its computing power, contributing storage space and online time. We assume that there exists well-designed P2P reputation system in P2P cloud which can help us to select peers with high reputation values. For these selected peers, we assume that most of them are trustful, while a few of them may be semi-trustful or malicious. Trustful and semi-trustful peers will perform our proposed mechanism in general, while malicious peers will not. Semi-trusted peers and malicious peers may collude with revoked users to access files. Note that, for well-designed P2P reputation system, only trusted peers with legitimate behaviors can keep high reputation rating. How to design P2P reputation system exceeds the scope of this paper, which has been extensively studied.

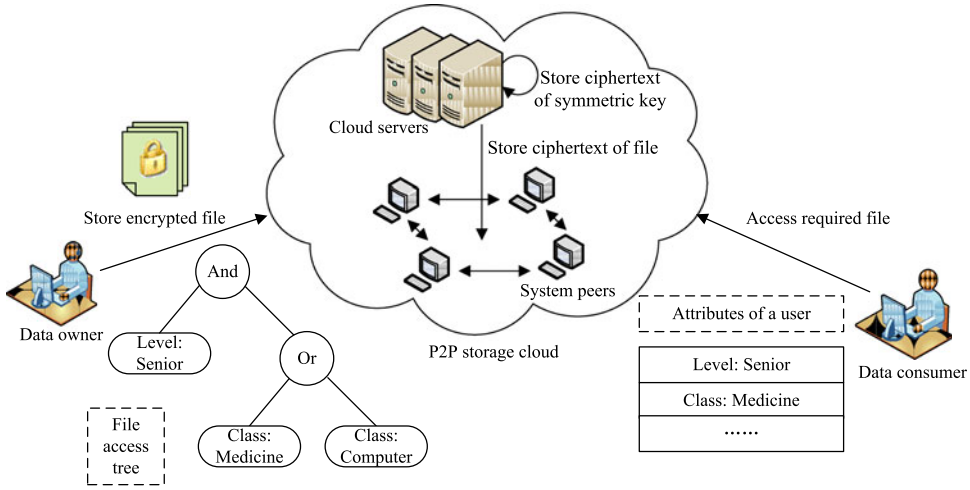


Fig. 2. An example of the process of data access control in ACPC.

3 Our Proposed Mechanism

In this section, we describe the full implementation of ACPC. First, we give the overview of ACPC, and then we present PCCP-ABE and PCPRE, which are the base of ACPC. Finally, we describe ACPC in detail.

3.1 Overview

In ACPC, we associate each file with an expressive access tree which is defined over attributes, and associate each user secret key with a set of these attributes. Since CP-ABE is inefficient when it is directly used to encrypt the data file, especially for the files of large size, the data owner encrypts the file using symmetric encryption algorithm first, and then encrypts the symmetric key with PCCP-ABE. The ciphertext of the symmetric key including the file access tree is stored in cloud servers, while the ciphertext of the file is stored in P2P cloud. The data consumer can decrypt the symmetric key using his secret key if the associated attributes satisfy the file access tree, and then he can decrypt the file using the symmetric key. Fig. 2 presents an example in a practical scenario that describes the process of data access control in ACPC. The data owner first defines attributes, next encrypts files using the above method, and then publishes them in P2P cloud. The data consumer applies for some attributes with a monetary cost from the owner and obtains the attributes and a secret key associated with these attributes. The data consumer can then access the required files. In the rest of this paper, we just use "user" to represent "data consumer" for simplicity.

User revocation is a challenging issue when we utilize this construction to enforce access control in P2P cloud, where user revocation may occur frequently and in different granularities. For instance, users can unsubscribe any attributes at any time in the example. When revoking a user, the data owner may require revoking all the attributes of the user, or just a subset of the attributes. Then he needs to re-encrypt all the files associated with revoked attributes and update secret keys for all the remaining users who also have these attributes. User revocation will introduce heavy computation overheads and cumbersome online requirements towards the data owner, if the data owner performs all these tasks.

To address this issue, we combine PCCP-ABE with PCPRE and P2P reputation system. It enables the data owner to delegate file re-encryption to cloud servers and delegate secret key update, the most computation intensive task, to the system peers with good reputations, picked out by the P2P reputation system. To further reduce the computation overheads on cloud servers and reputable peers, these tasks are delayed until users access the files, so that multiple revocation operations can be aggregated. Since some of these selected peers may not be fully trusted as they may collude with revoked users, we make multiple reputable peers cooperatively update the secret key for a user. Unless a revoked user can find enough colluding peers to update its secret key, he cannot gain the revoked access privilege. For well-designed P2P reputation system, it is reasonable to assume that very few of the selected peers are semi-trustful or malicious and it is quite difficult for a revoked user to find such colluding peers. We also make it easy to check whether the secret key is correctly updated by reputable peers. In addition, when the reputation value of a selected reputable peer decreases, it may be removed.

Through ACPC, we are able to provide fine-grained data access control in P2P cloud. Data confidentiality is achieved since cloud servers and users in the cloud are not able to learn the content of any file when they do not have the access privilege to the file. ACPC has the property of collusion resistance. It is secure against collusion attacks, in which colluding users may combine their secret keys to decrypt a file that they should not have access to. It can also effectively prevent collusion involving reputable peers who are not fully trusted. ACPC also protects user access privilege information against cloud servers. Moreover, ACPC can achieve efficiency and scalability benefit, in which, 1) the complexity of encryption and key management is independent to the number of users in the system, and is just related to the number of system attributes; and 2) the data owner can implement user revocation with minimal overheads in terms of computation effort and online time. The computation overheads brought to cloud servers are also significantly reduced.

3.2 PCCP-ABE

PCCP-ABE consists of four algorithms, namely setup, secret key generation, encryption and decryption. We describe them as follows.

PCCP-ABE setup algorithm, as shown in Algorithm 1, generates a system public key and a system master key, according to the attribute universe.

Algorithm 1. Setup

INPUT: Security parameter λ determining the size of the groups, attribute universe U.
OUTPUT: System public key PK, system master key MK.
1. Generate a bilinear group G_0 of prime order p with a generator g and a bilinear pairing e: $G_0 \times G_0 \rightarrow G_1$.
2. Randomly choose $\alpha \overset{R}{\leftarrow} Z_p^*$.
3. Compute $y = e(g, g)^{\alpha}$.
4. For each attribute $a_j \in$ U
   4.1 Randomly choose $t_{j1}, t_{j2} \overset{R}{\leftarrow} Z_p^*$.
   4.2 Compute $T_j = g^{t_{j1}t_{j2}/(t_{j1}+t_{j2})}$.
5. Return $PK = (e, g, y, \forall\, a_j \in U\text{: } T_j)$, $MK = (\alpha, \forall\, a_j \in U\text{: } (t_{j1}, t_{j2}))$.

For master key $MK = (\alpha, \forall\, a_j \in U\text{: } (t_{j1}, t_{j2}))$, public key $PK = (e, g, y, \forall\, a_j \in U\text{: } T_j)$, (t_{j1}, t_{j2}) is the master key component of attribute a_j, T_j is its public key component.

PCCP-ABE secret key generation algorithm, as shown in Algorithm 2, generates a user secret key associated with an attribute set, according to MK.

Algorithm 2. Secret key generation

INPUT: System master key $MK = (\alpha, \forall\, a_j \in U\text{: } (t_{j1}, t_{j2}))$, attribute set S.
OUTPUT: Secret key SK associated with S.
1. Randomly choose r $\overset{R}{\leftarrow} Z_P^*$.
2. Compute $d_0 = g^{\alpha - r}$.
3. For each attribute $a_j \in$ S
   3.1 Compute $d_{j1} = g^{r/t_{j1}}, d_{j2} = g^{r/t_{j2}}$.
4. Return $SK = (d_0, \forall\, a_j \in S\text{: } (d_{j1}, d_{j2}))$.

For secret key $SK = (d_0, \forall\, a_j \in S\text{: } (d_{j1}, d_{j2}))$, (d_{j1}, d_{j2}) is the secret key component of a_j.

PCCP-ABE encryption algorithm encrypts the data plaintext under its access tree, according to PK. A user can decrypt the ciphertext only when the attributes associated with the user's SK satisfy the access tree. The PCCP-ABE encryption algorithm is described in Algorithm 3.

Algorithm 3. Encryption

INPUT: System public key $PK = (e, g, y, \forall\, a_j \in U\text{: } T_j)$, plaintext M, access tree T over attribute universe U.
OUTPUT: Ciphertext C under T.
1. Randomly choose a secret s $\overset{R}{\leftarrow} Z_p^*$.
2. Compute $C_0 = g^s$, $C_1 = My^s = Me(g, g)^{as}$.
3. Let ro denote the root node of T.
4. For each node x in T (In a top-down manner, starting from ro)
   4.1 Let k_x denote the threshold value of x.
   4.2 If x is ro then
   Randomly choose a polynomial q_{ro} with degree $d_{ro} = k_{ro} - 1$, let $q_{ro}(0) = s$, and then divide s using secret sharing scheme, assigning each child node y a secret share $s_y = q_{ro}(index(y))$.
   4.3 Else
   Randomly choose a polynomial q_x with degree $d_x = k_x - 1$, let $q_x(0) = s_x = q_{parent}(x)(index(x))$, and for non-leaf node x, further divide s_x using secret sharing scheme, assigning each child node y a secret share $s_y = q_x(index(y))$.
5. Let X denote the set of leaf nodes in T.
6. For each leaf node x $\in$ X
   6.1 Let a_j denote the attribute associated with x.
   6.2 Compute $C_j = T_j^{q_x(0)} = g^{q_x(0)t_{j1}t_{j2}/(t_{j1}+t_{j2})}$.
7. Return $C = (T, C_0, C_1, \forall\, x \in X\text{: } C_j)$.

For ciphertext $C = (T, C_0, C_1, \forall\, x \in X\text{: } C_j)$, C_j is the ciphertext component of a_j. PCCP-ABE decryption algorithm decrypts C using SK. Only if the attributes associated with SK satisfy the access tree can the decryption be successful. The PCCP-ABE decryption algorithm is presented in Algorithm 4.

Algorithm 4. Decryption

INPUT: Ciphertext $C = (T, C_0, C_1, \forall\, x \in X\text{: } C_j)$, secret key $SK = (d_0, \forall\, a_j \in S\text{: } (d_{j1}, d_{j2}))$.
OUTPUT: Plaintext M or $\perp$.
1. For each leaf node x $\in$ X
   1.1 Let a_j denote the attribute associated with x.
   1.2 If $a_j \in$ S then compute

$$F_x = e(C_j, d_{j1}d_{j2}) = e(g^{q_x(0)t_{j1}t_{j2}/(t_{j1}+t_{j2})}, g^{r/t_{j1}}g^{r/t_{j2}}) = e(g^{q_x(0)t_{j1}t_{j2}/(t_{j1}+t_{j2})}, g^{r(t_{j1}+t_{j2})/t_{j1}t_{j2}}) = e(g,g)^{rq_x(0)}$$

   1.3 Else $F_x = \perp$.
2. For each non-leaf node x in T (In a down-top manner)
   2.1 Let S_x denote an arbitrary k_x-sized set of child nodes y such that $F_y \neq \perp$.
   2.2 If no such set exists then the node is not satisfied by S and $F_x = \perp$.
   2.3 Else compute (using Lagrange interpolation)

$$F_x = \prod_{y \in S_x} F_y^{\lambda_{i,S_x'}} = \prod_{y \in S_x} (e(g,g)^{rq_y(0)})^{\lambda_{i,S_x'}} = \prod_{y \in S_x} (e(g,g)^{rq_{parent(y)}(index(y))})^{\lambda_{i,S_x'}} = \prod_{y \in S_x} (e(g,g)^{rq_x(i)})^{\lambda_{i,S_x'}} = e(g,g)^{rq_x(0)},$$

   where $i = index(y)$, $S_x' = (\forall\, y \in S_x\text{: } index(y))$, and $\lambda_{i,S_x'}$ is the Lagrange coefficient.
3. Let ro denote the root node of T.
4. If $F_{ro} = \perp$ then T is not satisfied by S and return $\perp$.
5. Else
   5.1 Compute

$$e(C_0, d_0)F_{ro} = e(C_0, d_0)e(g,g)^{rq_{ro}(0)} = e(C_0, d_0)e(g,g)^{rs} = e(g^s, g^{\alpha - r})e(g,g)^{rs} = e(g,g)^{\alpha s}$$
$$M = C_1/e(g,g)^{\alpha s} = Me(g,g)^{\alpha s}/e(g,g)^{\alpha s}$$

   5.2 Return M.

3.3 PCPRE

PCPRE consists of three algorithms, namely attribute update, ciphertext component re-encryption, secret key component update. We describe them as follows.

PCPRE attribute update algorithm, as shown in Algorithm 5, updates an attribute by redefining its master key component and its public key component, and then generates a PRE key to update the old master key component and public key component.

Algorithm 5. Attribute update

INPUT: Master key component (t_{j1}, t_{j2}) of attribute a_j, public key component T_j of a_j.
OUTPUT: Updated master key component (t_{j_1}', t_{j_2}') of a_j, updated public key component T_j' of a_j, PRE key rk_j.
1. Randomly choose $t_{j1}', t_{j2}' \xleftarrow{R} Z_p^*$.
2. Compute $T_j' = g^{t_{j1}' t_{j2}'/(t_{j1}'+t_{j2}')}$, $rk_{j1\leftrightarrow j1'} = t_{j1}'/t_{j1}$, $rk_{j2\leftrightarrow j2'} = t_{j2}'/t_{j2}$, $rk_{j\leftrightarrow j'} = (t_{j1}' t_{j2}'/(t_{j1}'+t_{j2}'))\Big/(t_{j1}t_{j2}/(t_{j1}+t_{j2}))$.
3. Return (t_{j1}', t_{j2}'), T_j', $rk_j = (rk_{j1\leftrightarrow j1'}, rk_{j2\leftrightarrow j2'}, rk_{j\leftrightarrow j'})$.

For PRE key $rk_j = (rk_{j1\leftrightarrow j1'}, rk_{j2\leftrightarrow j2'}, rk_{j\leftrightarrow j'})$, it contains three components.

As shown in Algorithm 6, PCPRE ciphertext component re-encryption algorithm re-encrypts a ciphertext component of an attribute using the third component of the corresponding PRE key. The re-encrypted ciphertext component coincides with the latest master key component of the attribute. The corresponding user secret key can be used to decrypt the re-encrypted ciphertext only when the secret key component of the attribute is also updated.

Algorithm 6. Ciphertext component re-encryption

INPUT: Ciphertext component C_j of attribute a_j, PRE key component $rk_{j\leftrightarrow j^{(n)}}$.
OUTPUT: Re-encrypted ciphertext component $C_j^{(n)}$ of a_j.
1. Compute

$$C_j^{(n)} = (C_j)^{rk_{j\leftrightarrow j^{(n)}}} = (g^{q_x(0)t_{j1}t_{j2}/(t_{j1}+t_{j2})})^{(t_{j1}^{(n)}t_{j2}^{(n)}/(t_{j1}^{(n)}+t_{j2}^{(n)}))/(t_{j1}t_{j2}/(t_{j1}+t_{j2}))} = g^{q_x(0)t_{j1}^{(n)}t_{j2}^{(n)}/(t_{j1}^{(n)}+t_{j2}^{(n)})}$$

2. Return $C_j^{(n)}$.

As shown in Algorithm 7, PCPRE secret key component update algorithm updates a secret key component of an attribute using the first two components of the corresponding PRE key. The updated secret key component coincides with the latest master key component of the attribute, which makes the corresponding user secret key useful to decrypt the re-encrypted ciphertext.

Algorithm 7. Secret key component update

INPUT: One part of the secret key component d_{ji} of attribute a_j, PRE key component $rk_{ji\leftrightarrow ji^{(n)}}$, $i = 1, 2$.
OUTPUT: Updated secret key component $d_{ji}^{(n)}$ of a_j.
1. Compute

$$d_{ji}^{(n)} = (d_{ji})^{1/rk_{ji\leftrightarrow ji^{(n)}}} = (g^{r/t_{ji}})^{t_{ji}/_{ji}^{(n)}} = g^{r/t_{ji}^{(n)}}$$

2. Return $d_{ji}^{(n)}$.

3.4 ACPC

In this section, we describe the system operations in ACPC, which include system setup, new file creation, new user grant, user revocation, file access, and reputable peer removal.

3.4.1 System Setup

In this operation, the data owner defines the attribute universe U and calls Algorithm 1, which outputs the system public key PK and the system master key MK. It associates each public key component of an attribute with a version number for the purpose of achieving efficient user revocation. For attribute a_j, we use ver_j to denote the version number associated with its public key component T_j. Then the data owner sends (U, PK) to cloud servers.

Cloud servers store the received tuple. Then they pick out a set of reputable peers through P2P reputation system, and divide these peers into two subsets, i.e. SET_1 and SET_2. Note that no peer can exist in the two subsets simultaneously.

3.4.2 New File Creation

To store a data file in P2P cloud, the data owner performs as follows.

1) Select a unique identity ID_f for this file.
2) Randomly select a symmetric encryption key K, and encrypt the file with K using a symmetric encryption algorithm such as AES. We use E to denote the ciphertext of the file.
3) Define an access tree T over U for the file and encrypt K using Algorithm 3. We use C to denote the ciphertext of K, and then associate each ciphertext component of an attribute in C with a version number. For a_j, we use ver_{jc} to denote the version number associated with its ciphertext component C_j and set it as $ver_{jc} = ver_j$.
4) Send (ID_f, C, E, VER_C) to cloud servers, where VER_C denotes the set of version numbers associated with ciphertext components.

Cloud servers store the received tuple (ID_f, C, VER_C) and they store E distributedly in P2P cloud.

3.4.3 New User Grant

When a new user wants to join the system, the data owner assigns a set of attributes to the user, and generates a secret key associated with these attributes for the user as follows.

1) Assign the new user a set S of attributes.
2) Generate a secret key SK for this user using Algorithm 2. Then associate each secret key component of an attribute in SK with a version number. For a_j, we use ver_{jd} to denote the version number associated with its secret key component (d_{j1}, d_{j2}) and set it as $ver_{jd} = ver_j$.
3) Send $(S, SK, VER_D, \{T_j\}_{a_j \in s})$ to the user, where VER_D denotes the set of version numbers associated with secret key components.

The user stores the received tuple.

3.4.4 User Revocation

The data owner can revoke any subset of attributes from a user and we divide the user revocation into two stages.

In the first stage, the data owner performs attribute update for revoked attributes. Let R denote the subset of attributes to be revoked and ID_u denote the revoked user's identity. It performs as follows.

1) For each attribute a_j in R, call Algorithm 5 to generate the updated master key component (t_{j1}', t_{j2}'), the updated public key component T_j', and the corresponding PRE key $rk_j = (rk_{j1\leftrightarrow j1'}, rk_{j2\leftrightarrow j2'}, rk_{j\leftrightarrow j'})$, and set the version number associated with T_j' as $ver_j' = ver_j + 1$. Then replace each public key component T_j with T_j' and each master key component (t_{j1}, t_{j2}) with (t_{j1}', t_{j2}') to update PK and MK.
2) For each PRE key rk_j, associate a version number with it and we use ver_{jr} to denote the version number, setting it as $ver_{jr} = ver_j'$.
3) Send $(ID_u, R, \{T_j'\}_{a_j\in R}, \{rk_j\}_{a_j\in R}, \{ver_{jr}\}_{a_j\in R})$ to cloud servers.

On receiving this tuple, cloud servers update PK, store ID_u, R in the attribute revocation list ARL of the system, and store all the PRE keys and the associated version numbers. Every time a user revocation occurs, for each revoked attribute, a new PRE key will be generated and a version number will be associated with it. All the PRE keys of an attribute and the associated version numbers are stored in the attribute history list AHL of this attribute. Every attribute has its own AHL. Obviously, the version number associated with the next generated PRE key is always equal to the previous version number plus one.

3.4.5 File Access

This is also the second stage of user revocation. In this stage, cloud servers respond user request on file access, and re-encrypt the file and delegate selected reputable peers to update the user's secret key if necessary. Let ID_f denote the identity of the requested file, and other notations have the same meaning as those in previous sections. Cloud servers first re-encrypt C. For each attribute a_j associated with a leaf node in T, they perform as follows.

1) Check AHL of a_j and locate the PRE key component $rk_{j\leftrightarrow j'}$ with which the version number satisfies $ver_{jr} = ver_{jc}+1$. If no such PRE key exists, C_j has already coincided with the latest master key component of a_j and does not need to be re-encrypted; otherwise, perform the following steps to re-encrypt C_j.
2) If the latest master key component of a_j is $(t_{j1}^{(n)}, t_{j2}^{(n)})$, compute a single PRE key component

$$\begin{aligned} rk_{j\leftrightarrow j^{(n)}} &= rk_{j\leftrightarrow j'} \cdot rk_{j'\leftrightarrow j''} \cdots rk_{j^{(n-1)}\leftrightarrow j^{(n)}} \\ &= \left(t_{j1}^{(n)} t_{j2}^{(n)} / \left(t_{j1}^{(n)} + t_{j2}^{(n)}\right)\right) / (t_{j1} t_{j2} / (t_{j1} + t_{j2})). \end{aligned}$$

3) Call Algorithm 6 with $rk_{j\leftrightarrow j^{(n)}}$ to re-encrypt C_j as $C_j^{(n)}$, which coincides with $(t_{j1}^{(n)}, t_{j2}^{(n)})$, and set the version number associated with $C_j^{(n)}$ as $ver_{jc}^{(n)} = ver_{jr}^{(n-1)}$, where $ver_{jr}^{(n-1)}$ denotes the version number associated with $rk_{j^{(n-1)}\leftrightarrow j^{(n)}}$.

Then cloud servers send (C, VER_C) to the user. When receiving this tuple, the user performs as follows.

1) Check if S satisfies T and if true, perform the following steps; otherwise, the user cannot decrypt C.
2) For any attribute a_j in S, if there exists $ver_{jd} < ver_{jc}^{(n)}$, then pick out (d_{j1}, d_{j2}).
3) If no such attributes exist, the user's SK does not need to be updated and the user can decrypt C with SK to get K immediately by calling Algorithm 4. Otherwise, send $(S', \{(d_{j1}, d_{j2})\}_{a_j\in S'}, \{ver_{jd}\}_{a_j\in S'})$ to cloud servers, where S' denotes the set of all such attributes in S.

When receiving this tuple, cloud servers update SK for this user, for each attribute a_j in S', they perform as follows.

1) Check ARL if a_j is revoked from the user and if true, (d_{j1}, d_{j2}) will not be updated; otherwise, perform the following steps to update (d_{j1}, d_{j2}).
2) Check AHL of a_j and locate the PRE key components $rk_{j1\leftrightarrow j1'}$, $rk_{j2\leftrightarrow j2'}$ with which the version number satisfies $ver_{jr} = ver_{jd} + 1$.
3) If the latest master key component of a_j is $(t_{j1}^{(n)}, t_{j2}^{(n)})$, compute two PRE key components

$$\begin{aligned} rk_{j1\leftrightarrow j1^{(n)}} &= rk_{j1\leftrightarrow j1'} \cdot rk_{j1'\leftrightarrow j1''} \cdots rk_{j1^{(n-1)}\leftrightarrow j1^{(n)}} = t_{j1}^{(n)} / t_{j1}, \\ rk_{j2\leftrightarrow j2^{(n)}} &= rk_{j2\leftrightarrow j2'} \cdot rk_{j2'\leftrightarrow j2''} \cdots rk_{j2^{(n-1)}\leftrightarrow j2^{(n)}} = t_{j2}^{(n)} / t_{j2}. \end{aligned}$$

4) Randomly select a reputable peer P from SET_1 and send $(a_j, d_{j1}, rk_{j1\leftrightarrow j1^{(n)}})$ to P; randomly select another reputable peer Q from SET_2 and send $(a_j, d_{j2}, rk_{j2\leftrightarrow j2^{(n)}})$ to Q.

On receiving this tuple, P calls Algorithm 7 to generate the first part of the updated secret key component $d_{j1}^{(n)}$ that coincides with $t_{j1}^{(n)}$, and returns $(a_j, d_{j1}^{(n)})$ to cloud servers. Similarly, Q generates the second part of the updated secret key component $d_{j2}^{(n)}$ that coincides with $t_{j2}^{(n)}$ and returns $(a_j, d_{j2}^{(n)})$ to cloud servers.

On receiving the tuples from P, Q, cloud servers send $(S'', \{T_j^{(n)}\}_{a_j\in S''}, \{(d_{j1}^{(n)}, d_{j2}^{(n)})\}_{a_j\in S''})$ to the user, where S'' denotes the set of attributes of which the secret key components are updated, and $T_j^{(n)}$ denotes the latest public key component of a_j.

Note that in order to improve the system efficiency, for the secret key components of different users of the same attribute, cloud servers delegate some same reputable peers to update them.

On receiving the tuple from cloud servers, the user performs as follows.

1) Replace each public key component T_j with $T_j^{(n)}$ to update PK. For each secret key component $(d_{j1}^{(n)}, d_{j2}^{(n)})$, check if it satisfies the following condition

$$e\left(d_{j1}^{(n)} d_{j2}^{(n)}, T_j^{(n)}\right) = e(d_{j1} d_{j2}, T_j)$$

If true, $(d_{j1}^{(n)}, d_{j2}^{(n)})$ is correctly computed, so (d_{j1}, d_{j2}) is replaced with $(d_{j1}^{(n)}, d_{j2}^{(n)})$ to update SK. Then set the version number associated with $(d_{j1}^{(n)}, d_{j2}^{(n)})$ as $ver_{jd}^{(n)} = ver_{jc}^{(n)}$. For the secret key components that are not correctly computed, regain them from cloud servers.

2) Check if any attributes are revoked. If true, delete the revoked attributes, their public key components and their secret key components, and then check if the rest of the attributes satisfy T. If true, perform the following steps, otherwise the user cannot decrypt C. If no attributes are revoked, perform the following steps.
3) Decrypt C with SK to get K by calling Algorithm 4.
4) Download E from P2P cloud. Then decrypt E with K to get the file content.

3.4.6 Reputable Peer Removal

When the reputation value of a selected reputable peer decreases, cloud servers may remove it from SET_1 or SET_2, and make it unable to update secret keys for users any longer. Once the peer is removed, cloud servers notify the data owner to update attributes of which the secret key components have been updated by the peer.

Specifically, assume the peer belongs to SET_1 and let R' denote the set of attributes which need to be updated. The data owner performs the steps as described in the user revocation operation to update attributes in R' and sends $(R', \{T_j'\}_{a_j \in R'}, \{rk_j\}_{a_j \in R'}, \{ver_{jr}\}_{a_j \in R'})$ to cloud servers. Since the peer previously belonged to SET_1 and according to the steps of the file access operation, it only performed update for the first parts of the secret key components. Thus, for each attribute a_j in R', only t_{j1} needs to be updated and $rk_{j2 \leftrightarrow j2'} = 1$. Then cloud servers update PK and store all the PRE keys and the associated version numbers.

After being removed from SET_1, the peer cannot update any secret key components any longer, even if it recorded all the PRE key components sent to it previously.

4 Security Analysis

In this section, we first analyze the security of PCCP-ABE and the security of ACPC, and then give some security properties of ACPC.

4.1 Security of PCCP-ABE

Compared with Ibraimi et al.'s CP-ABE scheme [24], PCCP-ABE has different forms of system master key, system public key and user secret key, which are generated respectively as follows according to Section 3.2.1.

$$
\begin{aligned}
MK &= (\alpha, \forall a_j \in U : (t_{j1}, t_{j2})),\\
PK &= (e, g, y, \forall a_j \in U : T_j = g^{t_{j1} t_{j2}/(t_{j1}+t_{j2})}),\\
SK &= (d_0, \forall a_j \in S : (d_{j1} = g^{r/t_{j1}}, d_{j2} = g^{r/t_{j2}})).
\end{aligned}
$$

If we choose $t_j \xleftarrow{R} Z_p^*$ which satisfies $t_j = t_{j_1} t_{j_2}/(t_{j_1} + t_{j_2})$, we can convert MK, PK and SK as the following forms, which are the same as those in Ibraimi et al.'s scheme.

$$
\begin{aligned}
MK &= (\alpha, \forall a_j \in U : t_j),\\
PK &= (e, g, y, \forall a_j \in U : T_j = g^{t_j}),\\
SK &= (d_0, \forall a_j \in S : d_j = g^{r/t_j}).
\end{aligned}
$$

The following encryption and decryption of PCCP-ABE is the same as that of Ibraimi et al.'s scheme. Thus the security of PCCP-ABE is the same as that of Ibraimi et al.'s scheme. In [24], the scheme is proved secure under the selective-attribute model given that the Decisional Bilinear Diffie-Hellman (DBDH) problem is hard. Therefore, PCCP-ABE is secure under the same model.

4.2 Security of ACPC

We analyze the security of ACPC by comparing it with an intuitive mechanism, in which data files are encrypted using symmetric encryption keys, and these keys are then encrypted using PCCP-ABE. In this intuitive mechanism, just the ciphertexts of the files and the ciphertexts of the keys are given to cloud servers. Since the used symmetric encryption algorithm, such as AES, is secure, the security of this intuitive mechanism merely relies on the security of PCCP-ABE. In Section 4.1, we have shown that PCCP-ABE is secure under the selective-attribute model given that the DBDH problem is hard. Thus, the intuitive mechanism is secure under the same model. Compared to the intuitive mechanism, ACPC discloses the following extra information to cloud servers and reputable peers, namely the secret key components of attributes and the PRE keys. Our goal is to show that ACPC is as secure as the intuitive mechanism and we prove the security of ACPC using two semantic security games as follows.

Game 1. This is the security game of the intuitive mechanism, which is also the security game of Ibraimi et al.'s CP-ABE scheme [24].

Game 2. This game is for ACPC. The difference between this game and Game 1 is that, in this game more than one (MK, PK) pairs are defined, and an adversary is given all the public keys and allowed to query many user secret keys except for the ones that can decrypt the challenge ciphertext. In addition, the adversary is also given the secret key components of attributes and the PRE keys between any two (MK, PK) pairs.

Proof that ACPC is as secure as the intuitive mechanism. We assume that the adversary can use a polynomial time algorithm $A(C_A, \{PK_u\}_{1 \le u \le n}, \{SK_v\}_{1 \le v \le qs}, \{RK_u\}_{1 \le u \le n\text{-}1})$ to win Game 2 with non-negligible advantage, where C_A represents the challenge ciphertext, $\{PK_u\}$ and $\{SK_v\}$ denote the set of all the public keys and the set of all the secret keys given to the adversary, n and qs represent the number of public keys and the number of secret key queries, and $\{RK_u\}$ denotes the set of all the corresponding PRE keys. For a user secret key $SK = (d_0, \forall\ a_j \in S\text{: } (d_{j1}, d_{j2}))$, although the secret key components of attributes $\{(d_{j1}, d_{j2})\}$ may be disclosed to the adversary because of the update for SK, he cannot use them to decrypt the ciphertext since d_0 is kept secret by the user himself. The secret key components of attributes are actually equivalent to the secret keys queried by the adversary, which cannot decrypt the challenge ciphertext, and thus we use $\{SK_v\}$ to also include these secret key

components. For a PRE key $rk_j = (rk_{j1\leftrightarrow j1'}, rk_{j2\leftrightarrow j2'}, rk_{j\leftrightarrow j'})$, reputable peers use $rk_{j1\leftrightarrow j1'}$, $rk_{j2\leftrightarrow j2'}$ to update d_{j1}, d_{j2} for the user to decrypt the re-encrypted ciphertext, while cloud servers can also use $rk_{j\leftrightarrow j'}$ to update $d_{j1}d_{j2}$ to achieve the same purpose. Thus, we just use $rk_{j\leftrightarrow j'}$ to represent the PRE key. Then, we can use Algorithm 8 to build a polynomial time algorithm that can win Game 1 with non-negligible advantage.

Algorithm 8. Polynomial time algorithm

INPUT: Challenge ciphertext $C_B = (T, C_0, C_1, \forall x \in X: C_j)$, system public key $PK = (e, g, y, \forall a_j \in U: T_j)$, secret keys given to the adversary $\{SK_v'\}_{1 \le v \le qs} = \{(d_{v0}, \forall a_j \in S_v: (d_{vj1}, d_{vj2}))\}_{1 \le v \le qs}$.

OUTPUT: Polynomial time algorithm A.

1. For each attribute $a_j \in U$
   1.1 For $u = 1$ to n
   Randomly choose $r_{uj} \xleftarrow{R} Z_p^*$.
2. Compute $C_A = (T, C_0, C_1, \{\forall x \in X : C_j^{r_{uj}}\}_{1 \le u \le n})$.
3. For $u = 1$ to n
   3.1 Compute $PK_u = (e, g, y, \forall a_j \in U : T_j^{r_{uj}})$.
4. For $v = 1$ to qs
   4.1 For $u = 1$ to n
   Compute $d_{vuj}=(d_{vj1}d_{vj2})^{1/r_{uj}}$.
   Separate d_{vuj} into two shares d_{vuj1}, d_{vuj2} where d_{vuj1} $d_{vuj2} = d_{vuj}$.
   Compute $SK_{vu} = (d_{v0}, \forall a_j \in S_v : (d_{vuj1}, d_{vuj2}))$.
   4.2 Compute $SK_v = \{SK_{vu}\}_{1 \le u \le n}$.
5. For $u = 1$ to n-1
   5.1 Compute $rk_{uj\leftrightarrow(u+1)j} = r_{(u+1)j}/r_{uj}$.
   5.2 Compute $RK_u = (\forall a_j \in U : rk_{uj\leftrightarrow(u+1)j})$.
6. Return $A(C_A, \{PK_u\}_{1 \le u \le n}, \{SK_v\}_{1 \le v \le qs}, \{RK_u\}_{1 \le u \le n\text{-}1})$.

Therefore, the advantage of the adversary in Game 2 is not higher than that in Game 1. On the other hand, the advantage of the adversary in Game 2 cannot be lower than that in Game 1 since the adversary is given more information in Game 2 than in Game 1. Thus, the advantages in the two games are the same and ACPC is as secure as the intuitive mechanism, which is provably secure. This proves that the data confidentiality of ACPC and cloud servers and users who do not have the access privilege to a file are not able to learn the file content.

4.3 Collusion Resistent

To prevent collusion attacks, Algorithm 2 generates a random value r for each user, which is embedded in each secret key component of the user. To decrypt a ciphertext, a user must recover $e(g, g)^{\alpha s}$. To do that he must first recover $e(g, g)^{rs}$, which requires him to have the secret key components with the same random value r. Colluding users who combine their secret key components would not extend their decryption power since different users have different random values in their secret keys. This means that different users cannot combine their secret keys and decrypt a file that the colluding users should not have access to. Thus, PCCP-ABE can resist collusion attacks from an arbitrary number of colluding users, and according to Section 4.2, we can conclude that ACPC are secure against collusion attacks in which an adversary can obtain multiple user secret keys.

In ACPC, we delegate secret key update to reputable peers picked out by P2P reputation system. However, some of these peers may be not fully trusted and may collude with revoked users, updating secret key components for them to access files. To prevent such collusion, we randomly select a pair of peers to cooperatively update the secret key component of an attribute. Unless a revoked user can find enough colluding peers to update the secret key components of all his revoked attributes, he cannot gain the revoked access privilege. For a well-designed P2P reputation system, it is reasonable to assume that very few of the selected peers are semi-trustful or malicious. Thus, it is quite impossible that a revoked user can find out enough such colluding peers which can update secret key for him and therefore ACPC is secure against such collusion.

Of course, we can make more than two peers update a secret key component to make the mechanism more secure against such collusion. For example, if we make three peers update a secret key component, then in PCCP-ABE, MK, PK, SK should be computed as

$$MK = (\alpha, \forall a_j \in U : (t_{j1}, t_{j2}, t_{j3})),$$
$$PK = (e, g, y, \forall a_j \in U : T_j = g^{t_{j1}t_{j2}t_{j3}/(t_{j1}t_{j2}+t_{j1}t_{j3}+t_{j2}t_{j3})}),$$
$$SK = (d_0, \forall a_j \in S : (d_{j1} = g^{r/t_{j1}}, d_{j2} = g^{r/t_{j2}}, d_{j3} = g^{r/t_{j3}})).$$

The encryption and decryption in this case is just similar as that in Section 3.2.1, whereas three peers will be used to cooperatively update a secret key component in the user revocation. Although here the mechanism is more secure, more computation overheads are brought to the selected reputable peers. When there exists a well-designed P2P reputation system in P2P cloud, it may not be necessary to adopt such a more complicated mechanism.

4.4 User Access Privilege Confidentiality

Although the access tree of a file is disclosed to cloud servers, the attributes of users are unknown to them and only when user revocation occurs, a user will disclose the attributes of which the secret key components need to be updated to cloud servers. Thus, it is hard for cloud servers to obtain the access privilege information of users unless they continually accumulate user attributes, which is contrary to the assumption of cloud servers in Section 2.5.

In addition, the data owner can change every attribute into an encrypted form and distribute them to cloud servers and users, which makes the meanings of attributes vague and furhter protects the privacy of access policies.

5 Performance Evaluation

In this section, we numerically evaluate the performance of ACPC in terms of computation overheads and storage overheads respectively. We preserve the notations of previous sections here.

5.1 Computation Overhead Evaluation

In this section, we evaluate the computation overheads of ACPC, compare computation overheads of different CP-ABE schemes, and measure the computation time of ACPC. We use C_0 and C_1 to denote the exponentiation in group G_0 and in group G_1 respectively, and use C_e to denote bilinear pairing.

TABLE 1
Comparison of Computation Overheads of CP-ABE Schemes

Algorithms	Secret key generation	Encryption	Decryption
BSW [20]	$(2\|S\|+2)C_0$	$(2\|X\|+1)$ C_0+C_1	$(2\|S\|+1)C_e+$ $(\|S\|+\|X'\|)C_1$
W [27]	$(\|S\|+2)C_0$	$(4\|X\|+1)$ C_0+C_1	$(2\|S\|+1)C_e+\|S\|C_1$
HN [23]	$(2\|S\|+2)C_0$	$(3\|X\|+1)$ C_0+C_1	$(2\|S\|+1)C_e+(\|S\|+$ $\|X'\|)C_1+\|S\|C_0$
PCCP-ABE	$(2\|S\|+1)C_0$	$(\|X\|+1)$ C_0+C_1	$(\|S\|+1)C_e$ $+(\|S\|+\|X'\|)C_1$

TABLE 2
Time to Generate SK as a Function of the Number of Attributes Associated With SK (Unit: s)

Attributes associated with *SK*	PCCP-ABE	BSW [20]	W [27]	HN [23]
1	0.015	0.020	0.015	0.020
10	0.105	0.155	0.060	0.155
20	0.205	0.304	0.110	0.304
30	0.304	0.454	0.160	0.454
40	0.404	0.604	0.210	0.604
50	0.504	0.753	0.259	0.753

5.1.1 Computation Overheads of ACPC

We evaluate the computation overhead introduced by each operation in ACPC.

System setup. In this operation, the data owner needs to define underlying bilinear groups, and generates *PK* and *MK*. The main computation overhead is $|U|C_0+C_1+C_e$.

New file creation. In this operation, the data owner needs to encrypt the data file using *K* and generate *C* of *K*. For the former, the computation overhead depends on the size of the file and the used symmetric encryption algorithm. For the latter, the main computation overhead is $(|X|+1)C_0+C_1$.

New user grant. In this operation, the data owner needs to generate *SK* for the user. The main computation overhead is $(2|S|+1)C_0$.

User revocation. This operation is composed of two stages. Here we just count the computation overhead of the first stage. That for the second stage will be included in the file access operation. In the first stage, the data owner performs attribute update for the attributes revoked from a user and the main computation overhead is $|R|C_0$.

File access. This is also the second stage of user revocation. In this stage, cloud servers respond the user request on file access. They may need to re-encrypt *C*. In the worst case, the main computation overhead is $|X|C_0$. In addition, reputable peers are responsible for updating *SK* for the user. Cloud servers delegate two reputable peers to generate an updated secret key component, and the main computation overhead for each peer is C_0. In the worst case, cloud servers need to delegate reputable peers to generate $|S|$ updated secret key components, and thus the whole computation overhead is about $2|S|C_0$. To get the file, the user needs to first decrypt *C*, and the computation overhead is $(|S|+1)C_e+(|S|+|X'|)C_1$ in the worst case, where *X'* denotes the set of interior nodes of *T* that are satisfied. Then, the user decrypts *E* using *K* and the computation overhead depends on the size of the file and the used symmetric encryption algorithm.

Reputable peer removal. In this operation, the data owner needs to perform attribute update for the attributes of which the secret key components have been updated by the removed peer. The main computation overhead is $|R'|C_0$.

5.1.2 Comparison of Computation Overheads of CP-ABE Schemes

In Table 1, we compare PCCP-ABE with two CP-ABE schemes proposed by Bethencourt et al. [20] and Waters [27], and Hur et al.'s [23] revocable CP-ABE scheme, in terms of the computation overheads of the main algorithms. These schemes can support the same expressive access policy.

5.1.3 Computation Time Measurement of ACPC

We measure the computation time of the operations in ACPC on Intel Core 2 Duo 2.40 GHZ processor machines with Ubuntu 12.10 operating system. The main algorithms are developed on CP-ABE toolkit [32] and PBC library [33], in which we use a 160-bit elliptic curve group based on the supersingular curve $y^2=x^3+x$ over a 512-bit finite field. On the test machine, C_0, C_1 and C_e are computed in approximately 4.99 ms, 0.58 ms and 4.98 ms, respectively. All the time results are averaged over 50 runs. The computation overheads of ACPC are closely related to the access control scale and the relevant factors, such as the number of user attributes, the number of attributes in file access tree, the number of total users, etc., will be specified in the following measurements.

In Tables 2, 3, and 4, we show the computation time of secret key generation, encryption and decryption in PCCP-ABE on practical settings respectively, which are involved in the operations in ACPC, and compare it with that in Bethencourt et al.'s scheme [20], Waters' scheme [27] and Hur et al.'s scheme [23].

Table 2 shows the time to generate *SK* as a function of the number of attributes associated with *SK*, and Table 3 shows the time to encrypt *K* as a function of the number of leaf nodes in *T* in the four schemes. As expected, the key generation time and encryption time in the four schemes are all linear with respect to the number of attributes with *SK* and the number of leaf nodes in *T*. Among the four schemes, PCCP-ABE has medium key generation time and minimum encryption time.

TABLE 3
Time to Encrypt K as a Function of the Number of Leaf Nodes in T (Unit: s)

Leaf nodes in *T*	PCCP-ABE	BSW [20]	W [27]	HN [23]
1	0.011	0.016	0.026	0.021
20	0.105	0.205	0.405	0.305
40	0.205	0.405	0.804	0.604
60	0.305	0.604	1.203	0.904
80	0.405	0.804	1.602	1.203
100	0.505	1.004	2.002	1.503

TABLE 4
Time to Decrypt K as a Function of the Number of Attributes Associated With SK (unit: s)

Attributes associated with *SK*	PCCP-ABE	BSW [20]	W [27]	HN [23]
1	0.010	0.015	0.016	0.020
10	0.065	0.115	0.110	0.165
20	0.127	0.226	0.216	0.326
30	0.188	0.337	0.321	0.487
40	0.249	0.449	0.427	0.648
50	0.311	0.560	0.532	0.809

Table 4 shows the time to decrypt K as a function of the number of attributes associated with SK in the four schemes. The decryption time depends on the specific T and the attributes available in SK. In the table, we measure the time in the worst case where all the attributes in SK are used for decryption, T is a complete binary tree in which each leaf node is associated with a different attribute in SK, and each non-leaf node is an AND gate. The decryption time in the four schemes is linear with respect to the number of attributes with SK, and PCCP-ABE has the minimum decryption time. In real-world cases, T may be of various forms and a user often uses partial secret key components. Thus, the actual decryption time is usually much shorter than the time in Table 4.

In all these tables, even for the largest problem size, the operations in ACPC can finish in a short time. Thus ACPC can support large user secret keys and access trees in practice while maintaining reasonable running time. In addition, compared with the other three schemes, PCCP-ABE can support user revocation in ACPC effectively.

In Tables 5, 6, and 7, we show the computation time of the data owner and cloud servers for user revocation scheme in ACPC under different practical settings. We compare it with that in an alternative scheme where the data owner delegates both file re-encryption and user secret key update to cloud servers (scheme 1), plus another scheme where the data owner performs all the revocation tasks himself (scheme 2). Note that scheme 1 is also the user revocation approach in Yu et al.'s schemes [21], [22]. For clarity, we consider the case of a file being accessed by many users.

Table 5 shows the time to revoke users as a function of the number of re-encrypted ciphertext components in the three schemes, where the number of users is fixed at 300

TABLE 5
Time to Revoke Users as a Function of the Number of Re-Encrypted Ciphertext Components (unit: s)

Re-encrypted ciphertext components	DO in ACPC	CS in ACPC	DO in scheme 1	CS in scheme 1	DO in scheme 2
5	0.025	0.025	0.025	4.516	4.541
10	0.050	0.050	0.050	4.541	4.591
20	0.100	0.100	0.100	4.591	4.691
30	0.150	0.150	0.150	4.641	4.790
40	0.200	0.200	0.200	4.691	4.890
50	0.250	0.250	0.250	4.741	4.990

DO denotes data owner and CS denotes cloud servers.

TABLE 6
Time to Revoke Users as a Function of the Number of Updated Secret Key Components per User (unit: s)

Updated secret key components per user	DO in ACPC	CS in ACPC	DO in scheme 1	CS in scheme 1	DO in scheme 2
1	0.075	0.075	0.075	1.572	1.647
2	0.075	0.075	0.075	3.069	3.144
4	0.075	0.075	0.075	6.063	6.138
6	0.075	0.075	0.075	9.057	9.132
8	0.075	0.075	0.075	12.051	12.126
10	0.075	0.075	0.075	15.045	15.120

and the number of updated secret key components is 3 on average for each user. In ACPC, when revoking users, the data owner just needs to perform attribute update for the revoked attributes, and cloud servers just need to re-encrypt the ciphertext components of revoked attributes. The computation time of the data owner and that of cloud servers are the same. As the number of re-encrypted ciphertext components increases, they both remain very short, since user secret key update, the most computation intensive task, is delegated to reputable peers. In scheme 1, the data owner needs to perform attribute update for revoked attributes, and the computation time is the same as that in ACPC. However, since cloud servers are delegated with both file re-encryption and user secret key update, their computation time is quite long. In scheme 2, the data owner needs to do all the tasks, and therefore the computation time of the data owner is even longer.

Table 6 shows the time to revoke users as a function of the number of updated secret key components per user in the three schemes, where the number of users is fixed at 300 and the number of re-encrypted ciphertext components is 15. The computation time of the data owner and cloud servers in ACPC and that of the data owner in scheme 1 are the same. They are all independent of the updated secret key components per user and remain to be constant. The computation time of cloud servers in scheme 1 and that of the data owner in scheme 2 are both linear with respect to the number of updated secret key components per user and grow very quickly.

Table 7 shows the time to revoke users as a function of the number of total users in the three schemes, where the number of re-encrypted ciphertext components is fixed at 15 and the number of updated secret key components per user is 3. The computation time of the three schemes in this

TABLE 7
Time to Revoke Users as a Function of the Number of Total Users (unit: s)

Total users	DO in ACPC	CS in ACPC	DO in scheme 1	CS in scheme 1	DO in scheme 2
100	0.075	0.075	0.075	1.572	1.647
200	0.075	0.075	0.075	3.069	3.144
400	0.075	0.075	0.075	6.063	6.138
600	0.075	0.075	0.075	9.057	9.132
800	0.075	0.075	0.075	12.051	12.126
1000	0.075	0.075	0.075	15.045	15.120

TABLE 8
Comparison of Storage Overheads of CP-ABE Schemes

Keys and ciphertext	*PK*	*MK*	*SK*	*C*
BSW [20]	$3L_0 + L_1$	$L_0 + L_p$	$(2\|S\| + 1)L_0$	$(2\|X\| + 1)$ $L_0 + L_1$
W [27]	$(\|U\| + 2)$ $L_0 + L_1$	L_0	$(\|S\| + 2)L_0$	$(2\|X\| + 1)$ $L_0 + L_1$
HN [23]	$3L_0 + L_1$	$L_0 + L_p$	$(2\|S\| + 1)L_0$ $+ (\text{Log}_2 N + 1)\, L_p$	$(2\|X\| + 1)$ $L_0 + L_1$
PCCP-ABE	$(\|U\| + 1)$ $L_0 + L_1$	$(2\|U\| + 1)L_p$	$(2\|S\| + 1)L_0$	$(\|X\| + 1)$ $L_0 + L_1$

case is the same as that in Table 6, except that the computation time of cloud servers in scheme 1 and that of the data owner in scheme 2 are both linear with respect to the number of users.

Note that the above measurement is taken on a single user revocation. In practical P2P cloud scenarios, revocation may occur quite frequently. Thus, ACPC can achieve much more efficiency benefit when compared with other related mechanisms. From all the above experimental results, we can conclude that the computation overhead introduced by each operation in ACPC is very small and ACPC keeps highly efficient and scalable even when users are large-scale and highly dynamic. Hence, ACPC is very suitable for P2P cloud.

5.2 Storage Overhead Evaluation

In this section, we evaluate the storage overheads of ACPC and make comparison of the storage overheads of different CP-ABE schemes. We use L_0, L_1 and L_p to denote the size of an element in G_0, G_1 and Z_p respectively, and use N to denote the number of all users in the system.

5.2.1 Storage Overheads of ACPC

We evaluate storage overheads for the data owner, a user and cloud servers respectively. A data owner needs to keep *PK* and *MK*. The size of *PK* is $(|U| + 1)L_0 + L_1$ and the size of *MK* is $(2|U| + 1)L_p$. A user needs to keep $\{T_j\}_{a_j \in s}$ and *SK*. The size of $\{T_j\}_{a_j \in s}$ is $|S|L_0$ and the size of *SK* is $(2|S| + 1)L_0$. Cloud servers need to keep *C*, *PK*, PRE keys. The size of *C* is $(|X| + 1)L_0 + L_1$ and the size of each PRE key is just $3L_p$. For a 160-bit elliptic curve group over a 512-bit finite field, L_0, L_1, L_p are 40 B, 64 B, 20 B respectively. Thus, the storage overheads are small in actual cases described in Section 5.1.3., and ACPC can support large user secret keys, access trees and attribute universe which contains many attributes.

5.2.2 Comparison of Storage Overheads of CP-ABE Schemes

In Table 8, we compare PCCP-ABE with Bethencourt et al.'s scheme [20], Waters' scheme [27] and Hur et al.'s scheme [23], in terms of the size of keys and ciphertext. Among the four schemes, PCCP-ABE has the smallest *C*, medium-sized *SK* and relatively large *PK* and *MK*. Since PCCP-ABE can support user revocation in ACPC effectively, such forms of keys are necessary.

6 Related Works

In this section, we focus on the works of ciphertext access control schemes which are closely related to our research.

Kallahalla et al. [17] designed a cryptography-based file system for secure sharing of data on untrusted servers, where each file is encrypted with a symmetric key. The key is further encrypted with a file-group key. When the data owner shares a file group with a set of users, he just distributes the file-group key to them. The system is not suitable for fine-grained access control since the complexity of key management is proportional to the total number of file groups that could be huge. Ateniese et al. [19] proposed a secure distributed storage scheme based on PRE. In their scheme, using a master private key and an authorized user's public key, the data owner generates PRE keys. A semi-trusted server then uses the PRE keys to translate a ciphertext into a form that can be decrypted by the authorized user. The scheme requires that there is no collusion between the server and any malicious user since a single such collusion would expose decryption keys of all the encrypted files. Vimercati et al. [18] proposed a scheme for securing data on semi-trusted storage servers based on key derivation methods. In their scheme, the data owner creates tokens from which, together with a user's secret key, the user can derive decryption keys of desired files. The data owner then transmits these tokens to the semi-trusted server and the server conducts a second level of encryption to enforce access control of the data. As the complexity of the file creation and the user grant and revocation is linear to the user number, the scheme is not scalable.

Yu et al. [21] proposed an attribute-based data access control scheme in the cloud that exploits Key-Policy ABE (KP-ABE) and PRE. When applied in data access control in P2P cloud where users are large-scale and highly dynamic, it may introduce cumbersome user key management towards the data owner, as the owner needs to manage all the user access structures. The scheme also brings heavy computation overheads to cloud servers, as the data owner delegates all the revocation tasks of file re-encryption and user secret key update to cloud servers, and thus increases the data owner's investment. Moreover, how to revoke partial access privilege of a user is not discussed. Yu et al. [22] also proposed a scheme that combines CP-ABE with PRE to achieve the same goal. The scheme is based on the CP-ABE scheme proposed by Cheung et al. [25] which only supports access policy of AND gates and is inefficient. In addition, the scheme also brings heavy computation overheads to servers. Hur et al. [23] proposed a fine-grained access control scheme on the outsourced data that combines CP-ABE and group key management algorithm. It requires a trusted authority and encumbers the data service manager with complicated group key distribution. Since the algorithm uses a binary tree of which each leaf node represents a user and enforces user-level access control per attribute group, it may cause considerable computation and communication overheads for user revocation especially applied in P2P cloud. Moreover, this scheme is vulnerable to collusion attacks between revoked users and users who have partial attributes of a file access tree. Zhao et al. [34] proposed a novel data sharing protocol for cloud storage by combining ABE and Attribute-Based Signature (ABS). This scheme ensures the confidentiality of

outsourced data, and also provides the data integrity verification. However, user revocation is not discussed. Tysowski et al. [35] considered a specific cloud computing environment where data are accessed by resource-constrained mobile devices, and proposed novel modifications to ABE, which assigned the higher computational overhead of cryptographic operations to the cloud provider and lowered the total communication cost for the mobile user.

In this paper, we extended CP-ABE and proposed ACPC with PCCP-ABE and PCPRE schemes to address the challenging issue of data access control in P2P cloud. The P2P reputation system was integrated into ACPC, which enables data owners to delegate most of the laborious user revocation tasks to cloud servers and reputable system peers. ACPC is provably secure, and is demonstrated to be more efficient and scalable than state-of-the-art revocable ABE schemes.

7 Conclusion

This paper aims at providing secure, efficient and fine-grained data access control in P2P storage cloud, which is not supported by current works. To achieve this goal, we design an efficient CP-ABE scheme and a corresponding PRE scheme, i.e., PCCP-ABE and PCPRE, and then propose ACPC based on those schemes. To efficiently address the issue of user revocation, in ACPC we integrate P2P reputation system and enable the data owner to delegate file re-encryption to cloud servers and delegate user secret key update, the most computation intensive task, to the reputable system peers picked out by P2P reputation system. Moreover, ACPC is provably secure under the standard security model and can resist collusion attacks and protect user access privilege information effectively. Our future works include deploying ACPC in real-world P2P storage cloud systems and designing specific scheduling schemes for reputable peers in such real systems.

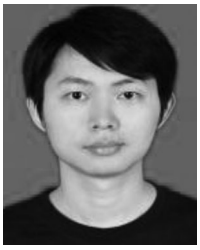
Acknowledgments

This work was supported by National Natural Science Foundation of China under Grants 61173170, 61300222, 61303117, 61433006 and U1401258, Innovation Fund of Huazhong University of Science and Technology under Grants 2013QN120, 2012TS052 and 2012TS053, and Young Scientist Foundation of Wuhan University of Science and Technology under Grant 2013xz012. Ruixuan Li is the corresponding author of the article.

References

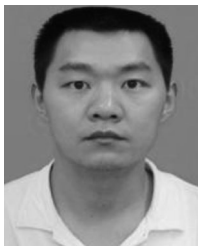
[1] P. Mell and T. Grance, "The NIST definition of cloud computing," *NIST Special Publication* 800-145, pp. 1–7, Sep. 2011.
[2] R. Rodrigues and P. Druschel, "Peer-to-peer systems," *Commun. ACM*, vol. 53, no. 10, pp. 72–82, Oct. 2010.
[3] J. Li and Q. Huang, "Erasure resilient codes in peer-to-peer storage cloud," in *Proc. IEEE Int. Conf. Acoustics, Speech Signaling Process.*, 2006, pp. 14–19.
[4] H. Kavalionak and A. Montresor, "P2P and cloud: A marriage of convenience for replica management," in *Proc. 6th IFIP TC 6 Int. Conf. Self-Organizing Syst.*, 2012, pp. 60–71.
[5] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, "Peer-to-peer cloud provisioning: Service discovery and load-balancing," in *Cloud Comput.: Principles, Systems and Applications*, Part 2, N. Antonopoulos, L. Gillam, Ed. London: Springer, pp. 195–217, May 2010.
[6] L. Bremer and K. Graffi, "Symbiotic coupling of P2P and cloud systems: The wikipedia case," in *Proc. IEEE Int. Conf. Commun.*, 2013, pp. 3444–3449.
[7] Z. Yang, B. Zhao, Y. Xing, S. Ding, F. Xiao, and Y. Dai, "Amazingstore: Available, low-cost online storage service using cloudlets," in *Proc. 9th Int. Workshop Peer-to-Peer Syst.*, 2010, pp. 2–2.
[8] Symform web site. (2014). [Online]. Available: http://symform.com/
[9] T. Mager, E. Biersack, and P. Michiardi, "A measurement study of the Wuala on-line storage service," in *Proc. 12th IEEE Int. Conf. Peer-to-Peer Comput.*, 2012, pp. 237–248.
[10] A. Montresor and L. Abeni, "Cloudy weather for P2P, with a chance of gossip," in *Proc. 11th IEEE Int. Conf. Peer-to-Peer Comput.*, 2011, pp. 250–259.
[11] R. Sweha, V. Ishakian, and A. Bestavros, "Angels in the cloud: A peer-assisted bulk-synchronous content distribution service," in *Proc. 4th IEEE Int. Conf. Cloud Comput.*, 2011, pp. 97–104.
[12] G. Kreitz and F. Niemela, "Spotify - large scale, low latency, P2P music-on-demand streaming," in *Proc. 10th IEEE Int. Conf. Peer-to-Peer Comput.*, 2010, pp. 1–10.
[13] F. Liu, S. Shen, B. Li, B. C. Li, H. Yin, and S. Li, "Novasky: Cinematic-quality VoD in a P2P storage cloud," in *Proc. 30th IEEE Int. Conf. Comput. Commun.*, 2011, pp. 936–944.
[14] A. H. Payberah, H. Kavalionak, V. Kumaresan, A. Montresor, and S. Haridi, "CLive: Cloud-assisted P2P live streaming," in *Proc. 12th IEEE Int. Conf. Peer-to-Peer Comput.*, 2012, pp. 79–90.
[15] H. Kavalionak, E. Carlini, L. Ricci, A. Montresor, and M. Coppola, "Integrating peer-to-peer and cloud computing for massively multiuser online games," in *Proc. Peer-to-Peer Netw. Appl.*, pp. 1–19, Sep. 2013.
[16] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and Z. Li, "Enabling security in cloud storage SLAs with CloudProof," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2011, pp. 31–31.
[17] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu, "Scalable secure file sharing on untrusted storage," in *Proc. 2nd USENIX Conf. File Storage Technol.*, 2003, pp. 29–42.
[18] S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: management of access control evolution on outsourced data," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 123–134.
[19] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Trans. Inform. Syst. Security*, vol. 9, no. 1, pp. 1–30, Feb. 2006.
[20] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. 28th IEEE Symp. Security Privacy*, 2007, pp. 321–334.
[21] 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. 534–542.
[22] S. Yu, C. Wang, K. Ren, and W. Lou, "Attribute based data sharing with attribute revocation," in *Proc. 5th ACM Symp. Inform., Comput. Commun. Security*, 2010, pp. 261–270.
[23] 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, Nov. 2011.
[24] L. Ibraimi, Q. Tang, P. Hartel, and W. Jonker, "Efficient and provable secure ciphertext-policy attribute-based encryption schemes," in *Proc. 5th Inform. Security Practice Experience*, 2009, pp. 1–12.
[25] L. Cheung and C. Newport, "Provably secure ciphertext policy ABE," in *Proc. 18th ACM Conf. Comput. Commun. Security*, 2007, pp. 456–465.
[26] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Syst.*, vol. 43, no. 2, pp. 618–644, Jul. 2007.
[27] B. Waters, "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization," in *Proc. 14th Int. Conf. Practice Theory Public Key Cryptography Conf. Public Key Cryptography*, 2011, pp. 53–70.
[28] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proc. Adv. Cryptology—EUROCRYPT' 11*, 2011, pp. 568–588.
[29] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Proc. Adv. Cryptology—EUROCRYPT' 98*, 1998, pp. 127–144.

[30] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *Proc. 21st Annu. Int. Cryptology Conf. Adv. Cryptology*, 2001, pp. 213–229.
[31] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[32] J. Bethencourt, A. Sahai, and B. Waters. (2011). Advanced crypto software collection: The cpabe toolkit. [Online]. Available: http://acsc.cs.utexas.edu/cpabe/
[33] B. Lynn. (2011). The Pairing-based cryptography library. [Online]. Available: http://crypto.stanford.edu/pbc/.
[34] F. Zhao, T. Nishide, and K. Sakurai, "Realizing fine-grained and flexible access control to outsourced data with attribute-based cryptosystems," in *Proc. 7th Int. Conf. Inform. Security Practice Experience*, 2011, pp. 83–97.
[35] P. K. Tysowski and M. A. Hasan, "Hybrid attribute- and re-encryption-based key management for secure and scalable mobile applications in clouds," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 172–186, Nov. 2013.

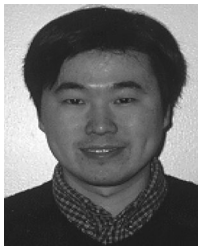


Heng He received the BS degree from School of Computer Science at Hubei University of Technology, Wuhan, China, in 2004, and the MS degree from School of Computer Science and Technology at Huazhong University of Science and Technology, Wuhan, China, in 2007. He is currently working toward the PhD degree in the School of Computer Science and Technology at Huazhong University of Science and Technology, and a Lecturer in the School of Computer Science and Technology at Wuhan University of Science and Technology. His research interests include cloud computing, peer-to-peer computing, network coding and network security.

Ruixuan Li received the BS, MS and PhD degrees in computer science from Huazhong University of Science and Technology, Wuhan, China in 1997, 2000 and 2004, respectively. He is currently a Professor in the School of Computer Science and Technology at Huazhong University of Science and Technology. He was a Visiting Researcher in Department of Electrical and Computer Engineering at University of Toronto, Toronto, ON, Canada, from 2009 to 2010. His research interests include cloud computing, distributed data management, and distributed system security. He is a member of IEEE and ACM.



Xinhua Dong received the MS degree in computer science from Huazhong University of Science and Technology, Wuhan, in 2008. He is currently working toward the PhD degree in the School of Computer Science and Technology at Huazhong University of Science and Technology, Wuhan, China. His research interests include information retrieval, cloud security and big data management.



Zhao Zhang received the BS and MS degrees in computer science from Huazhong University of Science of Technology, Wuhan, China, in 1991 and 1994, respectively, and the PhD degree in computer science from the College of William and Mary, Williamsburg, VA, USA, in 2002. He is currently an Associate Professor of computer engineering at Iowa State University, Ames, IA. His research interests include computer architecture, parallel and distributed systems and architectural support for system security. He is a member of the IEEE and a senior member of the ACM.