A Dynamic Secure Group Sharing Framework in Public Cloud Computing

Kaiping Xue, *Member, IEEE,* and Peilin Hong, *Member, IEEE*

Abstract—With the popularity of group data sharing in public cloud computing, the privacy and security of group sharing data have become two major issues. The cloud provider cannot be treated as a trusted third party because of its semi-trust nature, and thus the traditional security models cannot be straightforwardly generalized into cloud based group sharing frameworks. In this paper, we propose a novel secure group sharing framework for public cloud, which can effectively take advantage of the Cloud Servers' help but have no sensitive data being exposed to attackers and the cloud provider. The framework combines proxy signature, enhanced *TGDH* and proxy re-encryption together into a protocol. By applying the proxy signature technique, the group leader can effectively grant the privilege of group management to one or more chosen group members. The enhanced *TGDH* scheme enables the group to negotiate and update the group key pairs with the help of Cloud Servers, which does not require all of the group members been online all the time. By adopting proxy re-encryption, most computationally intensive operations can be delegated to Cloud Servers without disclosing any private information. Extensive security and performance analysis shows that our proposed scheme is highly efficient and satisfies the security requirements for public cloud based secure group sharing.

Index Terms—secure group sharing, forward secrecy, backward secrecy, public cloud computing, group key agreement

1 Introduction

The demand of outsourcing data has greatly increased in the last decade. To satisfy the need for data storage and high performance computation, many cloud computing service providers have appeared, such as Amazon Simple Storage Service (Amazon S3), Google App Engine, Microsoft Azure, Dropbox and so on. There are two obvious advantages to store data in Cloud Servers: 1) The data owners save themselves out from the trouble of buying extra storage servers and hiring server management engineers; 2) It is easier for the data owner to share their data with intended recipients when the data is stored in the cloud.

Despite of the above advantages of cloud storage, there still remain various challenging obstacles, among which, the privacy and security of users' data have become two major issues. Traditionally, the data owner stores his/her data in the trusted servers, which are generally controlled by a fully trusted administrator. However, the cloud is usually maintained and managed by a semi-trusted third party (Cloud provider). As a result, traditional security storage technologies cannot be directly applied in the cloud storage scenario. While it is desirable for the data owner to share his/her private data with intended recipients, it presents an even more challenging problem since we have to make sure that except the intended recipients, nobody, including the cloud providers, can obtain any useful information from the encrypted data.

The conventional approach to address the above mentioned problem is to use cryptographic encryption mechanisms, and store the encrypted data in the cloud. Authorized users can download the encrypted files and decrypt them with the given keys. But in this scenario, how to distribute and update session keys is one of the most important but hard problems. *Digital Envelope*[1] is used to address this task in [2], [3]: the data is encrypted with a randomly chosen session key by using symmetric encryption, and then the session key is encrypted with the public key of the specific user by using public-key encryption. For example, we assume that the user A wants to securely send a file F to the user B. First, The user A chooses a random session key K, and uses a symmetric encryption algorithm (such as *DES* and *AES*) to encrypt the file *FILE*: $\{FILE\}_K$. Then user A uses an asymmetric encryption algorithm (such as *RSA*) to encrypt the session key K: $E_{PuK_B}(K)$ (PuK_B is B's public key). Here, $E_{PuK_B}(K)$ is named as a digital envelope, which can be transmitted in the open environment, and be decrypted only by the user B. However, in normal ways, if a file is shared to N specific authorized users, N digital envelopes are required to be generated. Therefore, the computing and communication overhead of generating digital envelopes is $O(N)$ for one file. Meanwhile, the computational complexity and communication overhead of session key updating are both $O(N)$. Moreover, we assume that one session key is required for each one sharing file. If the total number of shared files is M for N specific recipients, the overall overhead of digital envelope generation for all shared files is as large as $O(MN)$.

There have been several other works[4], [5], [6], [7] on

• *K. Xue and P. Hong are with the Department of Electrical Engineering and Computer Science, University of Science and Technology of China, Hefei, China, 230027.*
E-mail: kpxue@ustc.edu.cn, plhong@ustc.edu.cn

the privacy preserving data sharing issue in cloud based on various cryptographic tools, such as attribute based encryption (*ABE*)[8], proxy re-encryption[9] , etc. Among these existing schemes, in [4], Yu et al. have provided a fine-grained and scalable solution. The efficiency of Yu et al.'s scheme[4] relies on that there is high attribute variability between different files and high attribute variability between different users. The efficiency of the schemes in [5] and [6] depends on the assumption that Cloud Servers must be absolutely trusted. Otherwise, Cloud Servers can launch the collusion attack with some curious leaving group members. [7] has tried to realize an *ABE* and proxy re-encryption based data sharing scheme in mobile devices, which also has the problem mentioned in [5], [6].

With the development of cloud services and social networks, a group can be easily organized between some people over Internet due to the same interests, so that group applications with the aid of Cloud Servers become possible and attract more and more attentions[10], [11], [12]. Some Internet companies, such as Facebook, Dropbox, and Tencent, have provided their own group applications. Taking Fig. 1 for example, the normal group application scenario in cloud can be described as follows: The group leader opens up a sharing area in the cloud to form a group application. Then, he/she grants the group members the right to implement data management. All the data in this group are available to all the group members, while they remain private towards the outsiders of the group including the cloud provider. The group leader can authorize some specific group members to help with the management of the group, and this privilege can also be revoked by the group leader. When a member leaves the group, he/she will lose the ability to download and read the shared data again.

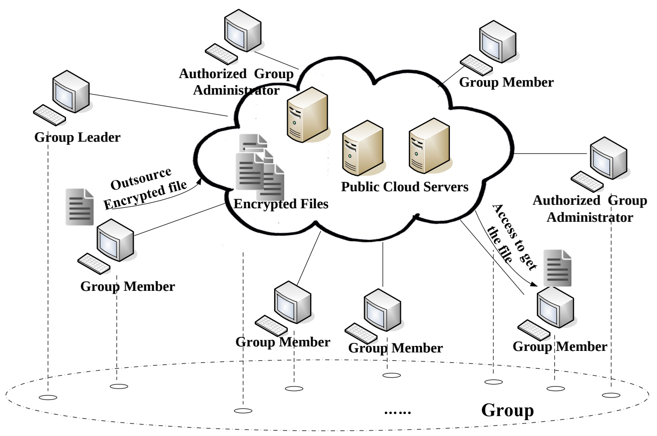


Fig. 1. An example of cloud based group sharing scenario

Our framework in this paper aims to reduce the overhead for the involved parties, while alleviating the trustiness dependence of the semi-trusted cloud provider. Additionally, there is another crucial design request in this scenario: any group member, including the group leader, can be temporary offline and become online again at any time.

Main contribution of this paper can be summarized as follows: 1) The proposed scheme supports the updating of the group key pair whenever group members' joining or leaving happens, which transfers most of the computational complexity and communication overhead to Cloud Servers without leaking the privacy. 2) Privilege of group management can be granted to any specific group member, which can be revoked at any time. 3) Enhanced on the original *TGDH*, with the help of Cloud Servers, the proposed scheme enables the group to negotiate and update the group key pairs even though not all of the group members are online together. Any offline group member can launch group key synchronization when he/she becomes online again in the next time.

The rest of this paper is organized as follows. In Section 2, we discuss the system models of our proposed scheme: network model and security model. Section 3 reviews some related technique preliminaries. Section 4 presents our dynamic secure group sharing framework in public clouds. In Section 5, we give security and performance analysis. Finally, we briefly discuss related work in Section 6, and conclude this paper in Section 7.

2 System Models

2.1 Network Model

Network model in this paper is shown in Fig. 1, where the group membership can change over time: each group member except the group leader can leave or apply to join the group at his/her will. Moreover, each group member in the group can be temporary offline and become online again at any time. Regardless of whether everyone is online or offline, the group can negotiate a group key pair (the group public key and the group private key) with the help of Cloud Servers. This group key pair is used to protect the data shared in the group. Group members' leaving and joining can launch key updating process. Temporary offline group members should be also considered in protocol design. When these group members become online again, they should implement key synchronizing to compute to get the current key pair.

Meanwhile, Cloud Servers have powerful computing and storing capability to help with the group key maintaining process, but cannot leak private information of the group, including data, group members' security parameter information and so on. When a member leaves the group, he/she will lose the ability to download and read the shared data ever again, which is called *backward secrecy* in cloud based group sharing. To the data shared before the revoked member's leaving, Cloud Servers can copy them, or the leaving member can download the encrypted data and related digital envelopes before his/her leaving. So wholly preventing leaving member from download and read the data shared before his/her leaving cannot be achieved. Actually, we introduce challenge-response based access policy, which can effectively prevent the leaving member from download the shared data, if Cloud Servers are semi-trusted. Different from group communication in traditional ways[13], [14], [15], [16], [17], [18], [19], [20], [21], *forward secrecy* is redefined in secure cloud sharing, which means that newly joining group members can decrypt and read all the shared files now and before.

There are three kinds of users in cloud based group sharing applications:

1) ***Group Leader*** (*GL* in short): There is only one group Leader for a group, who is the group creator and the top level group administrator. He/she buys or obtains storage and computing resource from the cloud provider. *GL* can authorize specific group members to manage the group, and this privilege of management can also be revoked by *GL*. *GL* provides initial group security parameters for all group members in the group.
2) ***Group Administrator*** (*GA* in short, described as GA_i, $i = 1, 2, \ldots, m$): There are 0, 1, or more authorized group administrators in a group. They can maintain group membership, and acts as sponsors to implement group key updating. Their privilege of management can be revoked by the group leader at any time. They also have all the functions of basic group members, such as uploading and downloading.
3) ***Group Member*** (*GM* in short, Described as M_j, $j = 1, 2, \ldots, n$): Each group member can implement file download and upload operations in the authenticated group. Each *GM* can get some related public information from Cloud Servers and compute the specific set of security parameters, such as group key pair.

Here, $GL \in \{GA\} \subseteq \{GM\}$, $m < n$.

2.2 Security Model

In this work, we just consider the cloud provider is ***semi-trusted***: honest but curious, which means that Cloud Servers would follow our proposed protocol in general, but would try to find out as much secret information as possible based on each group member's inputs. In general, we assume Cloud Servers are interested in data contents and group member's security information rather than other secret information. Cloud Servers might collude with some malicious members for the purpose of getting data contents and group members' private information.

Our scheme should satisfy the security requirements of *backward secrecy* and *forward secrecy*. The former one ensures that the revoked user cannot decrypted new ciphertexts. The later one ensures that the newly joined user can also access and decrypt the previously published data. This two security requirements are usually used in some cloud based data sharing scenarios, such as [4], [5], [7], [22], [23], [24], [25].

A potential adversary may be a former group member or any one out of the group. We assume that an adversary can be a passive attacker who could be a man-in-the-middle to monitor the communications among the group members and Cloud Servers. A former group member can collude with Cloud Servers and try to access data contents shared in his/her former group. An active adversary is able to impersonate an legitimate group member to gain some right.

In general, we say that our scheme is secure if no adversary can succeed with any possible attacks mentioned above.

3 Technique Preliminaries

3.1 Proxy Signature

Proxy signature[26], [27] is a signature scheme, in which an original signer can delegate his/her signing capability to a proxy signer, and then the proxy signer generates a signature on behalf of the original signer. From a proxy signature, a verifier can be convinced of the original signer's agreement on the signed message. Researchers have proposed 3 kinds of proxy signature algorithms: *full delegation*, *partial delegation* and *partial delegation by warrant*. The former two are eliminated by *partial delegation with warrant*[28], which is proved to be more secure and practical, so we also use partial delegation with warrant in our protocol design.

Let A be an original signer who has an authentic key pair(PrK_A and PuK_A), and B be a proxy signer who has an authentic key pair(PrK_B and PuK_B). Let m_w be A's warrant information for the delegation, which has semantic means including the original signer's identity, some information about the proxy signer(for example the identity), period of delegation validity, the qualification of messages on which the proxy signer can sign, etc. Let $\delta_A = Sign(PrK_A, m_w)$ be A's signature on the warrant m_w using his/her private key PrK_A. A transmits δ_A to the proxy signer B. Then partial delegation with warrant based proxy signature scheme is described as follows:

- (***Proxy signature key generation***) $\mathcal{PKG}$ is a proxy signature key generating algorithm that takes original signer's signature δ_A and proxy signer's private key PrK_B as inputs, and outputs a proxy signature key pair($PPrK_B$, $PPuK_B$). It is executed by the proxy signer:

$$(PPrK_B, PPuK_B) \leftarrow \mathcal{PKG}(\delta_A, PrK_B) \tag{1}$$

- (***Proxy signing***) $\mathcal{PS}$ is a proxy signing algorithm that takes proxy signature private key $PPrK_B$ and message m as inputs, and outputs proxy signature δ_P. It is executed by the proxy signer B:

$$\delta_P \leftarrow \mathcal{PS}(m, PPrK_B) \tag{2}$$

- (***Proxy signature verifying***) $\mathcal{PSV}$ is a proxy signature verifying algorithm that takes (δ_P, m, m_w,

PuK_A, PuK_B) as inputs, and outputs either accept or reject. It is executed by any verifier:

$$\mathcal{PSV}(\delta_P, m, m_w, PuK_A, PuK_B) \overset{?}{=} accept\ or\ reject \quad (3)$$

Till now, a lot of partial delegation with warrant based proxy signature schemes are proposed, for example in [29], [30], [31]. In our scheme we use the algorithm examples in [27], [28] to grant the group administration privilege to the specific group members. However, actually we only use the basic concept of proxy signature described here. Other newly proposed reasonable group signature schemes can also be used in our protocol.

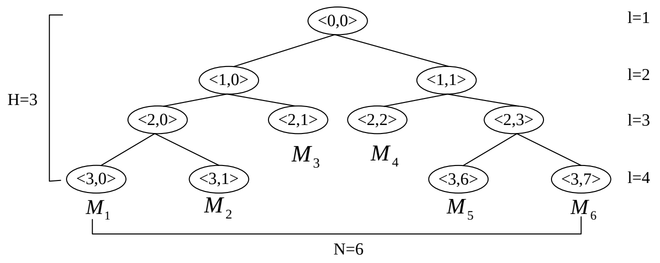


Fig. 2. A *TGDH* key Tree with 6 nodes

3.2 *TGDH* based Group Key Agreement

The *TGDH* protocol in [19] uses an adaptation of binary key trees in the context of fully distributed group key agreement based on Decisional Diffie-Hellman problem[32]. Let p and q be two prime numbers which satisfy the condition $q|p-1$ and the size of p and q are large enough so that solving the discrete logarithm problem in G is infeasible computational, where G is a subgroup with order q of a finite field Z_p^*. Let g be a generator of G. The binary key tree in *TGDH* protocol is organized in the following manner: each node $\langle l, v\rangle$ is associated with a secret key $K_{\langle l,v\rangle}$ and the corresponding blinded key $BK_{\langle l,v\rangle} = g^{K_{\langle l,v\rangle}}\ mod\ p$. Each secret key $K_{\langle l,v\rangle}$ of the internal node $\langle l, v\rangle$ is the Deffie-Hellman exchanged key between its two child nodes and can be computed recursively as follows:

$$\begin{aligned} K_{\langle l,v\rangle} &= BK_{\langle l+1,2v+1\rangle}{}^{K_{\langle l+1,2v\rangle}}\ mod\ p \\ &= BK_{\langle l+1,2v\rangle}{}^{K_{\langle l+1,2v+1\rangle}}\ mod\ p \\ &= g^{K_{\langle l+1,2v\rangle}K_{\langle l+1,2v+1\rangle}}\ mod\ p \end{aligned} \quad (4)$$

The key pair at the root node ($K_{\langle 0,0\rangle}$ and $BK_{\langle 0,0\rangle}$) is the established group key pair (group public key PuK_G and group private key PrK_G) shared by all group members: $PuK_G = K_{\langle 0,0\rangle}$ and $PrK_G = BK_{\langle 0,0\rangle}$. Each group member is associated with a leaf node, whose security key is randomly and securely chosen.

Based on the *TGDH* protocol, Each group member M_i at the leaf node $\langle l, v\rangle$ knows all publicly shared blinded keys of sibling nodes of all nodes in the path from $\langle l, v\rangle$ to $\langle 0, 0\rangle$ and can compute all secret keys of nodes in the path. For example in Fig. 2, M_2 knows his/her secret key $K_{\langle 3,1\rangle}$ and the blinded keys broadcasted by other group members: $BK_{\langle 3,0\rangle}$, $BK_{\langle 2,1\rangle}$, $BK_{\langle 1,1\rangle}$. Therefore, M_2 can compute the key pairs of nodes $\langle 2, 0\rangle$, $\langle 1, 0\rangle$ and $\langle 0, 0\rangle$.

There are five basic operations in *TGDH*: *Join*, *Leave*, *Merge*, *Partition* and *Key-refresh*. From [19] we know that:

1) A joining operation requires two rounds (broadcast) with two messages. The number of modular exponentiations is $O(2h-2)$ and $O(h-1)(h = \lceil log(n)\rceil)$, where $O(2h-2)$ modular exponentiations are needed by the sponsor to compute $h-1$ security keys Ks and blinded keys BKs, and $O(h-1)$ modular exponentiations are needed by each other member to compute related updated key in his/her path from its associated node to the root node.
2) A leaving operation requires one round with one message. The number of modular exponentiation needed are also $O(2h-2)$ and $O(h-1)$.

There have been a lot of work to enhance the robustness of *TGDH*[33], [34], [35], including how to keep the stability when frequently joining and leaving, overhead optimization when more than one group members joining or leaving at the same time, and so on. However, all of these schemes do not consider how to do key negotiation when not all the group members online together at the same time. The assumption that all group members should be online together cannot be guaranteed in the cloud environment, which makes that the traditional *TGDH* is not suitable. This paper will put forward an improved scheme to deal with this problem.

3.3 Proxy Re-encryption

Proxy re-encryption[9], [36], [37] is an cryptographic primitive in which one person (Take the user A for example) allows a semi-trusted proxy to re-encrypt his/her message that will be sent to another designated person (Take the user B for example). A should generate a proxy re-encryption key $rk_{PuK_A \to PuK_B}$ by combining his/her secret key with B's public key. This re-encryption key is used by the proxy as input of the re-encryption function, which is executed to convert a ciphertext encrypted under A's public key (PuK_A) into another ciphertext that can be decrypted by B's private key (PrK_B). Except for converting, the proxy cannot see the underlying data contents.

Proxy re-encryption is extensively used to provide ciphertext updating in cloud environment. By this way, most computational intensive operations of ciphertext updating can be transferred to Cloud Servers, without reveal any content of ciphertext to them. This paper is not the first to introduce proxy re-encryption into cloud based data sharing. A lot of works[4], [6], [7], [38] rely on proxy re-encryption to re-encrypt digital envelopes or shared data in Cloud Servers. Please refer to [9], [36], [37] for more details of proxy re-encryption schemes.

4 Our Proposed Scheme

This section first gives an overview of our proposed scheme, then describes the scheme in detail which mainly consists of five phases: *Group Initialization*, *Group Administration Privilege Management*, *Group Member Leaving and Joining* (including *Group Member Leaving*, *Group Member Joining* and *Group Administrator Leaving*), *Key Synchronizing*, and *Data Sharing Management*.

4.1 Overview

Obtaining storage and computing resource from the cloud provider, the group leader *GL* implements the phase of *Group Initialization* to initialize a binary tree and some related security information of the group. Then *GL* can unicast the private key of each leaf node to the associated group member under the protection of encryption and signature. With the help of Cloud Servers' storage, each member can compute the group private key PrK_G.

Relying on the proxy signature, the phase of *Group Administration Privilege Management* can help *GL* grant the group administration privilege to some specific group members.

Furthermore, we divide the phase of *Group Member Leaving and Joining* into three possible sub-phases: *Group Member Joining*, *Group Member Leaving* and *Group Administrator Leaving*. Through the sub-phase of *Group Member Joining*, a group administrator and the new joining group member interact with each other to update security information of the group, including the group key pair PrK_G and PuK_G. *Forward Secrecy* should be guaranteed when a group member joins, which ensures that the newly joined user can also access and decrypt the previously published data. Therefore, all the old digital envelopes used to protect session keys, which are generated to encrypted previously published data don't need to be updated. When a group member leaves, his/her associated node is mandated by a group administrator. In the sub-phase of *Group Member Leaving*, the group administrator *GA* launches enhanced *TGDH* based group key updating and then generates a proxy re-encryption key from the version of group public key used in the existing digital envelopes to the new updated version. Different from a general group member, a group administrator usually mandates more than one leaf node, and he/she knows all the secret keys of these leaf nodes. Therefore, when a group administrator leaves, another *GA* or *GL* should mandate all these leaf nodes, change the security keys, and update security information of the group including the group private key. The proxy re-encryption implementation is like that used in the sub-phase of *Group Member Leaving*. With the algorithm of proxy re-encryption, Cloud Servers can update all existing digital envelopes to be encrypted under the new updated group public key.

Key Synchronizing is a key part of enhanced *TGDH* in our scheme. With the help of Cloud Servers, it makes temporarily offline group members can compute the current agreed group private key and other security information which needs to be synchronized.

The phase of *Data Sharing Management* describes the method how to securely upload and download file in the group. Furthermore, we detailed describe all these phases in the following subsections.

4.2 Group Initialization

After obtaining storage and computing resource from the cloud provider, *GL* generates a shortest binary tree with n leaf nodes, where n is the number of group members (including *GL* itself) in the initial group. Each node of this binary tree can only have either zero (as a leaf node) or two child nodes, which means a node with one child is not allowed. Each one of these n leaf node is associated with one different group member.

GL chooses a random number for each leaf node, and uses it to generate a secret key for the associated group member. Then, *GL* follows Eq. 4 to compute the secret keys and the blinded keys for each node in the binary tree. *GL* initializes the version of each node to "0". The version is designed to determine whether key synchronizing is needed, which we will discuss in Section 4.7. Fig. 3 gives the pseudo-code about the group initialization. For each group member M_i, *GL* encrypts $Index_{M_i}$, K_{M_i} and a timestamp value T with PuK_{M_i}: $m_{w\to M_i} \leftarrow E_{PuK_{M_i}}(Index_{M_i}||K_{M_i}||T)$, where $Index_{M_i}$ is the index which represents group member's associated node position in the binary tree), K_{M_i} is the secret key of the leaf node associating with M_i, PuK_{M_i} is M_i's public key, and T is a timestamp value represent the current time. Then *GL* signs $m_{w\to M_i}$ with its private key PrK_{GL}: $\delta_{P_{GL}} \leftarrow Sign(m_{w\to M_i},\ PrK_{GL}, T)$. Finally, for each group member M_i, *GL* unicasts $\{m_{w\to M_i},\ \delta_{P_{GL}}\}$ to M_i.

After receiving the message from *GL*, each group member M_i can verify timeliness of the received message and signature validation, and then get his/her security key K_{M_i} and the index of its associated node $Index_{M_i}$. We assume that there is an authenticated channel between *GL* and Cloud Servers. After the above operation, *GL* uploads the binary tree structure and related information into the cloud. For each node, its blinded key and node version are included. Moreover, for each leaf node, The identity and the public key of its associated group member are further included.

After M_i sends a request including the index of its associated node to Cloud Servers, Cloud Servers reply the blinded keys of all sibling nodes of every node in the path from M_i's associated leaf node to the root node (We specially define that the root node's sibling node is itself.). After that, following Eq.4, the group member M_i can compute all secret keys and blinded keys of every node in the path, finally reaching the root node. The secret key of the root node is the group private key PrK_G, and the blinded key of the root node is the group public key PuK_G.

Take M_2 in Fig. 2 for example. After receiving M_2's request which contains the index of M_2's associated node ($\langle 3, 1\rangle$), Cloud Servers find out sibling nodes of all nodes in the path from the node $\langle 3,1\rangle$ to the root node $\langle 0,0\rangle$, and return the blinded keys of them: $BK_{\langle 3,0\rangle}$, $BK_{\langle 2,1\rangle}$, $BK_{\langle 1,1\rangle}$ and $BK_{\langle 0,0\rangle}$. After receiving these blinded keys, M_i can compute $K_{\langle 3,1\rangle}$, $K_{\langle 2,0\rangle}$, $K_{\langle 1,0\rangle}$, $K_{\langle 0,0\rangle}$ and $BK_{\langle 3,1\rangle}$, $BK_{\langle 2,0\rangle}$, $BK_{\langle 1,0\rangle}$, $BK_{\langle 0,0\rangle}$. Here, $K_{\langle 0,0\rangle}$ and $BK_{\langle 0,0\rangle}$ are the current group private key PrK_G and group public key PuK_G.

```
INITIALIZE()
1   Generate a shortest binary tree $T$ with $n$ leaf nodes.
      ▷ each node in $T$ only have 0 or 2 children.
2   $PrK_G$ = INIT_BINARY_T(ROOT)
      ▷ $Root$ is the root node of $T$
3   $PuK_G = g^{PrK_G}$ $mod$ $p$
4   Associate each leaf node to a group member $M_i$,
      $i = 1,\ 2,\ ...,\ n$

INIT_BINARY_T($s$)
 1  if ($s.left$ && $s.right$)
 2        do $ln = s.left$
 3           $rn = s.right$
 4           $ln.BK = g^{\text{INIT\_BINARY\_T}(ln)}$ $mod$ $p$
 5           $K = ln.BK^{\text{INIT\_BINARY\_T}(rn)}$ $mod$ $p$
 6           $ln.version = rn.version = 0$
 7     else
 8        do $K = random()$
 9  $s.K = K$
10  $s.BK = g^{s.K}$ $mod$ $p$
11  return $K$
```

Fig. 3. Pseudo-code of group initialization process

4.3 Group Administration Privilege Management

GAs can help the group leader GL manage the group, including accepting new group member's joining request, assisting group members to join the group and handling members' leaving event. GL can authorize and revoke the administration privilege to/from some specific group members by his/her will. When GL authorizes a group member GM_j to be an GA, GL first sets the combination of some semantic information such as M_j's identity ID_{M_j}, the starting time, period of validity and the qualification of signing, etc. as the warrant information ($m_{w_{M_j}}$) for the delegation to M_j. Following the process described in Section 3.1, M_j obtains a pair of proxy signature keys $PPrK_{M_j}$ and $PPuK_{M_j}$. After that, M_j can be verified by any other person only if he/she knows GL's public key. If a signature signed by M_j over specific data has passed the verification, M_j can be considered as a legitimate proxy signer delegated by GL, and the verified data can be accepted by the verifier.

There are also possibly leaving events for GA. We will describe the group member leaving and the group administrator leaving processes in Section 4.5 and Section 4.6 respectively. From these two sections, we can find that the group administrator leaving process is more complex than the group member leaving process. Based on this reason, selecting group members to become group administrators should have the great probability to be online for a long time.

4.4 Group Member Joining

When a group member joins, he/she sends a joining request to one group administrator(taking GA_j for example). GA_j handles this joining event as a sponsor. After verifying the new joining group member's legitimacy, GA_j processes as follows:

- GA_j tries to find a leaf node which is mandated by one of the group administrators: If so, the found node is set as the associated one of the new joining group member. If not, GA_j finds the leaf node with the smallest depth in the tree structure, and splits this node to a parent node and two children nodes. The left child is for existing group member associated to the found leaf node and the right one is for the new joining group member.

Then, the new joining group member's process is as follows:

- Randomly select a security key.
- Get the blinded keys of all sibling nodes of every node in the path from his/her associated node to the root node from Cloud Servers.
- Compute new security keys and blinded keys of each node in the path from his/her associated node to the root node.
- Set the versions of his/her associated node and its parent node to "0". Add 1 to the version of each of the other internal nodes in this path.
- Send all the blinded keys from his/her associated node to the root node in this path to the GA_j in an authentication tunnel.

After receiving the above message from the new joining group member, GA_j uploads all these blinded keys in the path to Cloud Servers. Cloud Servers update the tree structure, and the blinded keys of every node in the path. Then Cloud Servers set the versions of the new joining group member's associated node and its parent node to "0", and add 1 to the version of each other internal node in this path.

Group sharing applications often should have the property of forward secrecy, which means that new joining members can decrypt and read all the files even though shared before. Therefore, in our scheme, when a group member joins, the binary key tree and group key pair should be updated, but all digital envelopes do not need to be updated. The group administrator or the group leader as a sponsor can securely send the old group private key PrK_G^{DE}(used to decrypt digital envelopes) to the new joining member: $E_{PuK_G^{NEW}}(PrK_G^{DE})$. The sponsor also send $E_{PuK_G^{NEW}}(PrK_G^{DE})$ to Cloud Servers in an authentication tunnel, and then Cloud Servers store it. PrK_G^{DE} and PuK_G^{DE} are used in digital envelopes. PrK_G^{NEW} and PuK_G^{NEW} are the current new agreed key pair. $E_{PuK_G^{NEW}}(PrK_G^{DE})$ makes every group member, who knows PuK_G^{NEW}, be able to still securely get PrK_G^{DE}. Those group members who become offline to be online again can first compute to get PrK_G^{NEW} and then securely get PrK_G^{DE} from $E_{PuK_G^{NEW}}(PrK_G^{DE})$. If a group member needs to upload a file, he/she should still use PuK_G^{DE} to generate the related *DE*.

```
MEM-LEAVING-PROC($leav\_mem\_n$)
1  if $leav\_mem\_n.sibing$ is also mandated by an $GA$
2        do Merge $leav\_mem\_n$, $leav\_mem\_n.sibling$,
             and $leav\_mem\_n.parent$ to one node
3           Set this merged node
             as the current mandated node $cur\_mand\_n$
4     else  Set $leav\_mem\_n$ as $cur\_mand\_n$.
5  NODE_UPDATING_PROCESS($cur\_mand\_n$)

NODE_UPDATING_PROC($n$)
 1  $n.K = random(\ )$
 2  $n.BK = g^{n.K} \ mod \ p$
 3  $n.version = 0$
 4  $r = n.sibling$
 5  $s = n$
 6  for $t$ is each node in the path
    from $n.parent$ to the root node
 7  if ($t \neq root$)
 8        do $t.K = s.K^{r.BK} \ mod \ p$
 9           $t.BK = g^{t.K} \ mod \ p$
10           $t.version++$
11           $s = t$
12           $r = s.sibling$
13     else  Exit;
```

Fig. 4. Pseudo-code of group member leaving process on mandating *GA*'s side

4.5 Group Member Leaving

When a group member leaves, in order to provide backward secrecy, the group key pair should be updated, and all digital envelopes related to the sharing data in this group should be also updated and encrypted by the new group public key. In our scheme, one *GA* should mandate leaving group member's position in the binary tree and act as a sponsor to implement the group member leaving process.

Detailed pseudo-code of the group member leaving process is shown in Fig. 4. Assume GA_i is the current chosen mandator and sponsor for this leaving event. GA_i's implementation process is further described as follows:

- If the sibling of the leaving group member's associated node is also mandated by a *GA*, both these two nodes (the leaving group member's associated node and its sibling) and their parent node should be merged to one leaf node. GA_i mandates this new leaf node. Otherwise, if the sibling node is associated with a group member, *GA* straightly mandates the leaving group member's associated leaf node.
- Randomly choose a new secret key K' for the new mandated node $\langle i, j\rangle$, and then update secret keys and blinded keys in the path from $\langle i, j\rangle$ to the root node $\langle 0, 0\rangle$. The updated root security key and blinded key are the new group private key(PrK'_G) and group public key(PuK'_G).
- Add 1 to the version of each internal node in this path(except $\langle i, j\rangle$) and set the version of $\langle i, j\rangle$ to "0".

Then, GA_i computes the proxy re-encryption key and then uploads the updated information into the cloud as follows:

- GA_i computes the proxy re-encryption key $rk_{PuK_G \to PuK'_G}$.
- Assuming there are l nodes in the path from the mandated node$\langle i, j\rangle$ to the root node $\langle 0, 0\rangle$, the blinded keys can be named as BK_{l-1}, BK_{l-2}, ..., BK_0 (BK_0 is the new group public key). GA_i sends $Updating\_M||\delta_P||PuK_{GA_i}||m_{w_{GA_i}}$ to Cloud Servers. $Updating\_M$ and δ_P can be computed as follows:

$$\begin{aligned} Updateing\_M &= \langle i, j\rangle||l||BK_{l-1}||BK_{l-2}|| \\ &\quad ...||BK_0||rk_{PuK_G \to PuK'_G}||T, \\ \delta_P &= \mathcal{PS}(H(Updating\_M)) \end{aligned} \tag{5}$$

where T is the timestamp, which can be used to address the reply attack. PuK_{GA_i} and $m_{w_{GA_i}}$ can be stored in cloud in advance, and thus only $Updating\_M||\delta_P$ should be sent.

After receiving the message, Cloud Servers first check whether the timestamp T is within some allowed range compared with the current time, check semantically legitimacy of $m_{w_{M_i}}$(for example, whether the current time is still in the validity period of $m_{w_{M_i}}$), and then verify:

$$\begin{aligned} &\mathcal{PSV}(\delta_P, Updateing\_M, m_{w_{M_i}}, PuK_{GL}, PuK_{M_i}) \\ &\quad \overset{?}{=} accept \ or \ reject \end{aligned} \tag{6}$$

If being "accept", Cloud Servers store node mandating information, update all blinded keys from $\langle i, j\rangle$ to $\langle 0, 0\rangle$, and add 1 to the version of each internal node in this path(except $\langle i, j\rangle$). Set the version of $\langle i, j\rangle$ as "0".

When there is a group member leaving from the group, after group key pair (the group public key and the group private key) is updated, an arbitrary group administrator *GA* or the group leader *GL* should act as a sponsor to re-compute a proxy re-encryption key from the version of group public key used in the existing digital envelopes(PuK_G) to the new updated version(PuK'_G):$rk_{Puk_G \to PuK'_G}$. With this proxy re-encryption key, Cloud Servers can update all existing digital envelopes to be encrypted under the new updated group public key PuK'_G. This method can delegate most of the computation intensive operations to Cloud Servers without disclosing the encrypted data contents and keys in all digital envelopes. Meanwhile, the leaving group member loses the privilege of downloading and decrypting group sharing data from Cloud Servers.

```
ADMIN-LEAVING-PROCESS($leav\_admin$)
1  Paths from each of $leav\_admin$'s mandated nodes
      and associated node to the root node
      form a subtree $T'$
2  UPDAT_SUBTREE($Root\ in\ T'$)

UPDAT_SUBTREE($n$)
 1  if $n$ is not a leaf node in $T'$
 2      do $ln = n.left$ in $T$
 3         $rn = n.right$ in $T$
 4         if $ln$ is in $T'$
 5             do if $rn$ is in $T'$
 6                    do $rn.BK = g^{\text{UPDAT\_SUBTREE}(rn)}\ mod\ p$
 7                 else do get $rn.BK$
                           in $T$ from Cloud Servers
 8                 $ln.K =$ UPDAT_SUBTREE($ln$)
 9                 $K = rn.BK^{ln.K}\ mod\ p$
10      else if $ln$ is not in $T'$ and $rn$ in $T'$
11             do $rn.K =$ UPDAT_SUBTREE($rn$) $mod\ p$
12                $rn.BK = g^{rn.K}\ mod\ p$
13                get $ln.BK$ in $T$ from Cloud Servers
14                $K = ln.BK^{rn.K}\ mod\ p$
15      else if $n$ is a leaf node in $T'$
16          do $K = random()$
17  $n.K = K$
18  $n.BK = g^K\ mod\ p$
19  return $K$
```

Fig. 5. Pseudo-code of group administrator leaving process

4.6 Group Administrator Leaving

Usually each GA mandates more than one leaf node, and he/she knows the secret keys of these leaf nodes. When an GA leaves, another GA or GL should mandate these leaf nodes and change the security keys instead of him/her. As in Fig. 5, the new mandating GA or GL chooses a random secret key for each of the leaving GA's mandated leaf nodes, and computes the secret keys and blinded keys of all node in the path from each of these new mandated leaf node to the root node. All the paths from each of these leaf nodes to the root node form a subtree of the binary tree. Detailed pseudo-code of group administrator leaving process is shown in Fig. 5.

The mandating GA or GL first lists all internal nodes in the subtree T' but whose siblings are not in T', and gets the blinded keys of their siblings in T from Cloud Servers. Taking Fig. 2 for example, we assume that $\langle 3,0\rangle$ and $\langle 2,2\rangle$ are all mandated by an GA (GA_i). When GA_i leaves the group, another $GA(GA_j)$ should mandate these two nodes $\langle 3,0\rangle$, $\langle 2,2\rangle$ and M_i's associated node $\langle 3,6\rangle$. All the paths for these 3 nodes to the root node form a subtree T'. To update the key pairs in T', M_j lists all internal nodes in T' but whose siblings are not in T', and gets the blinded keys of their siblings in T from Cloud Servers. M_j chooses new security keys for $\langle 3,0\rangle$, $\langle 2,2\rangle$ and $\langle 3,6\rangle$. Then M_j computes all the security keys and blinded keys of the nodes in the subtree T', and add 1 to the version of each internal node in T'.

The updated security key of the root node $K_{\langle 0,0\rangle}$ is the new group private key PrK_G^{NEW}, and the updated blinded key of the root node $BK_{\langle 0,0\rangle}$ is the new group public key PuK_G^{NEW}. M_j re-computes a proxy re-encryption key for group's public key from its used version in digital envelopes(PuK_G^{DE}) to the updated version(PuK_G^{NEW}):$rk_{PuK_G^{DE}\rightarrow PuK_G^{NEW}}$. At last M_j uploads new mandating information, all blinded keys in T', and the proxy re-encryption key $rk_{PuK_G^{DE}\rightarrow PuK_G^{NEW}}$ with proxy signature. After passing proxy signature verification, Cloud Servers update the new mandating information, the public key of every node in T', and add 1 to the version of each internal node in T'. After the above process, Cloud Servers re-computes all the digital envelopes based on proxy re-encryption using $rk_{PuK_G^{DE}\rightarrow PuK_G^{NEW}}$.

4.7 Key Synchronizing

When an offline member(taking M_i for example) becomes online again, he/she should implement the key synchronizing process to get the current agreed group private key PrK_G and the current group private key used in DEs (PrK_G^{DE}). M_i gives the index of his/her associated leaf node and node version of every internal node in the path to Cloud Servers. Cloud Servers first get the right position according to M_i's given index. Because of the node joining process with the possible node splitting scenario described in Section 4.4, the position with the given index may not be a leaf node. If this scenario happens, Cloud Severs search in the direction of left down along the left subtree, until reach a leaf node. This reached leaf node can be set as M_i's current associated node. Cloud Servers reply the blinded keys of all siblings of every node in the path from M_i's current associated leaf node to the root node. If the position with the given index is a leaf node, for each inherent node of the path from U_i's associated node to the root node, Cloud Servers compare its version with its given versions from M_i, until to a node when inequality circumstance happens(assume the index of this node is $\langle i,j\rangle$). We can see that all the keys of the sub binary tree with root $\langle i,j\rangle$ haven't changed(except $\langle i,j\rangle$). Cloud Servers reply the blinded keys of all sibling nodes of every node in the path from the position with the index $\langle i,j\rangle$ to the root node. From Eq. 4, the group member M_i can compute security keys and blinded keys of every node in the path. The security key of the root node is the group private key PrK_G^{NEW}, and the blinded key of the root node is the group public key PuK_G^{NEW}. M_i can verify whether the computed group public key is equal to the received group public key from Cloud Servers.

Then, M_i decrypts $E_{PuK_G^{NEW}}(PrK_G^{DE})$ to get PrK_G^{DE}. When uploading files, he/she uses PuK_G^{DE} to generate the related *DE*s. Meanwhile, after downloading files, he/she uses PrK_G^{DE} to decrypt the related *DE* to get the encryption keys.

4.8 Data Sharing Management

Before uploading a file to Cloud Servers, the data owner gives the semantic description of the file: *DESCRIPTION*, which is convenient for searching in the group. Then, the data owner symmetrically encrypts the file with a randomly chosen session key *SK*. Together with uploading the encrypted sharing file, the data owner also uploads a digital envelope $E_{PuK_G^{DE}}(SK)$ (asymmetrically encrypt the session key SK with the group public key PuK_G^{DE}), which is currently used in DE generation. Each file is stored in the cloud as the format:

$$ID||DESCRIPTION||\{File\}_{SK}||E_{PuK_G^{DE}}(SK) \tag{7}$$

where *ID* is a unique identifier for the file.

Group members online or ones from offline to online should timely get the updated $E_{PuK_G^{NEW}}(PrK_G^{DE})$ from Cloud Servers to get the group private key used to decrypt *DE*s:PrK_G^{DE}. When a group member M_i requests to download a file, he/she sends a request *REQ* to Cloud Servers. Cloud Servers respond with a random number NUM and a signature:

$$NUM, Sign(H(NUM||REQ), PuK_{CS}) \tag{8}$$

where NUM is a challenge code to M_i, and PuK_{CS} is the public key for Cloud Servers in the system. M_i first verifies the signature. If passed, he/she uses the current agreed group private key PrK_G^{NEW} to sign NUM: $\delta = Sign(NUM||T_{M_i},\ PrK_G^{NEW})$, where T_{M_i} is a timestamp value. M_i sends δ to Cloud Servers. Cloud Servers verify the timeliness of T_{M_i}: whether the timestamp of the message is in a permitted time window. Then, Cloud Servers use the current agreed group public key PuK_G^{NEW} to verify the signature. If passed, they send the encrypted file and the specific digital envelope($\{File\}_{SK}||E_{PuK_G^{DE}}(SK)$) to M_i. M_i first uses PrK_G^{DE} to get SK, and then decrypts $\{File\}_{SK}$ to get the request file.

5 Security and Performance Analysis

5.1 Security Analysis

- ***Certificateless Authentication***. Based on proxy signature, *GL* can grant the privilege of group administration to some group members as *GA*s. As shown in Eq.6, GA_i only needs to provide $m_{w_{GA_i}}$ and PuK_{GA_i}, every one who knows *GL*'s public key can verify whether GA_i has got *GL*'s authorization. Revoking pre-existing administration privilege granting is also simple, which is like CRL(Certificate Revocation List) mechanism used in Traditional PKI[39], [40]. Cloud Servers can maintain a public stored PRL(Privilege Revocation List), which includes $m_{w_{GA_i}}$ and/or PuK_{GA_i}, when GA_i's management privilege is revoked. Before verifying legitimacy based on Eq.6, the verifier can first search this PRL.
- ***Backward Secrecy When a Group Member Leaves.*** This is provably secure based on the hard Decisional Bilinear Diffie-Hellman problem [32]. When a group member leaves, his/her position in the binary tree is mandated by a group administrator or the group leader *GL*. As illustrated in Fig.6, a group member only knowing the security key of one leaf node could compute security keys of every node in the path from the leaf node to the root node. Because of this reason, the security key of the mandated node should be changed after the group member's leaving. Following Eq.4, in our scheme, the security key and blinded key of each node in the path from the mandated node to the root node can be updated, so the group key pair is changed to a new one. Each of the other group members keeps his/her own previous security key. By request to get required changed blinded key from Cloud Servers. Following Eq.4, All these group members still in the group can compute the current new group key pair. Because leaving group member cannot know the new security key of his/her previously associated node and any other leaf nodes' security key, he/she cannot compute the updated group key pair.
- ***Backward Secrecy When a Group Administrator Leaves.*** This is also provably secure based on the hard Decisional Bilinear Diffie-Hellman problem [32]. When a *GA* leaves, as described in Section 4.6, all his/her mandated and associated nodes should be mandated by another *GA*. In order to make the leaving *GA* computing the final group key pair become impossible, all security keys of these nodes should be changed by the new mandator. After the group administrator leaving process, following Eq.4, all other group members still in the group can compute the final updated group key pair by requesting some necessary updated blinded keys from Cloud Servers. However, the leaving *GA* cannot know any leaf node's security keys, so he/she cannot compute the updated group key pair.
- ***Cloud Provider Cannot Compute the Group Private Key***. Although the cloud provider knows all the blinded keys of every node in the binary tree, he/she cannot know any leaf nodes' security keys. Following Eq.4, he/she cannot compute the security keys of any internal nodes and the root node, so he/she cannot get the final group private key.
- ***Data Confidentiality***. Just as in traditional ways, each file shared in the group (*FILE*) is symmetric encrypted with a session key(*KEY*): $\{FILE\}_{KEY}$ and *KEY* is asymmetric encrypted with the receiver's public keyPuK:$E_{PuK}(KEY)$. Assume the symmetric encryption algorithm and asymmetric encryption algorithm are secure, e.g. separately using AES and RSA. In our scheme, we design a group key pair. *KEY* is

encrypted with the group public key PuK_G. Now the security relies on PrK_G's security, so the scheme should guarantee only authenticated group members know the current group private key. From the above analysis, we know that only the group members who know the security keys of any leaf nodes can finally compute the group private key. Therefore, the security of PrK_G can be guaranteed. Also, when there's a group member leaving the group, PrK_G can be timely updated. Meanwhile Cloud Servers use re-encryption to change the digital envelopes from being under previous group public key to be under the new group public key. Only current group members who know the new group private key can decrypt the download file.

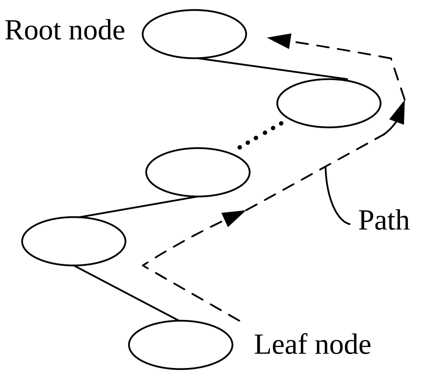


Fig. 6. Path illustration from a leaf node to the root node

5.2 Performance Analysis

We discuss computational complexity and communication overhead in our scheme.

In the process of group initialization, the group leader *GL* should give the definition of some necessary security parameters. *GL* also initializes *TGDH*-based binary tree and securely unicasts separate security keys to every group members, which contains $O(Nlog_2N)$ times exponential modular computation and $O(N)$ times unicast communication. These operations are one-time activities which are only implemented in the initialization stage.

For granting the privilege of group administration to a specific group member, the group leader *GL* needs to create the warrant information m_w and signs it, then securely transmits the signed m_w to the specific group member. These operations contain 2 times asymmetric encryption and 1 time symmetric encryption.

When a group member leaving or joining the group, the group leader *GL* or the specific group administrator who acts as a sponsor chooses a new security key for leaving group member's or new joining group member's associated leaf node, computes other related security keys and blinded keys, and then transmits all the computed blinded keys to Cloud Servers. The transmitted message should contain a proxy signature provide authentication. All these operations contain $O(log_2N)$ exponential modular computation, 1 time proxy signing operation and 1 time communication. When there's a group member leaving the group, digital envelopes need to be updated based on proxy re-encryption by Cloud Servers, which contains $O(L)$ proxy re-encryption computation(L is the number of sharing files in the group). When there's a group member joining the group, there is no need to update digital envelopes, and only a encrypted $E_{PuK_G^{NEW}}(PrK_G^{DE})$ should be computed, which contains only $O(1)$ time encryption operation on the side of the group leader or the group administrator who acts as a sponsor.

When a group administrator (taking GA_i for example) leaves from the group, another *GA* (GA_j) should mandate GA_i's mandated leaf nodes. GA_j chooses new security keys for each of these leaf nodes, and computes security keys and blinded keys of every node in the paths from each of these leaf nodes to the root node. Then, he/she transmits new mandating information, all blinded keys and the proxy re-encryption key to Cloud Servers. The transmitted message should contain a proxy signature to provide authentication. All these operations contain $l \in (O(log_2N), O(Nlog_2N))$ exponential modular computation, 1 time proxy signing operation and 1 time communication. In our scheme, we choose long-time online group members to become *GA* in order to prevent frequently launch *GA* leaving process.

For key synchronization, the group member online or becoming offline to be online again timely publicly gets related blinded keys from Cloud Servers, and then computes security keys and blinded keys of each node in the path from his/her associated node to the root node. All these operations contains $O(log_2N)$ exponential modular computation and 1 time communication. Then the group member decrypts $E_{PuK_G^{NEW}}(PrK_G^{DE})$ to get PrK_G^{DE}, which contains 1 time exponential modular computation.

For uploading a file, the file owner needs to choose a session key, encrypts the file, and generates a digital envelopes. All these operations contain 1 symmetric encryption, 1 time asymmetric encryption, and 1 time communication. The complexity of symmetric encryption and communication is linear with the length of the file.

Before downloading a file, Cloud Servers should first verify whether the group member knows the current group private key to provide authentication. In our scheme we use a Challenge-Response game, containing 2 time communication and 2 times asymmetric encryption on downloading group member's side. After the verification, the downloading group member can get the file and the related digital envelopes from Cloud Servers, then decrypt the digital envelopes to get the session key and decrypt the encrypted file, which contains 1 time asymmetric encryption, 1 time symmetric encryption and 1 time communication to download the file. Here, the complexity of symmetric encryption and communication is linear with the length of the file.

6 Related Work

In Yu et al.'s scheme[4], an encrypted file can be decrypted by a user only if he/she has all of the file's attributes. By using proxy re-encryption, the computing complexity of digital envelope generation for a session key of a sharing file decreases to only $O(1)$ at the data owner's side. For each one-time session key, the data owner needs to compute only one digital envelope by using his/her own public key. Based on the proxy re-encryption algorithm, Cloud Servers can compute digital envelopes for all intended recipient. The efficiency of Yu et al.'s scheme[4] relies on that there is high attribute variability between different files and high attribute variability between different users. But in group applications, different group members usually have same or similar interests, and they usually have attributes in common between them. In the scenario of interest based group sharing, if using Yu et al.'s scheme, the communication and computing overhead of user revocation will be dependent on the size of the group. The efficiency of the schemes in [5] and [6] depends on the assumption that Cloud Servers must be absolutely trusted. Otherwise, Cloud Servers can launch the collusion attack with some curious leaving group members. So, in order to protecting files from the prying eyes of curious Cloud Servers and leaving group members, the data owner needs to re-generate his key pairs and re-generate $N-1$ proxy-re-encryption keys when revoking a group member. This computing overhead is very high for the data owner, especially in the scenario of user joining and leaving frequently in the group.

In traditional studies, the security of group communication applications can be ensured by group key agreement[13], [14], [15], [16], [17], [18], [19], which can provide both *backward secrecy* and *forward secrecy*[20], [21], which are not totally the same as that defined in cloud based group sharing[22], [23], [24], [25]. These schemes can be divided into two categories: centralized [13], [14] and distributed [15], [16], [17], [18], [19], all of which require all group members to be online together during the protocol implementation. Unfortunately, it's difficult to have such "online together" guarantee in group applications in the cloud. How to make sure that such group applications in the cloud are secure and reliable remains a challenging problem. From what we know, only the work in [41], [42] makes a preliminary attempt, which provides a fully distributed *TGDH*(Tree-Based Group Diffie-Hellman)[19] based scheme. Although the scheme only requires asynchronous communication channels, it still requires the group members to participate in the process of protocol implementing and receive some others' sent messages when members' joining and/or leaving. Meanwhile, if a group member acting as a sponsor keeps in storing the private key of the shadow node, when he/she leaves the group, it is hard to keep backward secrecy in this scheme. Our work gives the extension to it to make more operability when any member online or offline at any time. In our scheme, based on Cloud Servers' help, Group members can implement key synchronization when they become online in the next time. We have also discussed the mode of security operations in cloud-based group applications.

7 Conclusion

In this paper we proposed a dynamic secure group sharing framework in public cloud computing environment. In our proposed scheme, the management privilege can be granted to some specific group members based on proxy signature scheme, all the sharing files are secured stored in Cloud Servers and all the session key are protected in the digital envelopes. We use Cloud Servers' aid based enhanced *TGDH* scheme to dynamical updating group key pair when there're group members leaving or joining the group. Even though not all the group members are online together, our scheme can still do well. In order to providing forward secrecy and backward secrecy, digital envelopes should be updated based on proxy re-encryption, which can delegate most of computing overhead to Cloud Servers without disclosing any security information. From the security and performance analysis, the proposed scheme can achieve the design goal, and keep a lower computational complexity and communication overhead in each group members' side.

Acknowledgments

The authors would like to thank Prof. Yuguang Fang, Dr. Hao Yue, Dr. Linke Guo and Dr. Huang Lin for helpful discussions and valuable suggestions during one year visiting in University of Florida. This work is supported by the National Natural Science Foundation of China under Grant No. 61379129 and No. 61170231.

References

[1] RFC2315, "PKCS #7: Cryptographic message syntax(version 1.5)," http://www.ietf.org/rfc/rfc2315.txt, Mar 1998.

[2] Y. Tang, P. Lee, J. Lui, and R. Perlman, "Secure overlay cloud storage with access control and assured deletion," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 6, pp. 903–916, 2012.

[3] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *Ieee Internet Computing*, vol. 16, no. 1, pp. 69–73, 2012.

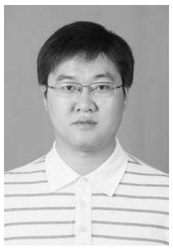
[4] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *IEEE INFOCOM 2010: Proc. The 29th Conference on Computer Communications*. IEEE, 2010.

[5] D. H. Tran, H.-L. Nguyen, W. Zha, and W. K. Ng, "Towards security in sharing data on cloud-based social networks," in *ICICS 2011: Proc. 8th International Conference on Information, Communications and Signal Processing*. IEEE CS, 2011.

[6] W. Jia, H. Zhu, Z. Cao, L. Wei, and X. Lin, "SDSM: A secure data service mechanism in mobile cloud computing," in *WKSHPS 2011: Proc. 2011 IEEE Conference on Computer Communications Workshops*. IEEE CS, 2011, pp. 1060–1065.

[7] P. Tysowski and M. Hasan, "Hybrid attribute-and re-encryption-based key management for secure and scalable mobile applications in clouds," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 172–186, 2013.

[8] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-based encryption for fine-grained access control of encrypted data,” in *CCS 2006: Proc. 13th ACM Conference on Computer and Communications Security*, ser. Proceedings of the ACM Conference on Computer and Communications Security. ACM, 2006, pp. 89–98.
[9] M. Blaze, G. Bleumer, and M. Strauss, “Divertible protocols and atomic proxy cryptography,” in *Advances in Cryptology - EUROCRYPT ’98 International Conference on the Theory and Applications of Cryptographic Techniques Proceedings,*, ser. Advances in Cryptology - EUROCRYPT ’98. Proceedings of International Conference on the Theory and Application of Cryptographic Techniques. Springer-Verlag, 1998, pp. 127–44.
[10] L. Garton, C. Haythornthwaite, and B. Wellman, “Studying online social networks,” *Journal of Computer-Mediated Communication*, vol. 3, no. 1, pp. 0–0, 2006.
[11] L. S. Lai and E. Turban, “Groups formation and operations in the web 2.0 environment and social networks,” *Group Decision and Negotiation*, vol. 17, no. 5, pp. 387–402, 2008.
[12] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, “Group formation in large social networks: membership, growth, and evolution,” in *ACM SIGKDD2006: Proc. 12th international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 44–54.
[13] A. T. Sherman and D. A. McGrew, “Key establishment in large dynamic groups using one-way function trees,” *IEEE Transactions on Software Engineering*, vol. 29, no. 5, pp. 444–458, 2003.
[14] C. K. Wong, M. Gouda, and S. S. Lam, “Secure group communications using key graphs,” *IEEE-ACM Transactions on Networking*, vol. 8, no. 1, pp. 16–30, 2000.
[15] M. Steiner, G. Tsudik, and M. Waidner, “Key agreement in dynamic peer groups,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 8, pp. 769–780, 2000.
[16] Y. Kim, A. Perrig, and G. Tsudik, “Group key agreement efficient in communication,” *IEEE Transactions on Computers*, vol. 53, no. 7, pp. 905–921, 2004.
[17] W. Yu, Y. Sun, and K. R. Liu, “Optimizing the rekeying cost for contributory group key agreement schemes,” *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 3, pp. 228–242, 2007.
[18] W. Trappe, Y. Wang, and K. R. Liu, “Resource-aware conference key establishment for heterogeneous networks,” *IEEE-ACM Transactions on Networking*, vol. 13, no. 1, pp. 134–146, 2005.
[19] Y. Kim, A. Perrig, and G. Tsudik, “Tree-based group key agreement,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 1, pp. 60–96, 2004.
[20] A. Perrig, “Simple and fault-tolerant key agreement for dynamic collaborative groups,” in *CCS’00: Proc. 7th ACM conference on Computer and communications security*. ACM, 2000, pp. 235–244.
[21] Y. Kim, A. Perrig, and G. Tsudik, “Efficient collaborative key management protocols for secure autonomous group communication,” in *CrypTEC’99: Proc. of International Workshop on Cryptographic Techniques and E-Commerce*, 1999, pp. 192–202.
[22] K. Yang, X. Jia, K. Ren, and B. Zhang, “DAC-MACS: Effective data access control for multi-authority cloud storage systems,” in *Proceedings of IEEE INFOCOM 2013*. IEEE, 2013, pp. 2895–2903.
[23] Z. Wan, J. Liu, and R. Deng, “HASBE: A hierarchical attribute-based solution for flexible and scalable access control in cloud computing,” *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 2, pp. 743–754, 2012.
[24] T. Jung, X. Li, Z. Wan, and M. Wan, “Privacy preserving cloud data access with multi-authorities,” in *Proceedings of IEEE INFOCOM 2013*. IEEE, 2013, pp. 2625–2633.
[25] S. Ruj, A. Nayak, and I. Stojmenovic, “DACC: Distributed access control in clouds,” in *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2011, pp. 91–98.
[26] S. Kim, S. Park, and D. Won, “Proxy signatures, revisited,” *Information and Communications Security*, pp. 223–232, 1997.
[27] B. Lee, H. Kim, and K. Kim, “Secure mobile agent using strong non-designated proxy signature,” in *ACISP2001: Proc. 6th Australasian Conference on Information Security and Privacy*, vol. 2119, 2001, pp. 474–486.
[28] M. L. Das, A. Saxena, and D. B. Phatak, “Algorithms and approaches of proxy signature: A survey,” *International Journal of Network Security*, vol. 9, no. 3, pp. 264–284, 2009.
[29] A. Boldyreva, A. Palacio, and B. Warinschi, “Secure proxy signature schemes for delegation of signing rights,” *Journal of Cryptology*, vol. 25, no. 1, pp. 57–115, 2012.
[30] S. Seo, K. Choi, J. Hwang, and S. Kim, “Efficient certificateless proxy signature scheme with provable security,” *Information Sciences*, vol. 188, pp. 322–337, 2012.
[31] T. Malkin, S. Obana, Satoshi, and M. Yung, “The hierarchy of key evolving signatures and a characterization of proxy signatures,” in *Advances in Cryptology-EUROCRYPT 2004*. Springer, 2004, pp. 306–322.
[32] D. Boneh, *The decision diffie-hellman problem*. Springer, 1998, pp. 48–63.
[33] P. Lee, J. Lui, and D. Yau, “Distributed collaborative key agreement and authentication protocols for dynamic peer groups,” *IEEE/ACM Transactions on Networking*, vol. 14, no. 2, pp. 263–276, 2006.
[34] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik, “On the performance of group key agreement protocols,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 3, pp. 457–488, 2004.
[35] W. Yu, Y. Sun, and K. Liu, “Optimizing the rekeying cost for contributory group key agreement schemes,” *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 3, p. 228, 2007.
[36] M. G. G. Ateniese, K. Fu and S. Hohenberger, “Improved proxy re-encryption schemes with applications to secure distributed storage,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 1, pp. 1–30, 2006.
[37] Z. C. J. Shao, P. Liu and G. Wei, “Multi-use unidirectional proxy re-encryption,” in *ICC2011: Proc. 2011 IEEE International Conference on Communications*. IEEE, 2011.
[38] H. Xiong, X. Zhang, D. Yao, X. Wu, and Y. Wen, “Towards end-to-end secure content storage and delivery with public cloud,” in *Proceedings of the second ACM conference on Data and Application Security and Privacy*. ACM, 2012, pp. 257–266.
[39] R. Housley, W. Polk, W. Ford, and D. Solo, “IETF RFC3280, internet x.509 public key infrastructure certificate and certificate revocation list (CRL) profile,” April 2002.
[40] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, “IETF RFC5280, internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile,” May 2008.
[41] I. L’m, S. Szebeni, and L. Butty’n, “Tresorium: cryptographic file system for dynamic groups over untrusted cloud storage,” in *ICPPW2012: Proc. 41st International Conference on Parallel Processing Workshops*. IEEE, 2012, pp. 296–303.
[42] I. L’m, S. Szebeni, and L. Butty’n, “Invitation-oriented TGDH: Key management for dynamic groups in an asynchronous communication model,” in *ICPPW2012: Proc. 41st International Conference on Parallel Processing Workshops*. IEEE, 2012, pp. 269–276.



Kaiping Xue was born in 1980. He graduated with a B.S. degree from the Department of Information Security, in 2003 and received a Ph.D. 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.

Peilin Hong was born in 1961. She received her B.S. and M.S. degrees from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 1983 and 1986. Currently, she is a Professor in the Department of EEIS, USTC. Her research interests include next-generation Internet, policy control, IP QoS, and information security. She has published 2 books and over 100 academic papers in several journals and conference proceedings.