Secure Sharing of Data for Dynamic Group in Public Cloud

Cungang Yang and Celia Li
(Corresponding author: Cungang Yang)
Department of Electrical and Computer Engineering, Ryerson University
350 Victoria Street, Toronto, Canada
(Email: cungang@gmail.com)
(Received June 2, 2019; Revised and Accepted Dec. 28, 2019; First Online Feb. 26, 2020)

Abstract

Cloud Computing has been envisioned as the next-wave system architectures of IT. Cloud storage service brings benefit of not only low cost and scalability, but also great appropriateness for group sharing, such as e-learning resource storage. In this paper, a scalable and efficient group sharing method for public cloud is proposed. It is based on the key tree approach and the well-known Diffie-Hellman cryptographic protocol. Security and performance analysis shows that our proposed scheme is secure and highly efficient for public cloud based secure storage.

Keywords: Cloud Storage; Group Key Management; Group Sharing; Proxy Re-Encryption

1 Introduction

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 [4, 8, 11, 18].

Cloud storage is an extensive new service in the field of cloud computing, which is a distributed file system that using software integrate various types storage to supply users with data storage and access service. Users can easily share their data and reduce the overhead of building and maintaining their data center.

Despite of the advantages of cloud storage, there are various challenges on the privacy and security of users' data. For example,

1) The cloud is usually maintained and managed by a semi-trusted third party (cloud provider).

2) 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.

In order to ensure that only authorized users can access the shared data, the data are encrypted using a cryptographic key known as the *group key*. Data are encrypted with the group key and stored in the cloud. The group key should only be known by authenticated and authorized members of a group. Authorized users can download the encrypted file and decrypt them with the group key. But how to distribute and update group keys is an important problems. We can use digital envelope to address this problem. A digital envelope is a secure electronic data container that is used to protect a message through encryption and data authentication. A digital envelope allows users to encrypt data with the speed of secret key encryption and the convenience and security of public key encryption. For example, when Bob wants to send a confidential message M to Alice, she can generate a digital envelop for M and send the envelop to Alice. On the sender's side the procedure is as follows:

1) Bob randomly generates a secret key K.

2) Bob encrypts M with K, $E(K, M)$.

3) Bob encrypts K with Alice's public key $E(Pub_A, K)$.

4) Bob concatenates $E(K, M)$ with $E(Pub_A, K)$ and sends the result to Alice as digital envelop.

Upon receipt of $E(K, M)$ and $E(Pub_A, K)$, Alice uses her private key PR_A to decrypt the message. The procedure is as follows:

1) Alice decrypts $E(Pub_A, K), K)$ with PR_A.

2) Alice decrypts $E(K, M)$ with K. The result is M.

Before uploading a file to cloud servers, the data owner symmetrically encrypts the file with a randomly chosen session key. The data owner also uploads a digital envelope. Group members should timely get the updated key from cloud servers to get the group private key. When a

group member requests to download a file, he/she sends a request to cloud servers. Cloud servers respond with a random number. The group member uses the current agreed group private key to sign the number. Cloud servers use the agreed group public key to verify the signature. If passed, they send the encrypted file and the specific digital envelope to the member. Mi open the envelop and get the requested file [15].

The architecture of secure shared cloud storage is shown in Figure 1 [15] where 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. To prevent the cloud provider to access the original date, proxy re-encryption is 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 the cloud provider.

Proxy re-encryption is a technique which allows a semi-trusted server transform the ciphertexts encrypted by one party's public keys into proxy ciphertexts encrypted by another party's public keys [2,3,5,10]. Under this circumstance, the two parties can conduct secure data sharing over the same plaintext without exchanging their private keys or worrying about sensitive data leakage in the semi-trusted server. Proxy re-encryption is an cryptographic primitive in which one person (Take the user A for example) allows a semi-trusted cloud provider to reencrypt 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 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 into another ciphertext that can be decrypted by B's private key. Except for converting, the proxy (cloud provider) cannot see the underlying data contents.

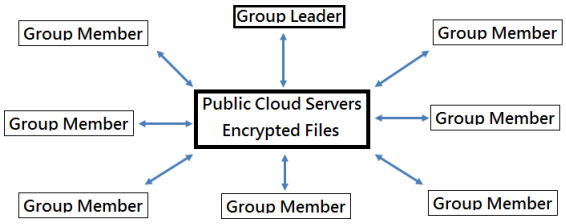

Figure 1: Network model

There are two kinds of users:

1) Group Leader: Only one group leader in a group, who is the group creator. The group leader buys or obtains storage and computing resource from the cloud provider.
2) Group member: Each group member can implement file download and upload operations in the group.

Each group member can get some related public information from cloud servers and compute the group key pair. The group membership can change overtime: each group member except the group leader can leave or apply to join the group at his/her will. All group members 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. Every time a membership change occurs, the group key must be changed to ensure backward and forward secrecy. Backwards secrecy guarantees that a new member joining the group does not have access to any old group keys. This ensures that a member cannot decrypt messages sent before it joins the group. Forward secrecy requires that a member leaving the group does not have access to any future group keys. This ensures that a member cannot decrypt future messages after it leaves the group.

Some researchers have emphasized the integrity and availability of outsourced data. Wang proposed a homomorpic distributed verification protocol using pseudorandom data to ensure cloud storage security [13]. The protocol focuses on the storage correctness as well as verifies misbehaving servers. However, Pseudorandom data does not cover the entire data while identifies the cloud servers, some data corruptions maybe missing such that the protocol do not provide full protection for cloud storage. Kamara proposed a framework of a cryptographic storage service which considers the issue of building a secure cloud storage service on cloud infrastructure where the service provider is not fully trusted by the user [6]. It is made up of three basic components and realizes encryption storage and integrity validation by a group of protocols. However, this method is hard to build since it considers at a high level, needs to modify lots of the source code of cloud storage platform. In addition, users have to query data owner to access the shared data, which will make a communication bottleneck as the number of users increases rapidly. Yeh proposed a secure group communication and data sharing scheme using public key cryptography [17]. However, using such asymmetric cryptography in group needs a PKI infrastructure and the trusted Certificate Authority in the system and each entity need to query the public key of other entity, which will be a overhead as large amount of group members.

Ateniese [1] proposed a proxy re-encryption scheme to manage distributed file systems that attempt to achieve secure data storage in the semi-trusted party. Based on bilinear maps, the scheme offers improved security guarantees. An example of group data sharing in cloud computing was proposed by Liu [9]. In [9], a secure scheme was proposed to support anonymous data sharing in cloud computing. In this paper, we proposed a scheme named CTDH, Our scheme supports the updating of the group

key pair whenever group members joining or leaving. Our approach transfers most of the computational complexity and communication overhead of group key updates to cloud servers without leaking their privacy.

The remainder of this paper is organized as follows. In Section 2, we proposed the group sharing method. The security proof of the scheme is also provided. We analyze the performance of the proposed scheme in Section 3. Section 4 summarizes the paper.

2 Our Proposed Group Sharing Method

The group is composed of the entity with similar interest or common purpose such as society, work-group, or e-learning team. Initially, the group manager as group leader takes charge of key management activities including key generation and key updating. We assume that the group key creation and updates are proceeded in secure channel.

Our scheme uses the key tree structure for scalability and minimal key computation and distribution. It employs the Diffie-Hellman (DH) cryptographic algorithm to avoid the need of setting up a shared key between a member and its group leader. Although each member contributes to the group key, the group leader is responsible for generating and distributing partial keys.

Our scheme is similar to TGDH [7, 12] with respect to the use of the key tree structure and the DH algorithm. They differ in several aspects, as follows:

- In our scheme, the group leader is responsible for computing and distributing the partial keys to the group when a new member joins or an existing member leaves. In TGDH, when a new member joins the group, the current right-most member in the logical key tree is responsible for computing and distributing the partial keys to the other members of the group.
- In TGDH, each member in the group needs to store the entire key tree. In our scheme, only the group leader needs to store the key tree. Each member needs to store only the partial keys of the nodes on the path from itself to the root of the key tree.
- Because the group key updates are now done by the group leader in our scheme instead by the right-most member, each intermediate node in the logical key tree requires two secret keys. These secret keys are changed whenever the partial key associated with the intermediate node needs to be updated upon a join or leave event. The use of these secret keys at intermediate nodes is described in Section 2.1.2.
- It has been shown that the key tree approach gives best performance with 4-ary trees [14, 16]. However, TGDH must work only on binary trees in order to maintain the optimal number of rekeying computations and messages. Our scheme, on the other hand, can support trees of any degree without affecting the complexity of key computation and communication.

In the following sub-sections, we discuss how the group key is computed and updated upon join and leave operations.

2.1 Group Key Generation

2.1.1 Key Tree

The key tree is a logical data structure.

The nodes are numbered as follows. $K_{i,j}$ denotes the content of node $< i, j >$.

Each leaf node $< i, j >$ is associated with a group member and contains the member's secret key (which is generated by the member himself) denoted by $s_{i,j}$.

$$K_{i,j} = s_{i,j}. \tag{1}$$

In the logical key tree displayed by Figure 2, the leaf nodes contain the secret keys $s_{3,0}, ...s_{3,3}, s_{2,2}$ and $s_{2,3}$ of the six group members.

The content of each non-leaf node $< i, j >$ is called a proper key and computed from the contents of its two children $< i+1, 2j >$ and $< i+1, 2j+1 >$ in a recursive manner:

$$K_{i,j} = f(K_{i+1,2j}, K_{i+1,2j+1}). \tag{2}$$

For example, $K_{1,1} = f(s_{2,2}, s_{2,3})$.

By this recursive definition, the proper key $K_{i,j}$ of a non-leaf node $< i, j >$ is made up of the secret keys of the group members in the subtree rooted at node $< i, j >$. For instance, $K_{1,0}$ is the result of $s_{3,0}$, ..., $s_{3,3}$.

The content of the root node $< 0, 0 >$ is the *group key* used by the data source to encode a message and by the group members to decode the message. The group key $K_{0,0}$ is made up of the secret keys of all the group members (i.e., the leaf nodes).

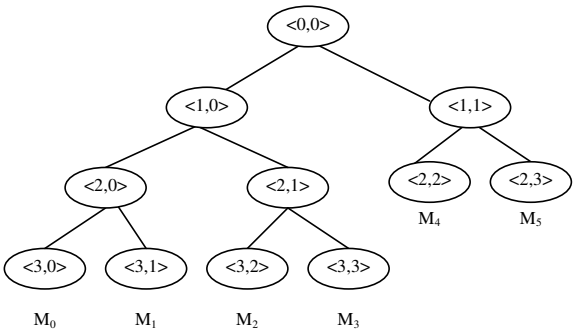

Figure 2: A logical key tree

In the logic key tree, we denote the group key $K(0, 0)$ and $g_K(0, 0)$ as the group private key PrKG and group public key PuKg. The established group key pair is shared by all group members: PrKG = $K_{0,0}$ and PuKG = $g_K(0, 0)$.

2.1.2 Group Key Generation

Each member contributes a share to the group key, but the group leader computed required partial keys and distribute to the whole group. Each member extracts the necessary partial keys from the leader's broadcast message, then combines with his own secret key to compute the group key as follows.

The required variable for each leaf node is following:

- Each member $< i, j >$ generates its own secret key $s_{i,j}$ to contribute towards the group key. It sends $g^{s_{i,j}}$ to the leader only.

Each non-leaf node $< i, j >$ requires the following variables:

- a proper key $K_{i,j}$ computed by the leader. The computation of $K_{i,j}$ is different from that in TGDH though, as will be shown shortly.
- a secret key $S_{i,j}$ generated by the leader for node $< i, j >$. The leader generates a new key $S_{i,j}$ when $K_{i,j}$ needs to be updated (i.e., node $< i, j >$ is on the path from the root to the joining/leaving member in the key tree).
- a hidden key $R_{i,j}$, also generated by the leader for node $< i, j >$. The hidden key $R_{i,j}$ is updated along with the secret key $S_{i,j}$.

Each member $< i, j >$ also has a proper key $K_{i,j}$, where $K_{i,j} = s_{i,j}$, the secret key created by the member himself.

If node $< i, j >$ is a non-leaf node, its proper key $K_{i,j}$ is computed from $S_{i,j}$, $R_{i,j}$ and the two proper keys of the two children of nodes $< i, j >$ as follows:

$$\begin{aligned} K_{i,j} &= (g^{R_{i,j}} \times g^{K_{i+1,2j}})^{S_{i,j}} \times (g^{S_{i,j}})^{K_{i+1,2j+1}} \\ &= (g^{R_{i,j}} \times g^{K_{i+1,2j+1}})^{S_{i,j}} \times (g^{S_{i,j}})^{K_{i+1,2j}}. \end{aligned}$$

The group key $K_{0,0}$ is computed recursively using Equations (1) and (2).

Note that the proper key $K_{i,j}$ of node $< i, j >$ contains the proper keys of its two children $K_{i+1,2j}$ and $K_{i+1,2j+1}$. In addition, it contains the secret key $S_{i,j}$ and hidden key $R_{i,j}$ generated by the leader.

If we would like to use the concept of blinded keys as in TGDH, we can re-write $K_{i,j}$ as follows:

$$\begin{aligned} K_{i,j} &= B_{i+1,2j} \times (g^{S_{i,j}})^{K_{i+1,2j+1}} \\ &= B_{i+1,2j+1} \times (g^{S_{i,j}})^{K_{i+1,2j}}, \end{aligned}$$

where

$$B_{i+1,2j} = (g^{R_{i,j}} \times g^{K_{i+1,2j}})^{S_{i,j}},$$

and

$$B_{i+1,2j+1} = (g^{R_{i,j}} \times g^{K_{i+1,2j+1}})^{S_{i,j}}$$

The secret key $S_{i,j}$ and the hidden key $R_{i,j}$ generated by the leader are used to assure that as long as the leader's contribution is chosen at random, even a coalition of all other parties will not be able to have any means of controlling the final value of the group key. Therefore, the protocol are fairer and more secure in order to prevent some parties having any kind of advantage over the others.

Group key pair will then be calculated as the group public key PrKG = $K_{0,0}$ and the group private key PuKg = $g_K(0, 0)$. The group leader generates a shared secret key K, encrypts shared data with K. The group leader then encrypt K with the group public key PrKG and upload the digital envelop and the encrypted data to the cloud.

2.2 CTDH Implementation

2.2.1 Group Key Refresh/Reinforce

The group key may need to change periodically and may not be related to any change of group membership. The purpose of refreshing the group key periodically is to prevent an adversary from having a sufficient time or resources to break the key. This process is initiated and performed by the group leader which generates a new secret key and a new hidden key, computes and multicasts the blinded value to the whole group. We show the group key refresh/reinforce operation below.

- Randomly choosing a non-leaf node $< i, j >$ and generating a new secret key $S'_{i,j}$ and a new hidden key $R'_{i,j}$ to replace $S_{i,j}$ and $R_{i,j}$.
- Updating the proper keys and partial keys as a result of the above secret key and hidden key changes.
- Broadcasting the updated partial keys and $g^{S'_{i,j}}$ to all members in the group.
- Each member re-computes the group key using the algorithm described in Section 2.1.2.

2.2.2 Join

When a new member joins the group, the member will sends a join request to the group leader. The join request triggers the rekeying procedure. The leader determines the insertion point in the key tree. It then adds a new member node and a new internal node. Once the key tree is updated, the leader generates a secret key $S_{i,j}$ and a hidden key $R_{i,j}$ for the new internal node $< i, j >$. Next, the leader computes all blinded keys and broadcast them with $g^{S_{i,j}}$ ($S_{i,j}$ is the secret key of new internal node $< i, j >$). All members can then compute the new group key.

Group members do not need to maintain a copy of the key tree like in TGDH as they only need to know the blinded keys of its sibling along the path to the root node. Also the future rekeying does not require member having addition information about the key tree. Thus, CTDH only requires member to know the least information to compute the group key.

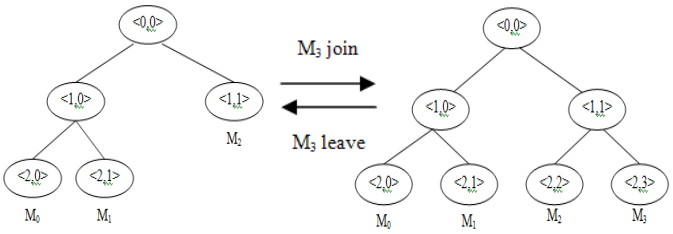

Figure 3: An example of CTDH implementation

Following is an example that illustrates the rekeying operations.

In Figure 3, when member M_3 wishes to join the group by sending its own blinded key to the group leader. The group leader performs the following operations to update the group key:

- Creating two new nodes as children of node <1, 1>.
- Moving M_2 and its proper key from node <1, 1> to node <2, 2>.
- Assigning node <2, 3> to the new member M_3. Member M_3 generates its own secret key $s_{2,3}$ and sends $g^{s_{2,3}}$ to the group leader.
- In general, the proper keys of the nodes on the path from the new member to the root are updated. Since node $<1,1>$ becomes a non-leaf node, the group leader needs to assign it with a secret key $S_{1,1}$ and a hidden key $R_{1,1}$.
- Updating the following blinded keys as a result of the above secret key and hidden key changes and tree restructuring: $B_{2,2}$, $B_{2,3}$ (two newly created partial keys) and $B_{1,1}$. Note that the blinded keys $B_{2,0}$, $B_{2,1}$, $B_{1,0}$ are not changed. The new blinded keys $B_{2,2}$, $B_{2,3}$ and $B_{1,1}$ are computed as follows:

$$\begin{aligned} B_{2,2} &= (g^{R_{1,1}} \times g^{s_{2,2}})^{S_{1,1}} \\ B_{2,3} &= (g^{R_{1,1}} \times g^{s_{2,3}})^{S_{1,1}} \\ B_{1,1} &= (g^{R_{0,0}} \times g^{K_{1,1}})^{S_{0,0}}. \end{aligned}$$

- Broadcasting the updated blinded keys $B_{2,2}$, $B_{2,3}$,$B_{1,0}$, $B_{1,1}$ and $g^{S_{1,1}}$ to all members in the group. Each member re-computes the group key using the algorithm described in Section 2.1.2.

  For example, member M_0 and M_1 can compute the new group key as follows:

$$K_{0,0} = B_{1,1} \times (g^{S_{0,0}})^{K_{1,0}}$$

  Member M_2 and M_3 can compute $K_{1,1}$, $K_{0,0}$ as follows:

$$\begin{aligned} K_{1,1} &= B_{2,3} \times (g^{S_{1,1}})^{s_{2,2}} \\ &= B_{2,2} \times (g^{S_{1,1}})^{s_{2,3}} \\ K_{0,0} &= B_{1,0} \times (g^{S_{0,0}})^{K_{1,1}} \end{aligned}$$

The group key pair shared by all group members will then be updated as the group private key PrKG = $K_{0,0}$ and the group public key PuKg = $g_K(0,0)$.

2.2.3 Leave

When an existing member leaves the group, the group key needs to be changed as well. We start with n member and assume member M_l leaves the group. The leader in this case first update the key tree by deleting the leaf node corresponding to M_l. The former sibling of M_l is promoted to replace M_l's parent node. That is to say that an internal node, the parent of M_l, is also deleted. Once the key tree is updated, the leader generates a new secret key $S_{i,j}$ and a new hidden key $R_{i,j}$ for the new parent node $< i, j >$ of M_l's former sibling. Next, the leader computes all blinded keys and broadcast them with $g_{S_{i,j}}$. This allows all members to compute the new group key.

Looking at the setting that Figure 3 shows, if member M_3 leaves the group, the leader performs the following rekeying procedures.

First, node $< 1, 1 >$, $< 2, 3 >$ are deleted. After updating the key tree, the leader picks a new secret key $S_{0,0}$ and a new hidden key $R_{0,0}$ for node $< 0, 0 >$.

- Removing node $< 2, 3 >$.
- Replace internal node $< 1, 1 >$ with member M_2's node.
- Generate a new secret key $S'_{0,0}$ and a new hidden key $R'_{0,0}$ for node $< 0, 0 >$. As node $< 1, 1 >$ becomes a leaf node, no secret key generated by the leader are needed for this node. Thus, previously generated secret key $R_{<1,1>}$ and $S_{<1,1>}$ by the group leader are discarded.
- Updating the proper keys $K_{<0,0>}$ (the group key) as a result of the above tree restructuring. In general, the proper keys of the nodes on the path from the new member to the root are updated.
- Updating the partial keys $B_{<1,0>}$, $B_{<1,1>}$ as a result of the above secret key changes and tree restructuring.

$$B_{1,0} = (g^{R'_{0,0}} \times g^{K_{1,0}})^{S'_{0,0}}$$

$$B_{1,1} = (g^{R'_{0,0}} \times g^{s_{1,1}})^{S'_{0,0}}$$

- Broadcasting the updated partial keys $B_{1,0}$, $B_{1,1}$, and $g^{S'_{0,0}}$ to all members in the group. Each member re-computes the group key using the algorithm described in Section 2.1.2.

  For example, member M_0 and M_1 can compute the new group key as follows:

$$K_{<0,0>} = B_{<1,1>}(g^{S'_{<0,0>}})^{K_{<1,0>}}$$

  Member M_2 can compute the new group key as follows:

$$K_{<0,0>} = B_{<1,0>}(g^{S'_{<0,0>}})^{s_{<1,1>}}$$

The group key pair shared by all group members will then be updated as the group public key $\text{PuKg} = g_K(0,0)$ and the group private key $\text{PrKG} = K_{0,0}$.

2.2.4 Updates on the Cloud Server

When a group member joins or leaves, all digital envelopes related to the sharing data in this group should be also updated and encrypted by the new group public key. After the group public key and the group private key are updated, the group leader will re-compute a proxy re-encryption key from the version of group public key (PuKG) used in the existing digital envelopes to the new updated version (PuKG1). The leader then uploads the updated information into the cloud. With this proxy re-encryption key, cloud servers can update all existing digital envelopes to be encrypted under the new updated group public key PuKG1. 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.

2.2.5 Upload and Download Data

Before uploading a file to cloud servers, the data owner encrypts the file with a randomly chosen session key K. Together with uploading the encrypted sharing file the data owner also uploads a digital envelope (asymmetrically encrypt the session key K with the group public key PuKG), which is currently used.

When a group member requests to download a file he/she sends a request to cloud servers. Cloud servers send the encrypted file and the specific digital envelope to the group member. The group member decrypts the digital envelop to get key K and then decrypts the request file using key K.

2.3 Security Proof

Assuming the DH algorithm is secure, we show that CTDH is secure using a binary tree as an example. In this proof, all operations are assumed to be performed mod p.

Theorem 1. *Let T be a key tree of height m. l be the level of T. Let $s_0, \ldots, s_n$ be secret keys of the members of the group. The shared group key K derived by any member in the application of CTDH is secure.*

Proof. The proof is based on induction over level l ($1 \leq l \leq m$).

Base Case: $l = 1$. Let member M_0 and M_1 be rooted at node $< m-1, 0 >$ as shown in figure 4. Denote the secret keys of M_0 and M_1 as $s_{m,0} = s_0$ and $s_{m,1} = s_1$ respectively. Let $S_{m-1,0}$ and $R_{m-1,0}$ be the secret key and hidden key of the node $< m-1, 0 >$ generated by the group leader. We show that the shared key K derived by any member in the group is secure.

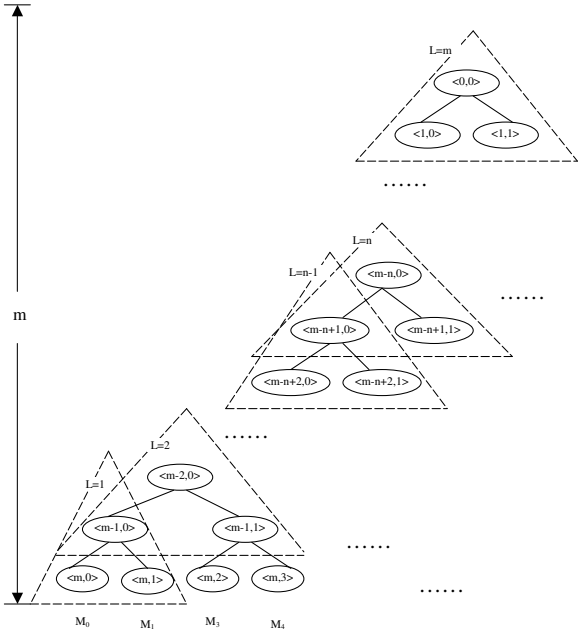

Figure 4: An example of group key calculation

The group leader computes the blinded key for member M_0 and M_1 as follows.

$$B_{m,0} = (g^{R_{m-1,0}} \times g^{s_{m,0}})^{S_{m-1,0}},$$

and

$$B_{m,1} = (g^{R_{m-1,0}} \times g^{s_{m,1}})^{S_{m-1,0}}.$$

After receiving $B_{m,0}$, $B_{m,1}$ and $g^{S_{m-1,0}}$ broadcast by the group leader, member M_0 can derive the shared key as $K = K_{m-1,0} = B_{m,1} \times (g^{S_{m-1,0}})^{s_{m,0}}$.

Since the shared key $K_{m-1,0}$ requires member $M_{m,0}$'s secret key, any passive adversary who gets the broadcast message cannot derive the group key without knowing that secret key.

In case the broadcast message is captured, the adversary tries to use $B_{m,0}$, $B_{m,1}$ and $g^{S_{m-1,0}}$ to derive the shared key $K_{m-1,0}$. Since $(g^{S_{m-1,0}})^{s_{m,0}}$ is the result of the two party Diffie-Hellman key exchange, which is assumed to be indistinguishable in polynomial time from a random number.

Moreover, from the public known information of $B_{m,0}$, $B_{m,1}$ and $g^{S_{m-1,0}}$, the adversary also cannot derived the shared key $K_{m-1,0}$ because the hidden key $R_{m-1,0}$ has never been released to anyone, even the group member. Only the group leader knows what the hidden key it factors into the shared key. Thus, from the broadcast information itself, the adversary cannot derive the shared key. Therefore, the shared key $K_{m-1,0}$ is secure.

Induction hypothesis: The shared key K derived by any member in the application of CTDH, at level $l = n-1$ ($n \geq 2$)is secure.

Induction step: At level $l = n$, $K_{m-n+1,0}$ and $K_{m-n+1,1}$ are the shared values of two subtrees rooted at node $< m-n+1, 0 >$ and $< m-n+1, 1 >$, respectively.

The group leader computes the blinded key for node $< m-n+1, 0 >$ and $< m-n+1, 1 >$ as follows

$$B_{m-n+1,0} = (g^{R_{m-n,0}} \times g^{s_{m-n+1,0}})^{S_{m-n,0}},$$

and

$$B_{m-n+1,1} = (g^{R_{m-n,0}} \times g^{s_{m-n+1,1}})^{S_{m-n,0}}$$

respectively.

After receiving $B_{m-n+1,0}$, $B_{m-n+1,1}$ and $g^{S_{m-n,0}}$ broadcast by the group leader, any member (such as M_0) who belongs to the subtree rooted at node $< m-n+1, 0 >$ can derive the shared key at level n as follows:

$$K_{<m-n,0>} = B_{m-n+1,1} \times (g^{S_{m-n,0}})^{K_{m-n+1,0}}.$$

From the induction hypothesis above, we know that $K_{m-n+1,0}$ is secure and only known by the member who are the children of this subtrees. Thus, the shared group key $K_{<m-n,0>}$ at level n is secure. We can conclude that the shared key of any level l $(1 \leq l \geq m)$ is secure. Therefore, the shared group key $K = K_{0,0}$ of level m derived by any member in the application of CTDH is secure. Furthermore, if extending the inductive method to n-ary key tree, we can apply the same derivation method and show that the derived group key of n-ary key tree is secure as well. □

Theorem 2. *When a new member joins or an exiting member leaves the group, the application of CTDH is still secure. We use Figure 3 as an an example to shown that CTDH provide backward secrecy and forward secrecy.*

First, CTDH provides backward secrecy. Backward secrecy states that a new member who knows the current group key cannot derive any previous group keys. In Figure 3, once a new member M_3 joins the key tree, the group leader for this join event generates a new secret key and a new hidden key and must involve the new member's secret key into the new group key, consequently, previous group key is changed. Therefore, the information learned by the new member with respect to the prior key tree is exactly same as the information of an outsider. Hence, the new member does not gain any advantage compared to a passive adversary.

Second, CTDH provides forward secrecy. Forward secrecy requires that a member who knows a contiguous subset of old group keys cannot discover subsequent group keys once it leaves the group. In Figure 3, once the current member M_3 leaves the group, the group leader refreshes the secret key and hidden key, therefore, all keys known to leaving members will be changed accordingly. Moreover, the new group key will subtract the member M_3's secret. Thus, the information learned by M_3 with respect to the new key tree is exactly same as the information of a passive adversary. This proves that CTDH provides both backward and forward secrecy.

3 Evaluation

In this section, we evaluate our proposed CTDH protocol with Tree-based Group Diffie-Hellman (TGDH) protocol in terms of computation and communication costs.

3.1 Computation Analysis

Table 1 shows computation comparison between TGDH and CTDH when a new member joins the group or an existing member leaves the group. CTDH needs to compute $3\log_2 n + 3$ exponentials while $4\log_2 n$ exponentials are required by TGDH upon joining. Upon leaving, $2\log_2 n + 1$ exponentials are computed using CTDH, where $3\log_2 n - 3$ exponentials are needed by TGDH.

Table 1: Computation comparison

Description		TGDH	CTDH
Join	Group Leader	$2\log_2 n$	$\log_2 n + 3$
	Existing member	at most $\log_2 n - 1$	at most $\log_2 n - 1$
	New member	$\log_2 n + 1$	$\log_2 n + 1$
	Total exponentials	$4\log_2 n$	$3\log_2 n + 3$
Leave	Group Leader	$2\log_2 n - 2$	$\log_2 n + 2$
	Existing member	at most $\log_2 n - 1$	at most $\log_2 n - 1$
	Total exponentials	$3\log_2 n - 3$	$2\log_2 n + 1$

Figure 5 shows that if there are 8 members before a new member joins the group, CTDH has same exponential computation as TGDH. Only if there is less than 8 members in the group before new member joins, TGDH requires 1 or 2 exponential computations less than CTDH. Since these 1 or 2 more exponentials are actually computed by the powerful node in CTDH, they should not cause any big delay comparing with TGDH. If there is more than 8 member in the group before new member joins, CTDH outperforms TGDH.

Figure 6 shows that if there are 16 members before a member leaves the group, CTDH has same exponential computation as TGDH. Only if there are less than 4 members in the group before anyone leaves, TGDH requires 1 or 2 exponential computations less than CTDH. Since group leader is much more powerful than a mobile device in terms of computation, memory, power supply, these 1 or 2 exponentials should not cause any big delay comparing with TGDH. If the gorup has more than 16 members before a member leaves, CTDH outperforms TGDH.

3.2 Communication Analysis

Upon member joins the group, TGDH needs 2 broadcast messages to distribute the partial keys, while CTDH needs 1 unicast and 1 broadcast to update the partial keys. Although CTDH and TGDH require same number

of messages, one unicast has less communication cost than a broadcast when sending same size message.

A more relevant measure for a group key management is the latency that a user experiences from the moment the group change was detected, until the new secure group is established. This time is greater than only analytical cryptographic cost, since it includes network latency. Our simulation results in Figures 7- 8 shows that CTDH provides best communication efficiency. Thus, CTDH perform much better than TGDH in terms of communication delay upon joining or leaving.

4 Conclusions

In this paper, a scalable and efficient group sharing method, CTDH, is proposed for public cloud. CTDH is scalable and efficient thanks to the key tree structure. It uses the contributory approach based on the Diffie-Hellman cryptographic algorithm to avoid the need of setting up pairwise secure channels among members We briefly analyze the security and performance of CTDH. We conclude that CTDH not only provides same level of security as TGDH, but also outperforms TGDH in terms of computation and communication costs. In the future, we would like to present a secure and fault-tolerant key agreement for group data sharing in a cloud storage scheme.

References

[1] G. Ateniese, K. Fu, M. Green and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Transactions on Information and System Security*, vol. 9, no. 1, pp. 1–30, 2006.

[2] M. G. G. Ateniese, K. Fu and S. Hohenberger, "Improved proxy reencryption schemes with applications to secure distributed storage," *ACM Transactions on Information and System Security*, vol. 9, no. 1, pp. 1-30, 2006.

[3] C. I. Fan, "Provably secure timed-release proxy conditional reencryption," *IEEE Systems Journal*, vol. 11, no. 4, 2017.

[4] L. Garton, C. Haythornthwaite and B. Wellman, "Studying online social networks," *Journal of Computer-Mediated Communication*, vol. 3, no. 1, pp. 75-106, 2006.

[5] L. Greenwald, K. Rohloff and D. Stott, "Secure proxy-reencryption-based inter-network key exchange," in *IEEE Military Communications Conference (MILCOM)*, pp. 780-785, 2018.

[6] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *International Conference on Financial Cryptography and Data Security*, pp. 136-149, 2010.

[7] Y. Kim, A. Perrig and G. Tsudik, "Tree-based group key agreement," *ACM Transactions on Information and System Security*, vol. 7, no. 1, pp. 60-96, 2004.

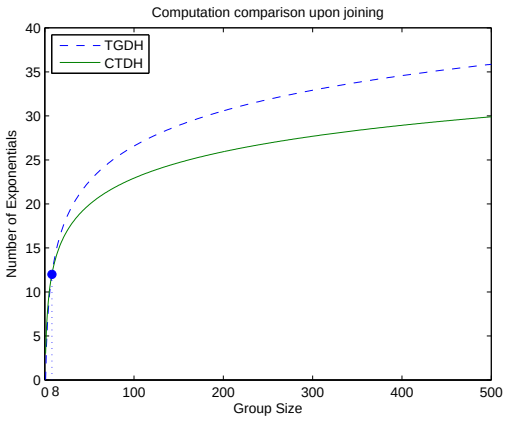

Figure 5: Computation comparison upon joining

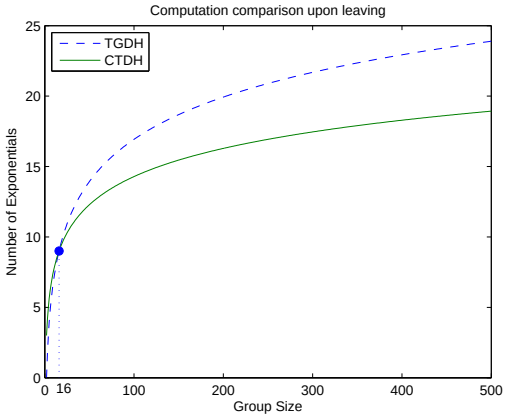

Figure 6: Computation comparison upon leaveing

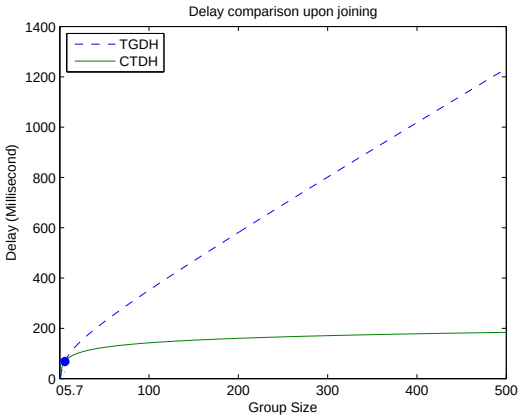

Figure 7: Delay comparison upon member joining

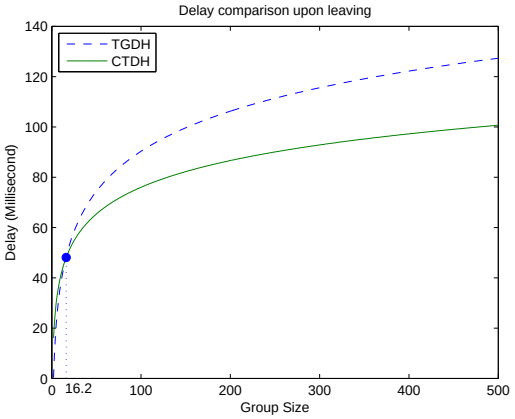

Figure 8: Delay comparison upon member leaving

[8] L. S. Lai and E. Turban, "Groups formation and operations in the web2.0 environment and social networks," *Group Decision Negotiation*, vol. 17, no. 5, pp. 387-402, 2008.

[9] X. Liu, Y. Zhang, B. Wang and J. Yan, "Mona: Secure multi-owner data sharing for dynamic groups in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1182–1191, June 2013.

[10] Z. C. J. Shao, P. Liu and G. Wei, "Multi-use unidirectional proxy reencryption," in *IEEE International Conference on Communications*, pp. 1-5, 2011.

[11] W. Song, "A practical group key management algorithm for cloud data sharing with dynamic group," *China Communications*, vol. 16, no. 6, 2016.

[12] V. R. Thakare and K. J. Singh, "Ternary tree based TGDH protocol for dynamic secure group data sharing in healthcare cloud," in *International Conference on Inventive Computation Technologies (ICICT'16)*, 2016. DOI: 10.1109/INVENTIVE.2016.7823294.

[13] C. Wang, Q. Wang, K. Ren, W. Lou, "Ensuring data storage security in Cloud Computing," in *The 17th International Workshop on Quality of Service*, pp. 1-9, 2009.

[14] C. Wong, M. Gouda and S. Lam, "Secure group communications using key graphs" *IEEE/ACM Transactions Networking*, vol. 8, pp. 16-30, 2000.

[15] K. Xue, P. Hong, "A dynamic secure group sharing framework in public cloud computing," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, 2014

[16] O. Yağan, "Zero-one laws for connectivity in inhomogeneous random key graphs," *IEEE Transactions on Information Theory*, vol. 62, no. 8, 2016.

[17] C. H. Yeh, Y. M. Huang, T. I. Wang and H. H. Chen, "DESCV—A secure wireless communication scheme for vehicle ad hoc networking," *Mobile Networks and Applications*, vol. 14, no. 5, pp. 611-624, 2009.

[18] Y. Zhang, "Authorized identity-based public cloud storage auditing scheme with hierarchical structure for large-scale user groups," *China Communications*, vol. 11, no. 15, 2018.

Biography

Cungang Yang completed his Ph.D degree in computer science in 2003 at University of Regina, Canada. In 2003, he joined the Ryerson University as an assistant professor in the Department of Electrical and Computer Engineering. His research areas include security and privacy, enhanced role-based access control model, information flow control, web security and secure wireless networks.

Celia Li completed her Ph.D degree in electrical engineering and computer science department in 2015 at York University. Her research is focused on security and privacy, role-based access control and wireless mesh network security.