Achieving Dynamic Privileges in Secure Data Sharing on Cloud Storage

Xingwen Zhao, Hui Li

School of Telecommunications Engineering, Xidian University, Xi'an 710071, China

Abstract

With rapid development of cloud computing, more and more enterprises will outsource their sensitive data for sharing in a cloud. Many data sharing and access control schemes have been submitted. However, dynamic privileges among the access groups were not considered. In many circumstances, some users may have higher privileges than others, and they can decrypt more contents than those with low privileges. Moreover, the data owner may want to dynamically control the privileges in data sharing. In this paper, we present an efficient framework for data sharing system to achieve dynamic privileges. Using this framework, any data sharing and access control scheme can be turned into a dynamic privileged scheme, in which the data owner can change the service class of each user dynamically and change the structure of privileges flexibly when it is needed. The proposed framework requires much less storage than previous schemes in handling dynamic privileges among the users.

Keywords: Multi-level security, dynamic privileges, cloud computing, access control.

1. Introduction

Cloud storage is a model of networked online storage, which enables users to remotely store their data in a cloud, and does not require end-user’s knowledge of the physical location and configuration of the system. Migrating data from end-users to the cloud brings great convenience to users, since they can
access data in the cloud anytime and anywhere, using any device, and need not care about the capital investment to deploy the hardware infrastructures. It is especially economical for small enterprises with limited budgets, since they can achieve cost savings and the flexibility to scale investments, by using cloud-based services to manage projects, enterprise-wide contacts and schedules, and so on.

However, there are security and privacy issues for sensitive corporate data, since the cloud service provider (CSP) is operated for making a profit. Thus, an untrustworthy CSP may sell the confidential information about an enterprise to its closest business competitors to earn more money. Therefore, a natural way to keep sensitive data confidential against an untrustful CSP is to store the data only in the encrypted form.

We consider a cloud data sharing system consisting of data owners, data users, cloud servers (maintained by CSP). A data owner stores his sensitive data on cloud servers. Users are issued private keys. To access the remote stored data files shared by the data owner, users need to download the data files from the cloud servers. For simplicity, we assume that the only behavior for users is data file reading, and we can adopt other behaviors in the future. Cloud servers are always online and they are assumed to have abundant storage capacity and computation power. In addition, we also assume that the data owner can store data files besides running his own code on cloud servers to manage his data files. The architecture is shown in Figure 1.

Several tools can be used to provide data sharing and access control on cloud storage, e. g. hierarchical identity-based encryption [1], attribute-based
Some emerging scenarios demand protection for content with different values and for groups with different privileges. As a simple instance shown in Figure 2, an online movie provider allows three classes of members, where VIP members can access all movies, senior members can access high-definition movies and standard-definition movies, and normal members can access only standard-definition movies. It needs 6 sets of ciphertext for encryption schemes without considering different privileges, while it needs only 3 sets of ciphertext for a privileged scheme [4]. For a complex instance (shown in Figure 3), there are many departments in an equipment manufacturer, with each department sharing data among their members. Quality assurance department, project management department and some other members may additionally have access to the data shared in research & development department and manufacturing department. And administrative group may have access to all data. Only ABE schemes among the related works discussed above can be used to fulfil such scenario, by adding proper attributes to users in corresponding groups. However, ABE schemes can achieve only static privileges that are corresponding to the roles, since the privileges cannot be changed once they are determined unless the private keys are renewed. In some cases, it is impossible to renew the private key freely, e.g. the key authority does not support online issuing, or the key authority is not always online. Moreover, online rekeying needs additional mechanisms, such as authentication scheme and secure key updating protocol.

Some scenarios may require dynamic arrangement of privileges. For in-
Figure 3: An complex instance with different privileges.

instance, in a data sharing system with incentives, those who contribute more to the system may have additional access to more valuable data. In such a system, users’ privileges are updated automatically (without updating the private keys) according to their contributions. Another instance is graded digital library (database). Data in the library (database) is graded. People can purchase password of higher grades to obtain more data, or password of lowest grade to access fundamental information. People can change their grades anytime and anywhere by making a phone call or subscribing through a self-service website, while their private keys don’t need to be updated. Therefore, it is highly desirable to design a dynamically privileged data sharing system to enable flexible access control for content with different values.

1.1. Our Contributions.

In this paper, we present an efficient framework for data sharing system to achieve dynamic privileges, and it can be applied to any previous data sharing schemes. We introduce a flexible one-way tree structure for privileges, which enables access groups with higher privileges to derive keys of descending access groups with low privileges. With this feature, previous works will become more practical and require much less storage cost to achieve dynamic privileges among the users.

1.2. Organization.

The remainder of this paper is organized as follows. In Section 2 we describe the related works of tools for data sharing and access control in details. In Section 3 we describe the protocol model and security requirements. In
Section 4, we describe the proposed one-way tree structure for privileges, and then the framework for achieving dynamic privileges in data sharing. Section 6 concludes the paper.

2. Related Works

2.1. Hierarchical identity-based encryption

Identity-based encryption (IBE) system was introduced by Boneh and Franklin [5]. In the system there is one private key generator (PKG) to distribute private keys to each user, which is undesirable for a large network because PKG has to bear the heavy job of issuing all the keys. In order to reduce the workload on the root PKG, Gentry and Silverberg [1] introduced a hierarchical identity-based encryption (HIBE) scheme. Their scheme is collusion resistance to an arbitrary number of levels. For better performance, Boneh et al. [6] proposed an efficient HIBE scheme which requires only a constant length of ciphertext and a constant number of bilinear map operations during decryption. Later, Gentry and Halevi [7] obtained a fully secure HIBE scheme from identity-based broadcast encryption with key randomization. Lewko and Waters [8] achieved unbounded HIBE in which public parameters did not impose additional limitations on the functionality of the systems.

2.2. Attribute-based encryption

Sahai and Waters [2] introduced the notion of attribute based encryption (ABE), in which a message is encrypted under several attributes that compose a (fuzzy) identity. Based on their work, Goyal et al. [9] proposed a fine-grained access control ABE scheme, which supports any monotonic access formula. Their scheme is characterized as key-policy ABE (KP-ABE) in which the access structure is specified in the private key and the attributes are used to describe the ciphertext. Later, Ostrovsky et al. [10] allows for non-monotonic access structures. Bethencourt et al. [11] introduced a ciphertext-policy ABE (CP-ABE) scheme, in which the roles of the ciphertext and keys are reversed with regard to the KP-ABE scheme. Muller et al. [12] constructed an efficient distributed attribute-based encryption (DABE) scheme achieving disjunctive normal form (DNF) policy. Li et al. [13] obtained a fine-grained data access control systems with user accountability from ABE scheme to prevent illegal key sharing in the cloud. Recently, Wang
et al. [14] proposed a conjunctive fuzzy and precise identity-based encryption (FPIBE) scheme for secure data sharing in cloud servers, by combining the HIBE system with the CP-ABE system. Some schemes [15, 16] achieved constant-length ciphertext.

2.3. Broadcast encryption

The first broadcast encryption (BE) scheme was formally proposed by [3]. Later, [17] brought forward two subset-cover schemes that are suitable for the case of stateless receivers, namely “complete subtree” method and “subset difference” method. In subset-cover schemes, each user belongs to a set of subsets and stores all the keys corresponding to these subsets. The center selects proper subsets to cover all the innocent users but the revoked users, and uses the keys of selected subsets to encrypt the messages. [18] proposed “layered subset difference” method to reduce the transmission cost. [19] described “stratified subset difference” method. In these well-known schemes, the receivers are stateless. That is to say, keys for each user are fixed throughout the lifetime of the system, and receivers cannot record the operating states.

There are some schemes do not rely on subset cover. Schemes such as [20, 21, 22, 23] specified all the revoked users in the broadcast. Scheme such as [24, 25, 26, 27, 28] included the set of all the receivers in the broadcast. The information of the revoked users or the intended receivers is needed for calculating the session key.

Some schemes [29, 4, 30] considered the situations where users belonged to different privileges. Schemes for static privileged situations were designed in [29, 4], and dynamically privileged scheme was considered in [30]. However, these schemes considered only the line-structured privileges.

3. Protocol Model and Security Requirements

3.1. Protocol Model

- **Setup** is the algorithm for system setup. It will generate the public parameters PK and private keys SK for the system.

- **KeyGen** is the algorithm for key generation. It will generate decryption key DKu for each user u.
• **GroupInit** is the algorithm for initializing the access group of each user. It takes the total number of groups $Num_{group}$ as input, outputs a number of sets $G_1, \ldots, G_{Num_{group}}$. If a user is allowed to access the data for group $G_i$ $(1 \leq i \leq Num_{group})$, its identity is put into set $G_i$. Users in the same group $G_i$ share some common access policy $S_i$ (may be an ID for HIBE, or an access structure for ABE, or a set of designated users for BE).

• **Enc** is an algorithm for data encryption. It will take in an access policy $S$ and output a ciphertext header Hdr. The temporary session key TK is encapsulated in the header, and the message M are symmetrically encrypted using the session key to generate ciphertext body C.

• **Dec** is algorithm each user used to recover the message. It will take in an access policy $S$ and a decryption key $DK_u$ where $u$ satisfies $S$ ($u$ matches the ID for HIBE, or $u$’s attributes satisfies the access structure for ABE, or $u$ is in the set of designated users for BE), and output the recovered message M.

• **AddUser** is the algorithm which is used to handle the event that some user $u$ joins certain access group $G_i$, where $1 \leq i \leq Num_{group}$. Related ciphertext headers and ciphertext bodies are updated accordingly.

• **RemoveUser** is the algorithm which is used to handle the event that some user $u$ is removed from certain access group $G_i$, where $1 \leq i \leq Num_{group}$. Related ciphertext headers and ciphertext bodies are updated accordingly.

• **AddGroup** is the algorithm which is used to handle the event that the data owner adds certain access group $G_j$, where $j > Num_{group}$. Related ciphertext headers and ciphertext bodies are updated accordingly.

• **RemoveGroup** is the algorithm which is used to handle the event that the data owner removes certain access group $G_i$, where $1 \leq i \leq Num_{group}$. Related ciphertext headers and ciphertext bodies are updated accordingly.

3.2. **Security Requirements**

• **Correctness.** Each honest user is able to recover the messages for his/her groups and descending groups.
- **Semantic Secure.** The eavesdroppers cannot obtain any information of messages encrypted in the ciphertext.

- **Collusion Resistant.** Even all the revoked users collude, they cannot obtain any information of messages from the ciphertext.

- **Service Bounded.** Users are restricted to recover the messages encrypted for current access groups and descending groups, they cannot recover the messages encrypted for other groups.

### 3.3. Computational Square Root Exponent Problem

The computational square root exponent problem (CSREP) was introduced by Konoma et al. in [31], and Zhang et al. [32, 33] proved that this problem is equivalent to computational Diffie-Hellman problem (CDHP). Let \( G \) be a cyclic multiplicative group generated by \( g \) with the prime order \( q \). The definitions of CDHP and CSREP are described as follows:

**Definition 1 (CDHP).** Given \((g, g^a, g^b)\) for \(a, b \in \mathbb{Z}_q^*\) to compute \(g^{ab}\).

**Definition 2 (CSREP).** Given \((g, g^a)\) for \(a \in \mathbb{Z}_q^*\) to compute \(g^{a^{1/2}}\), if \(a\) is a quadratic residue modulo \(q\).

### 4. Generic Framework Suitable for Dynamically Privileged Data Sharing System

In this section, we describe a generic framework suitable for dynamically privileged data sharing system. The framework can adopt any data sharing scheme. We denote the employed data sharing scheme as \((\text{DS.Setup}, \text{DS.KeyGen}, \text{DS.Enc}, \text{DS.Dec})\). They are as follows.

- **DS.Setup** is the algorithm for system setup. It will generate the public parameters \(PK\) and private keys \(SK\) for the system. We denote it as \((PK, SK) \leftarrow \text{DS.Setup}()\).

- **DS.KeyGen** is the algorithm for key generation. It will generate decryption key \(DK_u\) for each user \(u\). We denote it as \((DK_u) \leftarrow \text{DS.KeyGen}(SK, u)\).
- **DS.Enc** is algorithm for data encryption. It will take in an access policy $S$ (an ID for HIBE, or an access structure for ABE, or a set of designated receivers for BE) which describes who will have access to the encrypted data, and output a ciphertext header $Hdr$ and a ciphertext body $C$. The temporary session key TK is encapsuled in the header, and the message M are symmetrically encrypted using the session key to generate ciphertext body. We denote the algorithm as $Hdr \leftarrow DS.Enc(PK, S, TK), C \leftarrow SEnc_{TK}(M)$, where $SEnc(\cdot)$ is the symmetric encryption algorithm.

- **DS.Dec** is algorithm each user used to recover the data. It will take in an access policy $S$ and a decryption key $DK_u$ where $u$ satisfies $S$ (if the ID for HIBE, or $u$'s attributes satisfies the access structure for ABE, or $u$ is in the set of designated receivers for BE), and output the recovered message $M$. We denote it as $TK \leftarrow DS.Dec(Hdr, S, DK_u), M \leftarrow SDec_{TK}(C)$. $SDec(\cdot)$ is a symmetric decryption algorithm corresponding to $SEnc(\cdot)$.

4.1. The Proposed Framework for Dynamic Tree-Structured Privileges

Based on any data sharing scheme (denoted as $DS$) without considering dynamic privileges, our proposed framework can achieve $O(T)$ sets of original ciphertext headers and ciphertext bodies, where $T$ is the number of access groups. It means that the ciphertext length is only dependent on the number of access groups, if the ciphertext length of $DS$ is constant, e.g. schemes in [6, 26, 15, 16]. We denote the dynamically privileged framework (DPF) as (DPF.Setup, DPF.KeyGen, DPF.GroupInit, DPF.Enc, DPF.Dec, DPF.AddUser, DPF.RemoveUser, DPF.AddGroup, DPF.RemoveGroup). This framework achieves dynamic tree-structured privileges, while previous schemes allow only dynamic line-structured privileges (e.g. scheme in [30]), or static privileges (e.g. HIBE schemes and ABE schemes). Our idea is inspired by a tool called chameleon hash function [34]. This tool can help us to generate different session keys for different groups with a same privilege in the same branch, and these keys can derive a same hash value which will be used as session key for the common descending group with a lower privilege. It is not trivial to fulfil the flexible tree-structured privileges as shown in Figure 3. Without loss of generality, we use the instance shown in Figure 4 to show how the keys are computed, which covers one-to-many, many-to-one and many-to-many cases. The algorithms are described as follows.
Figure 4: An instance for showing computations, with each arrow indicating the key deriving direction.

- **DPF.Setup** is the algorithm for system setup. By running \((PK, SK) \leftarrow DS.Setup()\), the data owner obtains \(PK\) as the public parameters and \(SK\) as private keys for the system. Additionally, the data owner selects a Gap Diffie-Hellman group \(G\) of some large prime order \(q\), as required for the chameleon hash function in [34]. \(g\) is a generator of \(G\). It also selects a random integer \(x \in \mathbb{Z}_q\), and computes \(y = g^x\). A one-way function \(f(\cdot) : G \rightarrow \mathbb{Z}_q\) is selected. The public parameters \((G, q, g, y, f(\cdot))\) are added to the parameters generated by DS.Setup(). We assume that all groups’ identities are elements from \(G\).

- **DPF.KeyGen** is the algorithm for key generation. We directly use the algorithm of DS.KeyGen(). By running \((DK_u) \leftarrow DS.KeyGen(SK, u)\), we generate decryption key \(DK_u\) for each user \(u\). The key is transferred to each user in a secure channel which is not considered in this paper.

- **DPF.GroupInit** is the algorithm for initializing the access groups of all users in the system. As shown in Figure 4, there are 7 access groups in the system. If a user \(u\) is allowed to access the data for group \(i (1 \leq i \leq 7)\), the identity of \(u\) is put into the set \(G_i\). Finally, \(\{G_1, \ldots, G_7\}\) and \(\{S_1, \ldots, S_7\}\) are output, with each \(G_i\) containing all the innocent users in the access group, and \(S_i\) is the common access policy for access group \(G_i\).

- **DPF.Enc** is algorithm for data sharing. The data owner computes as
follows:

1. It selects random integers \( TK_1, TK_5 \in \mathbb{Z}_q \) as temporary session keys for uppermost groups \( G_1 \) and \( G_5 \) respectively.

2. It selects random integers \( R_{1,2}, R_{1,3}, R_{1,4} \in \mathbb{Z}_q \), and computes \( y^{R_{1,2}}, y^{R_{1,3}} \) and \( y^{R_{1,4}} \). The temporary session keys for the nearest descending groups \( G_2, G_3 \) and \( G_4 \) are computed as
   \[
   TK_2 = f((g \cdot G_1)^{TK_1} \cdot y^{R_{1,2}}),
   \]
   \[
   TK_3 = f((g \cdot G_1)^{TK_2} \cdot y^{R_{1,3}}),
   \]
   \[
   TK_4 = f((g \cdot G_1)^{TK_3} \cdot y^{R_{1,4}}),
   \]
   where \( f(\cdot) \) is the one-way function and \( G_1 \) denotes the identity of group \( G_1 \). As \( G_1 \) is the uppermost group, the computations can be simply reduced to the one-way function without exponentiations. However, we use the form \( f((g \cdot G_i)^{TK_i} \cdot y^{R_{i,j}}) \) to be consistent with the computations of descending groups, which means the deriving is from \( G_i \) to \( G_j \) for descending branch \( k \). The branches are numbered from left to right as 1,2,3 and so on, with respect to the source group. For instance, the branch from \( G_1 \) to \( G_4 \) is numbered as 3 with respect to \( G_1 \).

3. It selects random integers \( R_{3,6} \in \mathbb{Z}_q \), and computes \( y^{R_{4,6}} \) as
   \[
   y^{R_{4,6}} = \frac{(g \cdot G_3)^{TK_3}}{(g \cdot G_4)^{TK_4}} \cdot y^{R_{3,6}},
   \]
   where \( y^{R_{4,6}} \) and \( y^{R_{4,6}} \) allow \( G_3 \) and \( G_4 \) to derive a same temporary session key for \( G_6 \) as
   \[
   TK_6 = f((g \cdot G_3)^{TK_3} \cdot y^{R_{3,6}}) = f((g \cdot G_4)^{TK_4} \cdot y^{R_{4,6}}),
   \]
   thanks for the help of chameleon hash function in [34].

4. It selects random integers \( R_{3,7} \in \mathbb{Z}_q \), and computes \( y^{R_{4,7}} \) and \( y^{R_{5,7}} \) as
   \[
   y^{R_{4,7}} = \frac{(g \cdot G_3)^{TK_2}}{(g \cdot G_4)^{TK_2}} \cdot y^{R_{3,7}},
   \]
   \[
   y^{R_{5,7}} = \frac{(g \cdot G_3)^{TK_2}}{(g \cdot G_5)^{TK_5}} \cdot y^{R_{3,7}},
   \]
so that $G_3$, $G_4$ and $G_5$ can derive a same temporary session key for $G_7$ as

$$TK_7 = f((g \cdot G_3)^{TK_3} \cdot y^{R_{3,7}}) = f((g \cdot G_4)^{TK_4} \cdot y^{R_{4,7}}) = f((g \cdot G_5)^{TK_5} \cdot y^{R_{5,7}}).$$

Then the ciphertext headers for all groups are generated as follows:

$$\text{Hdr}_1 = \text{DS.Enc}(PK, S_1, TK_1),$$
$$\text{Hdr}_2 = \text{DS.Enc}(PK, S_2, TK_2),$$
$$\ldots$$
$$\text{Hdr}_7 = \text{DS.Enc}(PK, S_7, TK_7).$$

($\text{Hdr}_1, \text{Hdr}_2, \ldots, \text{Hdr}_7$) are the ciphertext headers. The temporary session keys ($TK_1, \ldots, TK_7$) are used to encrypted messages for $G_1$ to $G_7$ respectively in symmetric encryption algorithm as follows:

$$C_1 = SEnc(M_1, TK_1),$$
$$C_2 = SEnc(M_2, TK_2),$$
$$\ldots$$
$$C_7 = SEnc(M_7, TK_7),$$

where $M_i$ ($1 \leq i \leq 7$) denotes the message for access group $i$ and $C_1, \ldots, C_7$ are the ciphertext bodies. ($y^{R_{1,2}}, y^{R_{1,3}}, y^{R_{1,4}}, y^{R_{3,6}}, y^{R_{3,7}}, y^{R_{4,6}}, y^{R_{4,7}}, y^{R_{5,7}}$) ($\text{Hdr}_1, \text{Hdr}_2, \ldots, \text{Hdr}_7$) and ($C_1, \ldots, C_7$) are stored in the cloud servers. As we notice that, encryption schemes without considering dynamic flexible privileges will result in 7 ciphertext headers and 17 ciphertext bodies, which will cost much more storage than our framework.

- **DPF.Dec** is algorithm for decryption. Suppose user $u$ belongs to access group $G_1$. $DK_u$ is used to recover temporary session key $TK_1$ as $TK_1 = \text{DS.Dec}(\text{Hdr}_1, S_1, DK_u)$. User $u$ can derive temporary session keys for descending groups as follows:

$$TK_2 = f((g \cdot G_1)^{TK_1} \cdot y^{R_{1,2}}),$$
$$TK_3 = f((g \cdot G_1)^{TK_1^2} \cdot y^{R_{1,3}}),$$
$$TK_4 = f((g \cdot G_1)^{TK_1^3} \cdot y^{R_{1,4}}),$$
$$TK_6 = f((g \cdot G_3)^{TK_3} \cdot y^{R_{3,6}}),$$
$$TK_7 = f((g \cdot G_3)^{TK_3^2} \cdot y^{R_{3,7}}).$$
Then user \( u \) can recover messages \( M_1, M_2, M_3, M_4, M_6, M_7 \) with a symmetric decryption algorithm using these keys.

- **DPF.AddUser** is the algorithm which is used to handle the event that some user \( u \) joins certain access group \( G_i \), where \( 1 \leq i \leq Num_{init} \). The identity of \( u \) is put into the set \( G_i \), and \( S_i \) is the updated common access policy for the new access group \( G_i \). The ciphertext header \( Hdr_i \) is updated as \( Hdr_i = DS.Enc(PK, S_i, TK_i) \). Other ciphertext headers and ciphertext bodies remain unchanged.

- **DPF.RemoveUser** is the algorithm which is used to handle the event that some user \( u \) is removed from certain access group \( G_i \), where \( 1 \leq i \leq Num_{init} \). We assume that user \( u \) is removed from the system when considering the algorithm **DPF.RemoveUser**, and we can regard the case that \( u \) is moved from group \( G_i \) to \( G_j \) as that \( u \) is added to \( G_j \) by the algorithm **DPF.AddUser** after it is removed from the system.

We define the co-parent groups of group \( G_i \) as other parent groups of \( G_i \)’s nearest descending groups. For instance, \( G_4 \) is the co-parent group of \( G_3 \) with respect to the descending group \( G_6, G_4 \) and \( G_5 \) are the co-parent groups of \( G_3 \) with respect to \( G_7 \). We set \( \Omega \) to an empty set, and add \( G_i \), \( G_i \)’s descending groups, co-parent groups of \( G_i \) to it. If there are other descending groups for groups in \( \Omega \) that are not included in \( \Omega \), we add them to \( \Omega \). If there are other co-parent groups for groups in \( \Omega \) that are not included in \( \Omega \), we add them to \( \Omega \). The operations are repeated until \( \Omega \) is never changed. The temporary session keys, the ciphertext headers, ciphertext bodies of \( \Omega \) and the random values concerning \( \Omega \) should be renewed. There are 7 cases for the instance shown in Figure 4:

1. case 1: \( u \) is removed from \( G_1 \). All temporary session keys, random values, ciphertext headers and ciphertext bodies should be renewed.
2. case 2: \( u \) is removed from \( G_2 \). A new random value \( R_{1,2} \) is selected to compute new \( TK_2 \), \( Hdr_2 \) and \( C_2 \).
3. case 3: \( u \) is removed from \( G_3 \). New random values \( R_{1,3} \) and \( R_{1,4} \) are selected to compute temporary session keys, random values, ciphertext headers and ciphertext bodies corresponding to \( G_3, G_4, G_5, G_6 \) and \( G_7 \).
4. case 4: $u$ is removed from $G_4$. The computations are the same as case 3.

5. case 5: $u$ is removed from $G_5$. The computations are the same as case 3.

6. case 6: $u$ is removed from $G_6$. A new random value $R_{3,6}$ is selected to compute new $R_{4,6}$, $TK_6$, $Hdr_6$ and $C_6$.

7. case 7: $u$ is removed from $G_7$. A new random value $R_{3,7}$ is selected to compute new $R_{4,7}$, $R_{5,7}$, $TK_7$, $Hdr_7$ and $C_7$.

\* **DPF.AddGroup** is the algorithm which is used to handle the event that the data owner adds certain access group $G_j$, where $j > Num_{group}$. $Num_{group}$ is increased accordingly. There are four cases:

1. case 1: $G_j$ is a descending group of some group $G_i$. A new random value $R_{i,j}$ is selected to compute new $TK_j$, $Hdr_j$ and $C_j$.

2. case 2: $G_j$ is a descending group of some groups $G_i$, $G_t$. A new random value $R_{i,j}$ is selected to compute new $TK_j$, $Hdr_j$, $C_j$ and $R_{t,j}$.

3. case 3: $G_j$ is a co-parent group of some group $G_i$ with respect to the descending group $G_t$. A new random value $TK_j$ is selected to compute new $Hdr_j$ and $C_j$, and $R_{j,t}$ is computed from $TK_i$, $R_{i,t}$ and $TK_j$.

4. case 4: $G_j$ is the only parent group of some group $G_i$. A new random value $TK_j$ is selected to compute new $Hdr_j$ and $C_j$. $R_{j,i}$ can not be computed easily if $f(\cdot)$ is not a trapdoor one-way function, so $\Omega$ may be selected with respect to $G_i$ as described in algorithm **DPF.RemoveUser**. Then the temporary session keys, the ciphertext headers, ciphertext bodies of $\Omega$ and the random values concerning $\Omega$ should be renewed.

\* **DPF.RemoveGroup** is the algorithm which is used to handle the event that the data owner removes certain access group $G_i$, where $1 \leq i \leq Num_{group}$. $Num_{group}$ is decreased accordingly. This algorithm is the similar to the algorithm **DPF.RemoveUser**, since removing an access group means removing a set of users except that $TK_i$, $Hdr_i$, $C_i$ and the related random values $R_{i,*}$ are no longer needed.
4.2. Security Analysis

**Correctness.** The correctness is straightforward due to the above description.

**Theorem 1.** The dynamically privileged framework is semantically secure against the eavesdroppers assuming the employed data sharing scheme and the symmetric encryption algorithm are semantically secure.

**Proof:** (Sketch Proof.) The dynamically privileged framework uses the data sharing scheme to encrypt the session key for each access group. If some one who is not included in an access group can obtain information about the encrypted key for that group, it raises a contradiction to the semantic security of the data sharing scheme.

Only the users who are included in a service group can recover the corresponding messages (actually recover the session key and then the messages). If any eavesdropper can learn some thing about a message $M_i$ from the ciphertext $SEnc(M_1, TK_1), \ldots, SEnc(M_{Num_{group}}, TK_{Num_{group}})$, with $1 \leq i \leq Num_{group}$, then he/she can be used to break the semantic security of symmetric encryption algorithm, with regard to the ciphertext $SEnc(M_i, TK_i)$, where $TK_i$ is unknown to the eavesdroppers.

**Theorem 2.** The dynamically privileged framework is collusion resistant assuming the employed data sharing scheme is collusion resistant.

**Proof:** (Sketch Proof.) The dynamically privileged framework uses the data sharing scheme to encrypt the session key for each access group. If all colluding users who are not included in an access group can obtain information about the encrypted key for that group, it raises a contradiction to the collusion resistance of the data sharing scheme.

**Theorem 3.** In our framework, the users are service bounded, i.e. each user is restricted to recover the messages for his/her group and descending groups, assuming the chameleon hash function [34] is unforgeable, CSREP is hard in $G$, the key extraction function (from a group with higher privilege to a group with lower privilege) is one-way and the symmetric encryption algorithm is semantically secure.

**Proof:** Without loss of generality, we use the instance shown in Figure 4 to prove the service-bounded security. We consider the security in several cases:
Case 1: The adversary holds a session key and attempts to recover the ascending session key. Suppose the adversary holds a session key of access group $G_i$ and wants to recover the messages for class $G_j$, where $G_j$ is the ascending group of $G_i$. The session key $TK_j$ for class $j$ is randomly selected. The adversary is allowed to query the the derived key $TK_i$ and encryption of messages for class $j$. The challenge for the adversary is to decide whether a ciphertext $c_j$ is encrypted by $TK_j$ on the message $M$ selected by himself/herself, or on a random ciphertext. Our proof consists of a sequence of hybrid games, starting with the real attack and ending in a game in which the adversary’s advantage is zero. We use $S_n$ to denote $Game_n$ and $Pr[S_n]$ to denote the probability of the adversary to win the $Game_n$, with $n = 0, 1, 2, \ldots$.

Game0: This game corresponds to the real game, in which $TK_i$ is derived from $TK_j$ and ciphertext $c_j$ is encrypted by $TK_j$ on the message $M$ selected by the adversary or a random ciphertext. By definition, the advantage of the adversary $A$ is

$$ADV_A = Pr[S_0] - 1/2. \quad (1)$$

Game1: In this game, the $TK_i$ is randomly selected and is not derived from $TK_j$. That is to say, the adversary is deprived of the advantage of obtaining the key information from $TK_i$. Therefore, the difference between the success probability of the adversary $A$ for current and previous games is at most the advantage of breaking the one-way property of key extraction function $f(\cdot)$, denoted as $ADV_A(f(\cdot))$. Thus, we have

$$|Pr[S_1] - Pr[S_0]| \leq ADV_A(f(\cdot)). \quad (2)$$

Game2: In this game, the ciphertext $c_j$ is randomly selected for the message $M$ and is not really encrypted by $TK_j$ on the $M$ selected by the adversary. Therefore, the difference between the success probability of the adversary $A$ for current and previous games is at most the advantage of breaking the semantic security of symmetric encryption algorithm, denoted as $ADV_A(SE)$. Thus, we have

$$|Pr[S_2] - Pr[S_1]| \leq ADV_A(SE). \quad (3)$$
As we notice that, the randomly selected ciphertext $c_j$ for the message $M$ has nothing to do with the $M$. So the probability of the adversary $A$ to win is 1/2. That is $Pr[S_2] = 1/2$. Then we have

$$ADV_A = Pr[S_0] - 1/2$$

$$\leq Pr[S_1] + ADV_A(f(\cdot)) - 1/2$$

$$\leq Pr[S_2] + ADV_A(SE) + ADV_A(f(\cdot)) - 1/2$$

$$\leq ADV_A(SE) + ADV_A(f(\cdot))$$

If $A$ has non-negligible probability in recovering the messages for class $j$, we can take advantage of $A$ to break the one-way property of key extraction function $f(\cdot)$ or the symmetric encryption algorithm.

- **Case 2:** The adversary holds a session key, and wants to recover the session key of co-parent group. The event that the adversary knowing a session key wants to recover the session key of a descending group’s co-parent group also belongs to this case. For instance in Figure 4, the adversary holding session key $TK_3$ of group $G_3$ wants to recover the session key $TK_4$ of group $G_4$. In this case, the adversary knows the common chameleon hash values $(g \cdot G_3)^{TK_3} \cdot y^{R_{3,6}} = (g \cdot G_4)^{TK_4} \cdot y^{R_{4,6}}$, and $(g \cdot G_3)^{TK_3} \cdot y^{R_{3,7}} = (g \cdot G_4)^{TK_4} \cdot y^{R_{4,7}}$. If the adversary can obtain the session key $TK_4$ of group $G_4$, it raises a contradiction to the unforgeability of the chameleon hash function (Theorem 3 in [34]).

- **Case 3:** The adversary holds a session key, and wants to recover the session key of co-parent group’s another descending group. For instance, the event knowing $TK_2$ to recover $TK_3$ belongs to this case. In this case, the adversary knows one common chameleon hash value among $G_3$, $G_4$ and $G_5$ that $(g \cdot G_3)^{TK_2} \cdot y^{R_{3,7}} = (g \cdot G_4)^{TK_2} \cdot y^{R_{4,7}} = (g \cdot G_5)^{TK_2} \cdot y^{R_{5,7}}$, so the adversary knows $(g \cdot G_3)^{TK_2}$ and $(g \cdot G_4)^{TK_2}$. However, it is hard for the adversary to obtain $(g \cdot G_3)^{TK_2}$ or $(g \cdot G_4)^{TK_2}$, or it can be used to break CSREP. The probability that the adversary guesses the right session key $TK_6$ without knowing the source value of the one-way function is at most $1/q$, which is negligible for a $q$ that is large enough.

- **Case 4:** Cases other than the above three cases. For instance, the event that knowing $TK_2$ to recover $TK_3$ belongs to this case. The probability
that the adversary guesses the right session key is at most $1/q$, which is negligible for a $q$ that is large enough.

Therefore, the users are service bounded.

5. Efficiency Evaluation

We compare our framework with previous related schemes ([4, 13, 30]) on storage cost (on cloud servers), user storage (refers to private key size in asymmetric encryption schemes), center storage (refers to public key size in asymmetric encryption schemes), encryption cost and decryption cost. The comparison is shown in Table 1. We suppose that there are totally $N$ users, $r$ ($r < N$) revoked users and $N_G$ ($N_G < N - r$) access groups in each scheme.

As we notice that, our proposal is efficient in storage cost, which is dependent only on the number of access groups in the system and the ciphertext length of the employed data sharing scheme. It is as efficient as ABE-based data sharing schemes (e.g.[13, 14]) which achieve static privileges only. If the ciphertext length of the employed data sharing scheme is constant, the storage cost is dependent only on the number of access groups in the system. In other words, our scheme can turn a data sharing scheme (e.g.[24, 13, 14]) into a dynamically privileged scheme with a little extra cost in encryption. Moreover, our framework is efficient in decrypting messages in descending groups since it needs only $O(1)$ computations, while it may not be constant in the employed data sharing schemes. For instance, it needs $O(|S|)$ computations in [24], where $|S|$ is the number of users in the receiver set $S$. It needs computations in proportion to the average number of wildcards used in the access policies for scheme in [13], and computations in proportion to the number of attributes described in access structure for the attribute-based decryption in [14].

We also notice that subset-cover based schemes [29, 4, 30] are efficient in decryptions. However, the storage cost in previous static schemes [29, 4] is dependent on the number of revoked users or the number of access groups, and the storage cost in previous dynamic schemes [30] is dependent on the number of access groups, the number of revoked users and the total number of users (described in SubSection 5.2 of [30]). Moreover, our framework allows flexible tree-structured privileges, while these works allow only line-structured privileges, i.e. one group in each privilege.
<table>
<thead>
<tr>
<th>Privileged Structure</th>
<th>Storage Cost in Cloud Servers</th>
<th>User Storage</th>
<th>Center Storage</th>
<th>Encryption Cost</th>
<th>Decryption Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4] Static Line</td>
<td>$O(\text{Max}(2r, N_G))$</td>
<td>$\frac{1}{2} \log^2 N + \frac{1}{2} \log N + 1$</td>
<td>$(\frac{\log N}{4} - 1)$</td>
<td>$O(\text{Max}(2r, N_G))$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>[13] Static Tree</td>
<td>$O(N_W \cdot N_G)$</td>
<td>$O(1)$</td>
<td>$O(N_A)$</td>
<td>$O(N_A \cdot N_G)$</td>
<td>$O(N_W)$</td>
</tr>
<tr>
<td>[30] Dynamic Line</td>
<td>$r \log \left( \frac{N}{r} \right) \leq c \leq \log N$</td>
<td>$2N-1$</td>
<td>$O(c)$</td>
<td>$O(1)$</td>
<td></td>
</tr>
<tr>
<td>Ours Dynamic Tree</td>
<td>$O(C_1 \cdot N_G)$</td>
<td>$C_2$</td>
<td>$C_3$</td>
<td>$O((C_4 + 1) \cdot N_G)$</td>
<td>$C_5^1$</td>
</tr>
</tbody>
</table>

$\dagger$: reduced to $O(1)$ in descending groups;
$r$: the number of revoked users;
c: the ciphertext length in the scheme of [30];
$N$: the number of total users;
$N_A$: the number of attributes in the system;
$N_W$: the average number of wildcards used in the access policies;
$N_G$: the number of groups in the system;
$C_1$: the ciphertext length of the employed data sharing scheme;
$C_2$: the private key size of the employed data sharing scheme;
$C_3$: the public key size of the employed data sharing scheme;
$C_4$: the encryption cost of the employed data sharing scheme;
$C_5$: the decryption cost of the employed data sharing scheme.
6. Conclusion

Inspired by a tool called chameleon hash function [34], we present an efficient framework for data sharing system to achieve dynamic privileges. Using this framework, any data sharing and access control scheme can be turned into a dynamic flexible tree-structured privileges scheme, while previous schemes achieve only static privileges or dynamic line-structured privileges. In our framework, the data owner can change the access group of each user dynamically and change the structure of privileges flexibly when it is needed.

Acknowledgement

This work is supported by the Fundamental Research Funds for the Central Universities (No. ).

References


