10-2004

Dynamic Access Control for Multi-Privileged Group Communications

Di MA
Institute for Infocomm Research

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Yongdong WU
Institute for Infocomm Research

Tieyan LI
Institute for Infocomm Research

DOI: https://doi.org/10.1007/978-3-540-30191-2_39

Dynamic Access Control for Multi-Privileged Group Communications

Di Ma[1], Robert H. Deng[2], Yongdong Wu[1], Tieyan Li[1]

[1]Institute for Infocomm Research
21 Heng Mui Keng Terrace, Singapore 119613
{madi,wydong,litieyan}@i2r.a-star.edu.sg
[2]School of Information Systems
Singapore Management University
469 Bukit Timah Road, Singapore 259756
{robertdeng}@smu.edu.sg

Abstract. Recently, there is an increase in the number of group communication applications which support multiple service groups of different access privileges. Traditional access control schemes for group applications assume that all the group members have the same access privilege and mostly focus on how to reduce rekeying messages upon user joining and leaving. Relatively little research effort has been spent to address security issues for group communications supporting multiple access privileges. In this paper, we propose a dynamic access control scheme for group communications which support multiple service groups with different access privileges. Our scheme allows dynamic formation of service groups and maintains forward/backward security when users switch service groups.

1 Introduction

With the rapid progress in technologies underlying multicast networking, group communication applications such as video conferencing, live sports, concerts broadcasting, are gaining popularity. For the purpose of security or billing, many access control schemes [1] - [9] have been proposed to prohibit unauthorized access to group communications. With the development of scalable video coding which enables users with different preferences, privileges or capabilities to access different parts of a video stream, group communication applications begin to support multiple service groups with different access privileges. As traditional access control schemes are designed to tackle security problems in single-privileged group communications, they cannot be applied to address new security issues, such as privilege change and dynamic service group formation, encountered in multi-privileged group communications. In this paper, we propose a dynamic access control scheme for group communications supporting multi-privileged service groups. Our scheme allows dynamic formation of service groups and maintains forward/backward security when users switch service groups.

1.1 Single Privileged Access Control

Traditional access control for group communications treats all the users in a multicast group with exactly the same access privilege. A group key used to encrypt communication traffic is established and shared by all the group members. The group key or the content encryption key (CEK) is established either by a centralized server or by combining contributory parts from all the group members. Schemes involving a centralized key server are called centralized key management schemes and the centralized key server is called key distribution center (KDC). To securely and efficiently distribute a group key to all the legal participants, a set of key encryption keys (KEKs) are created to encrypt the group key and other control data. A major concern in centralized key management schemes is how to update the keys (both CEK and KEKs) efficiently to accommodate membership changes upon user join/leave while preserve the forward/backward security.

Several schemes [1] - [6] used a tree-based approach to manage keys as well as to reduce communication, computation and storage cost on maintaining keying and rekeying materials. Wong *et al.* [5] performed an extensive theoretical and experimental analysis on various types of *key graphs* and concluded that the most efficient key graph for group key management is a d-degree tree. A typical 2-degree tree for key management is shown in Figure 1. The leaf nodes of the key tree are associated with private keys of the group members. The root of the key tree is the group key or the CEK which is used to encrypt and decrypt data traffic for the group. The intermediate nodes are associated with a set of KEKs which are used to encrypt the CEK and other KEKs to provide secure update of the CEK among all the legal group members. Thus each group member possesses a private key (which is the shared secret between the KDC and the user), the CEK and a set of KEKs along the path from the leaf node associated with it to the root of the key tree. The total number of keys stored by the KDC is approximately $(dn-1)/(d-1)$ and the total number of keys stored by each member is $\log_d n + 1$. The tree-based key management scheme can update keys using $d \log_d n$ messages.

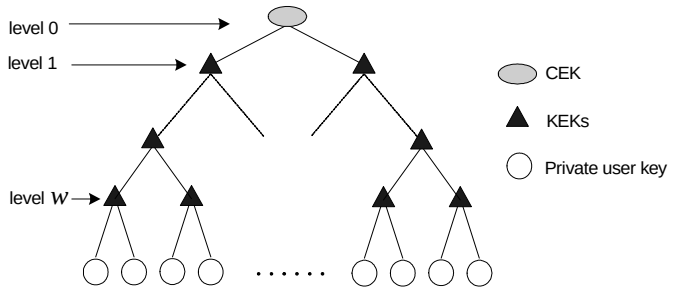

Fig. 1. A typical key management tree for a single service group.

To further reduce the number of rekey message transmissions, in [2], each key is identified by a *revision* field. When a user joins, it is positioned by the KDC into a new leaf node. The KDC increases the *revision* of all the keys to be transmitted to the new participant by passing all the keys through a one-way function. The KDC also informs all the existing participants about the use of the new keys. The existing participants will notice the *revision* change visible in ordinary data packets, and thus pass their keys through the same one-way function. Therefore, there is no need to transmit additional messages when a user joins.

1.2 Static Multi-Privileged Access Control

A number of works [10]-[15] relating to static multi-privileged access control have been proposed. Almost all of them assume that information items as well as users are classified into a certain type of hierarchy and there is a relationship between the encryption key assigned to a node and those assigned to its children. They do not address the dynamic membership problem which is critical in group communications.

Recently scalable video coding has gained increasing acceptance due to its flexibility and good adaptability to network bandwidth and end-user capability. Scalable access control schemes are required to protect the communication data as well as to preserve the scalable features of multimedia streams. There are a couple of recently reported schemes [16]-[17] that were specifically designed for scalable multimedia applications.

Scalable access control is usually achieved through scalable encryption. Unlike encryption of non-scalable streams where a unique key is used through out the entire communication process, scalable encryption encrypts each scalable unit in a frame using a separate unit encryption key. A scalable unit is a segment of the stream data and associated with a certain service level, one-dimensional or multi-dimensional. In [16], an MPEG-4 FGS video frame, supporting T PSNR service levels and M bitrate service levels, is divided into $T \times M$ different two-dimension units. In [17], a single tile JPEG2000 frame can support 4-dimensional scalability innately: resolution, quality, component and precinct. If a JPEG2000 frame has N_R resolution levels, N_L quality layers, N_C components and N_P precincts in each resolution, there will be $N_R \times N_L \times N_C \times N_P$ scalable units and the same number of unit encryption keys are needed to encrypt them.

The example MPEG-4 FGS stream in [16] can provide about $T \times M$ access levels which allow the formation of $T \times M$ service groups. Let the term "service group" denote the group of participants who have exactly the same access privilege. Different service groups will have different number of unit encryption keys to decrypt the authorized units. A service group with a full access privilege will possess all the $T \times M$ keys. A service group with access to all the PSNR levels with the lowest bitrate level will possess T keys.

A major difference between [16] and [17] is that [17] uses a tree-based key management scheme to generate the scalable unit keys and manage the unit keys much more efficiently while [16] generates the unit keys independently and does

not care how to reduce the communication cost to manage and transmit these keys. In [17], the key server only needs to send one key, the *resolution 0* key, to the service group which has an access privilege to *resolution 0* of a RLCP ordered JPEG2000 stream. The group members then use this resolution key to generate $N_L \times N_C \times N_P$ unit encryption keys.

Like all the classical hierarchical access control schemes in [10]-[15], these two scalable access control schemes both work in static scenarios where the dynamic features of realtime group communication are not addressed.

1.3 Dynamic Multi-Privileged Access Control

We identify in this paper two new places where security should be addressed in the studies of group communication applications which support multi-privileged service groups. One is privilege change and the other is dynamic service group formation.

It is very natural for a user to change access privilege by switching from one service group to another. For example, during a live concert broadcasting, a user subscribing to both video and audio may want to switch to the audio only service group as he just want to enjoy the music; a user subscribing to a high quality service group may want to change to a low quality service group simply because his favorite singer has finished her performance.

As stated in the previous section, group communication applications with multi-dimension scalable video allow the formations of many service groups. For example, a motion JPEG2000 stream, if each frame has 6 resolutions, 5 quality layers, 3 components and $16(4 \times 4)$ precincts in each resolution, theoretically speaking, allows the formation of $6 \times 5 \times 3 \times 16 = 1440$ service groups. Some considerations should be taken into account when one designs an access control scheme capable of handling a lot of service groups. Firstly, it is not wise to design an access control scheme which takes into account all the service groups in advance. As handling each service group consumes certain resources, a full-fledged scheme is definitely complex and might be too expensive for an application which only has a small number of participants. Secondly, it is not flexible to fix the service groups in advance either. By fixing the number of service groups in advance, the scheme weakens the scalability of the stream. Last but not least, not all of the service groups will have subscribers. Both the full-fledged scheme and the fixed scheme waste resources on handling service groups which have no subscribers. Thus an ideal scheme should support dynamic service group formation so that it can handle the basic set of service groups at the beginning while at the same time it can be extended easily to a full-fledged scheme when necessary.

While writing this paper, a hierarchical access control scheme that supports dynamic user privilege relocation was presented in [18]. The scheme integrates the multicast key tree structure with the hierarchical access control structure to form an integrated key graph to maintain all the keying materials for all the members. It uses 3 steps to construct the key management graph. Firstly, it constructs a subtree for each service group with leaves as group members. Then it constructs a subtree for each data group with leaves as service groups which

have access to this data group. Finally it combines the subtrees of service groups and the subtrees of data groups together to form an integrated key graph. The scheme supports dynamic privilege change when a participant switches from one service group to another. However, the scheme is more suitable for group communications where the number of service groups is fixed and the data stream is scalable in one-dimension. The scheme is not flexible for dynamic service group formation and decomposition, and is cumbersome in handling a lot of service groups.

1.4 Our Scheme

In this paper, we propose a dynamic access control scheme for group communications which supports multiple dynamic service groups. This scheme extends the traditional multicast key management tree to a key management graph to accommodate dynamic groups and uses two fields *version* and *revision* to eliminate or reduce the rekey messages upon a user join, leave or switch operation. The rest of the paper is organized as follows. Section 2 introduces the scheme as well as several examples to illustrate our rekeying algorithm. Section 3 gives the security analysis of the scheme in terms of forward and backward security. Section 4 presents the performance of the scheme in terms of storage overhead and rekeying overhead. Finally, conclusion is drawn in Section 5.

2 Dynamic Access Control Scheme

2.1 The Key Management Graph

The key management graph supporting multi-privileged service groups is shown in Figure 2. Each service group forms a subtree whose leaf nodes are the participants in this group and whose root is associated with an access key (AK) set. An AK set is a subset of the CEK set. The CEK set consists of all the unit encryption keys of a scalable stream. An AK set is possessed by a service group and represents an access privilege. Unlike the traditional multicast key management tree shown in Figure 1 where the root of the group tree is a single CEK, here the root of each subtree is associated with an AK set. We call the KEK right below the root node of the subtree as the service root key (SRK).

Let Ω_C=$\{ck_0,\ ck_1,\ldots,ck_{N-1}\}$ denote the set of CEKs which contains N separate unit encryption keys. Suppose that there are I service groups, $S_i, i = 1, 2, \ldots, I$. Participants in the same service group have exactly the same access privilege. Each service group S_i is associated with an AK set Ω_i, $\Omega_i \subseteq \Omega_C$. Let srk_i denote the SRK of S_i. Each participant in S_i possesses a private key, a set of KEKs from the key represented by the immediate node above the leaf node to srk_i and an AK set Ω_i.

Let $S_i \rightarrow S_j$ denote a switch from S_i to S_j. We use the service group S_0 to denote a virtual group which has no access privilege, that is $\Omega_0 = \phi$ where ϕ denotes null. With this notation, a general *switch* can be defined as follows. A

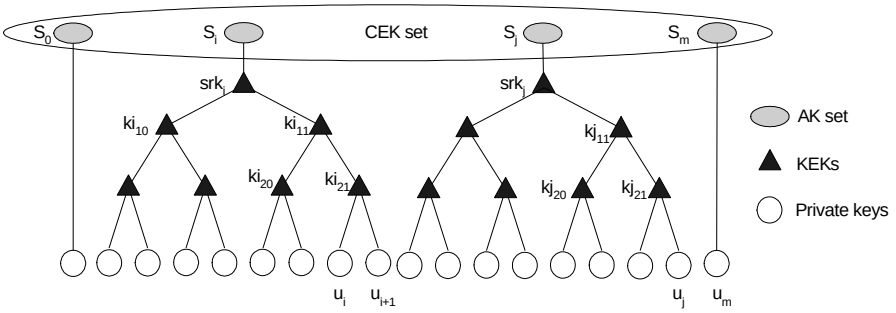

Fig. 2. Key management graph supporting multiple service groups.

join can be viewed as a user switching service groups from S_0 to S_i, $S_0 \rightarrow S_i$; a dynamic group *formation* is a user switch $S_0 \rightarrow S_i$ when there is no participant in S_i; a *leave* can be viewed as a user switches service groups from S_i to S_0, $S_i \rightarrow S_0$; and a dynamic group *decomposition* is a user switch $S_i \rightarrow S_0$ when there is no more participant left in S_i.

The KEKs in each service group are organized in a balanced d-degree tree as in a traditional multicast session [1] - [6]. The CEKs are arranged in a flat manner.

2.2 Identification of a Key

As in [2] every CEK and KEK in our scheme is addressed through a *key selector*, consisting of a unique key *ID*, a *version* field and a *revision* field. The key *ID* uniquely identifies a key and remains unchanged even if the secret keying material changes. The *version* and *revision* fields reflect the change of the secret keying material. Unlike [2] where only the *revision* field is used to eliminate the need for sending rekey messages, our scheme uses both fields for the same purpose.

The *revision* is increased whenever the key is passed through a one-way function. When a participant notices the increment of the *revision* of a key in his possession he will update the key through the same one-way function. The *version* field is increased whenever a new keying secret is sent out by the KDC and the key is passed through a keyed one-way function. When a participant, after receiving the new keying secret in advance, notices the increment of the *version* of a key in his possession he will update the key through the same keyed one-way function.

2.3 Rekeying Algorithm

A rekeying operation is executed when a general switch happens to provide forward security so that the new joining user is unable to decrypt the previous communication data correctly in the new joining group, as well as to provide

backward security so that the leaving user is no longer able to decrypt the future communication data correctly in the departed group. Let the switch be $S_i \rightarrow S_j$, the rekeying algorithm consists of 4 steps in a sequence:

1. Update of the KEKs in S_i from the departed user to srk_i.
   The KDC generates new KEKs along the path from the departed user to srk_i in the subtree of S_i. Suppose the subtree of S_i is a full-loaded balanced tree with w_i in depth. This will result in up to $dw_i - 1$ rekey messages being sent out.
2. Update of the unit encryption keys in $\Omega_i \cap \overline{\Omega_j}$.
   Let $|\Omega|$ denote the number of elements in the key set Ω. To update $|\Omega_i \cap \overline{\Omega_j}|$ unit encryption keys, firstly the KDC generates a secret ck_s. Then the KDC updates the keys in $\Omega_i \cap \overline{\Omega_j}$ through a keyed one-way function so that $k' = H_{ck_s}(k)$ (where $k \in \Omega_i \cap \overline{\Omega_j}$ and k' denotes the updated version of key k) and increases the *version* fields of those new keys. Then for any service group S_l including S_i that $\Omega_l \cap (\Omega_i \cap \overline{\Omega_j}) \neq \phi$, the KDC sends out the rekey message $\{ck_s\}_{srk_l}$. After obtaining ck_s, the affected users will know the key change when the data packet indicating the increase of the *version* numbers first arrives, and compute the new keys using the same keyed one-way function $H_{ck_s}(\cdot)$. Suppose there are u such service groups, this step results in u rekey messages being sent out.
3. Update of the KEKs in S_j from the new joining user to srk_j.
   The KDC chooses a leaf position on the subtree of S_j to position the joining user. The subtree can be either partially-loaded or fully-loaded. If the subtree is partially-loaded as shown in Figure 3(a) where there is an intermediate node which has j children and $j < d$, the KDC updates all the existing KEKs along the path from the new leaf to srk_j by generating the new keys from the old keys using a one-way function so that $k' = H(k)$ and increases the *revision* field of all the updated keys. All the new KEKs are encrypted by using the new joining user's private key and sent out to the new joining user. No rekey messages are necessary for delivering the new KEKs to the exiting users as they will know about the key change when the data packet indicating the increase of the *revision* numbers first arrives, and compute the new keys using the same one-way function $H(\cdot)$ by themselves.
   If the subtree is fully-loaded as shown in Figure 3(b), a leaf node is chosen and split to accommodate the new joining user. The KDC need generate a new KEK for these two leaf nodes. This step results in two rekey messages being sent out, one for sending all the updated existing KEKs and the new KEK to the new joining user, and the other is for sending the new KEK to the split user.
4. Update of the unit encryption keys in $\overline{\Omega_i} \cap \Omega_j$.
   Update of the unit encryption keys in $\overline{\Omega_i} \cap \Omega_j$ follows a similar way as in Step 3 through a one-way function $H(\cdot)$ and the increase of the *revision* field. All the affected users will update these keys by themselves when they notice the increase of the *revision* numbers through regular data packets. No rekey message is sent out in this Step.

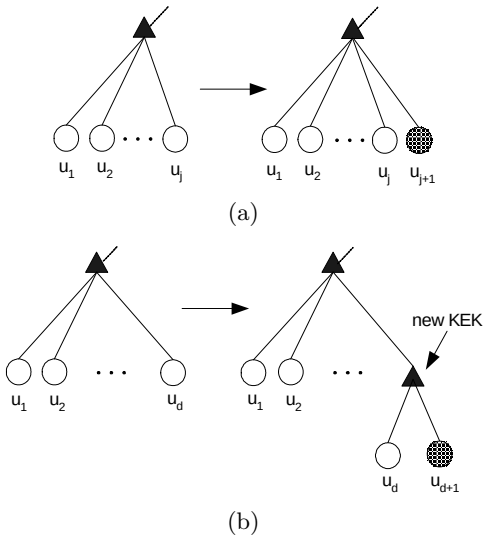

Fig. 3. User joins (a) a partially-loaded tree (b) a fully-loaded tree

The 4-Step rekeying algorithm stated above can be simplified for the following special situations:

- In the case when a new user joins an existing group S_i ($S_0 \rightarrow S_i$), because there is no KEK in S_0 and $\Omega_0 \cap \overline{\Omega_i} = \phi$, the key update process only needs to do Step 3 and Step 4.
- In the case when a new user forms a new group S_k ($S_0 \rightarrow S_k$), because there is no KEK in both S_0 and S_k and $\Omega_0 \cap \overline{\Omega_k} = \phi$, the key update process only needs to do Step 4.
- In the case when a user leaves a group S_i which has more than one participant ($S_i \rightarrow S_0$), because there is no KEK in S_0 and $\Omega_0 \cap \overline{\Omega_i} = \phi$, the key update process only needs to do Step 1 and Step 4.
- In the case when the last user leaves a group S_k ($S_k \rightarrow S_0$) and the group S_k decomposes, because there is no KEK in both S_0 and S_k and $\Omega_0 \cap \overline{\Omega_i} = \phi$, the key update process only needs do Step 4.
- In the case when a user switch from group S_i to a lower privilege group S_j ($S_i \rightarrow S_j$) such that $\Omega_i \supset \Omega_j$, because $\overline{\Omega_i} \cap \Omega_j = \phi$, the key update process only needs to do Step 1 to Step 3.
- In the case when a user switch from group S_i to a higher privilege group S_j ($S_i \rightarrow S_j$) that $\Omega_i \subset \Omega_j$, because $\Omega_i \cap \overline{\Omega_j} = \phi$, the key update process only needs do Step 1, Step 3 and Step 4.

We use two examples to illustrate this rekey algorithm for new group formation and service group switch. Examples given are for multimedia delivery, but the technique is potentially useful in other scenarios as well. Suppose there

is a scalable video with 2 resolution levels and 3 quality layers. In each frame there are total 6 scalable units arranged in RL (resolution-layer) order. Let $\Omega_C = \{ck_{00}, ck_{01}, ck_{02}, ck_{10}, ck_{11}, ck_{12}\}$ be the CEK set for this scalable video stream. The key ck_{ij} allows a user to access a scalable unit of resolution i and quality layer j. Keys ck_{ij} $(i = 0, \cdots, r,\ j = 0, \cdots, l)$ allow a user to access the video stream at resolution r with l quality layers. In the initial stage, there are two service groups S_i and S_j formed as shown in Figure 2 and all the subtrees are binary. Let the group S_i have a privilege to access the video stream at resolution 0 with full quality, we have $\Omega_i = \{ck_{00}, ck_{01}, ck_{02}\}$. Let the group S_j have a privilege to access only the lowest quality video stream, quality 0, thus we have $\Omega_j = \{ck_{00}, ck_{10}\}$.

Formation of a new service group. User u_m joins with a privilege to access the video stream at resolution 0 with middle quality. The KDC forms a new service group S_m such that $\Omega_m = \{ck_{00}, ck_{01}\}$. Only Step 4 in the rekey algorithm is necessary to complete the key update. The KDC updates ck_{00} and ck_{01} through a one-way function and increases the *revision* fields of them. Participants in S_i and S_j will update ck_{00}, ck_{01} accordingly when they see the data packet indicating the increase of the revision numbers. The rekey message size in this example is 0.

Switch service groups. Suppose that a user u_i switches from S_i to S_j, the KDC splits the leaf node of u_j to accommodate u_i. After u_i leaves, u_{i+1} moves up and occupies the node which was previously associated with ki_{21}. Let kp_i denote the private key of user u_i. Firstly, the KDC generates ki'_{11} and srk'_i and distributes them through the rekey messages $\{ki'_{11}\}_{kp_{i+1}}$, $\{ki'_{11}\}_{ki_{20}}$, $\{srk'_i\}_{ki_{10}}$ and $\{srk'_i\}_{ki'_{11}}$. Secondly, the KDC generates a secret ck_s and updates ck_{01},ck_{02} $(\Omega_i \cap \overline{\Omega_j} = \{ck_{01}, ck_{02}\})$ through a keyed one-way function so that $ck'_{01} = H_{ck_s}(ck_{01})$ and $ck'_{02} = H_{ck_s}(ck_{02})$. The KDC increases the *version* field of ck_{01}, ck_{02} and distributes the secret ck_s through rekey messages $\{ck_s\}_{srk_i}$, $\{ck_s\}_{srk_m}$. Thirdly, the KDC updates KEKs kj_{21}, kj_{11}, srk_j through a one-way function $H(\cdot)$ and increases their *revision* field. All the existing participants possessing these keys update them accordingly. Then the KDC generates a new KEK kj_{31} for u_j and u_i. It distributes kj_{31} to u_j through the rekey message $\{kj_{31}\}_{kp_j}$ and to u_i through the rekey message $\{kj_{31}, kj'_{21}, kj'_{11}, srk_j\}_{kp_i}$. Finally, the KDC updates ck_{10} $(\overline{\Omega_i} \cap \Omega_j = \{ck_{10}\})$ through a one-way function and increases its *revision* field. The rekey message size in this example is 8.

3 Security Analysis

Backward Security Step 1 and Step 2 provide backward security for traffic data protected by $\Omega_i \cap \overline{\Omega_j}$. Step 1 updates the set of KEKs previously possessed by u_i. This update prevents u_i from decrypting ck_s successfully in Step 2. This results u_i being unable to get the updated version of $\Omega_i \cap \overline{\Omega_j}$ and achieves backward security for traffic data protected by $\Omega_i \cap \overline{\Omega_j}$.

Forward Security Step 3 and Step 4 provide forward security in S_j and for traffic data protected by $\overline{\Omega_i} \cap \Omega_j$. Step 3 updates the set of KEKs in S_j which will be possessed by u_i. This update prevents u_i from decrypting successfully the previous data traffic in S_j which may contain unit encryption keys belonging to Ω_j. Step 4 further updates keys in $\overline{\Omega_i} \cap \Omega_j$ so that forward security for traffic data protected by $\overline{\Omega_i} \cap \Omega_j$ is achieved.

4 Performance Analysis

Storage Overhead We analyze the storage overhead in terms of the number of keys stored in both the KDC and each participant.

Similar to [18] and other key management schemes [1]-[5], the key tree investigated in this work is maintained as balanced as possible by positioning the joining users on the shortest branches. We use $l_d(n)$ to denote the length of the branches and $k_d(n)$ to denote the number of keys on the key tree when the key tree has degree d and accommodates n users.

As the tree is maintained as balanced as possible, $l_d(n)$ is either L or $L+1$, where $L = \lfloor \log_d n \rfloor$. Particularly, the number of users who are on the branches with length L is $d^L - \lceil \frac{n-d^L}{d-1} \rceil$ and the number of users who are on branches with length $L+1$ is $n - d^L + \lceil \frac{n-d^L}{d-1} \rceil$. Thus the total number of keys on this key tree is given by

$$k_d(n) = n + \frac{d^L - 1}{d-1} + \lceil \frac{n-d^L}{d-1} \rceil. \tag{1}$$

The KDC stores Ω_C, all the KEKs and private keys in all the subtrees. Except the key information, the KDC also stores privilege information which tells how to map between the AK sets and the service groups. This kind of privilege information can be stated explicitly by associating a service group with a privilege table or indirectly by linking keys with service groups in real implementation. Let $n(S_i)$ denote the number of participants in S_i. The number of keys stored in the KDC is then

$$K_{KDC} = \sum_{i=1}^{I} k_d(n(S_i)) + |\Omega_C| = \sum_{i=1}^{I} k_d(n(S_i)) + N. \tag{2}$$

Let $|\Omega|$ denote the number of keys in a key set Ω. A user in S_i stores $|\Omega_i|$ CEKs, $l_d(n(S_i)) - 1$ KEKs and its private key. Therefore, the users' storage overhead is

$$K_{u \in S_i} = l_d(n(S_i)) + |\Omega_i|. \tag{3}$$

If we assume each service group contains the same number of users, denoted by $n(S_i) = n_0$, thus $n = I \cdot n_0$. Using (2), the KDC's storage overhead is calculated as

$$K_{KDC} = I \cdot k_d(n_0)) + N. \tag{4}$$

Using (3), a user's storage overhead is given by

$$K_{u \in S_i} = l_d(n_0) + |\Omega_i|. \tag{5}$$

As N is usually fixed throughout the communication process and $|\Omega_i|$ is fixed for S_i, we evaluate the storage overhead in the KDC side asymptotically as the number of group members goes up. From (1) we know that $\lim_{n\to\infty} k_d(n) = \frac{d}{d-1}n$. Therefore, from (4) and (5) we have

$$K_{KDC} \sim O(\frac{d}{d-1} I \cdot n_0) \text{ or } O(\frac{d}{d-1} n). \tag{6}$$

$$K_{u \in S_i} \sim O(\log_d n_0). \tag{7}$$

From (6) we see that the storage overhead in the KDC is linear to the total number of participants in the communication process. It is reasonable that a relatively powerful machine is chosen as the KDC. From (7) we know the storage overhead in each participant is logarithmic to the number of participants in the same service group.

Rekey Overhead According to the rekeying algorithm stated in Section 2.3, with similar assumptions stated in Section 4, the number of rekey messages sent out by the KDC is bounded by

$$M_{KDC} \leq dw_i - 1 + u + 2 = d \cdot l_d(n(S_i)) + u + 1. \tag{8}$$

which includes up to $dw_i - 1$ messages generated in Step 1, u messages generated in Step 2 and up to 2 messages in Step 3. The equality holds when both the subtrees of S_i and S_j are full-loaded so that both $\log_d(n(S_i))$ and $\log_d(n(S_j))$ are integers. As $u \leq I$ and in average I changes less frequently than $n(S_i)$, it is reasonable for us to evaluate (8) in the condition of increasing $n(S_i)$ only. Thus when $n_0 \to \infty$, we can see that

$$M_{KDC} \sim O(d \cdot \log_d(n(S_i))). \tag{9}$$

This shows that the number of rekey messages is logarithmic to the size of the service group S_i which the user switches from and not related to the size of the service group S_j which the user switches to. The number of rekey messages can be reduced in those special cases listed in Section 2.3.

5 Conclusion

This paper presented a dynamic access control scheme for group communications with multi-privileged service groups. The proposed scheme uses a key management graph extended from traditional management tree to maintain and manage keys. Each key is associated with a *version* field and a *revision* field. Both fields are used to eliminate or reduce the number of rekeying messages. The proposed

scheme allows users to join/leave group communications and switch access levels. It scales well when new service group forms and achieves forward and backward security when users roam among service groups. The storage overhead required by this scheme in the KDC side is linear to the total number of participants in the communication process. The storage overheard for each participant is logarithmic to the number of participants in the same service group. The number rekeying messages is logarithmic to the number of participants in the service group from which the user switches.

References

1. D.M. Wallner, E.J. Harder, and R.C. Agee, "Key management for multicast: issues and architectures," Internet Draft Report, Sept. 1998, Filename:draft-wallner-key-arch-01.txt.
2. M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner, "The VersaKey framework: versatile group key management," *IEEE Journal on selected areas in communications*, vol. 17, no. 9, pp. 1614-1631, Sep. 1999.
3. R. Canetti, J. Garay, G. Itkis, D. Miccianancio, M. Naor, and B. Pinkas, "Multicast security: a taxonomy and some efficient constructions," in *Proc. IEEE INFOCOMM'99*, vol. 2, pp. 708-716, March 1999.
4. M.J. Moyer, J.R. Rao, and P. Rohatgi, "A survey of security issues in multicast communications," *IEEE Network*, vol.13, no. 6, pp. 12-23, Nov.-Dec. 1999.
5. C. Wong, M. Gouda, and S. Lam, "Secure group communications using key graphs," *IEEE/ACM Trans. on Networking,* vol. 8, pp. 16-30, Feb. 2000.
6. W. Trappe, J. Song, R. Poovendran, and K.J.R. Liu, "Key districution for secure multimedia multicasts via data embedding," *Proc. IEEE ICASSP'01*, pp. 1449-1452, May 2001.
7. S. Mittra, "Iolus: a framework for scalable secure multicasting," in *Proc. ACM SIGCOMM'97*, 1997, pp. 277-288.
8. A. Perrig, D. Song, and D. Tygar, "ELK, a new protocol for efficient large-group key distribution," in *Proc. IEEE Synmposium on Security and Privacy,* 2001, pp. 247-262.
9. S. Banerjee and B. Bhattacharjee, "Scalable secure group communication over IP multicast," *JSAC Special Issue on Network Support for Group Communication*, vol. 20, no. 8, pp/ 1511-1527, Oct. 2002.
10. S.G. Akl and P.D. Taylor, "Cryptographic solution to a problem of access control in a hierarchy," *ACM Transactions on Computer Systems, 1(3)*, pp. 239-248, 1983.
11. S.J. MacKinnon, P.D. Taylor, H. Meijer and S.G. Akl, "An optimal algorithm for assigning cryptographic keys to access control in a hierarchy," *IEEE Transactions on Computers,* C-34(9), pp. 797-802, 1985.
12. R. S. Sandhu, "Cryptographic implementation of a tree hierarchy for access control," *Information Processing Letters, 27(2)*, pp. 95-98, 1988.
13. G.C. Chick and S.E. Tavares, "Flexible access control with master keys," In G. Brassard, editor, *Advances in Cryptology: Proceedings of Crypto'89*, LNCS 435, pp. 316-322, Springer-Verlag, 1990.
14. L. Harn and H.Y. Lin, "A cryptographic key generation scheme for multi-level data security," *Journal of Computer and Security, 9(6)*, pp. 539-546, 1990.

15. K. Ohta, T. Okamoto and K. Koyama, "Membership authentication for hierarchical multigroup using the extended Fiat-Shamir scheme," In I. B. Damgard, editor, *Advances in Cryptology: Proceedings of Eurpcrypt'90,* LNCS 473, pp. 316-322, Springer-Verlag, 1991.
16. C. Yuan, B. Zhu, M. Su, X. Wang, S. Li and Y. Zhong, "Layered access control for MPEG-4 FGS video," *IEEE Int. Conf. Image Processing 2003*, Sep. 2003
17. Robert H. Deng, Yongdong Wu, Di Ma, "Securing JPEG2000 Code-Streams," *International Workshop on Advanced Developments in Software and Systems Security*, Dec. 2003
18. Yan Sun, K. J. Ray Liu, "Scalable hierarchical access control in secure group communications," *Proc. IEEE INFOCOMM'04*, 2004.