Hierarchical Group Access Control for Secure Multicast Communications

Yan Sun* and K. J. Ray Liu†
* Department of Electrical and Computer Engineering
University of Rhode Island, Kingston, RI 02881
† Department of Electrical and Computer Engineering
University of Maryland, College Park, MD 20742

Abstract— **Many group communications require a security infrastructure that ensures multiple levels of access control for group members. While most existing group key management schemes are designed for single level access control, we present a multi-group key management scheme that achieves hierarchical group access control. Particularly, we design an integrated key graph that maintains keying material for all members with different access privileges. It also incorporates new functionalities that are not present in conventional multicast key management, such as user relocation on the key graph. Analysis is performed to evaluate the storage and communication overhead associated key management. Comprehensive simulations are performed in various application scenarios where users statistical behavior is modelled using a discrete Markov chain. Compared with applying existing key management schemes directly to the hierarchical access control problem, the proposed scheme significantly reduces the overhead associated with key management and achieves better scalability.**

Keywords— **access control, communication system privacy, system design**

I. Introduction

The ubiquity of communication networks facilitates the development of wireless and Internet applications that allow users to communicate and collaborate among themselves. In the future, group-oriented applications, such as video conferences, will be one of the important services that facilitate real-time information exchange among a large number of users [1]. Most group-oriented applications require mechanisms that guarantee information security [2]. Among all security requirements of group communication, *access control* is paramount as it is the first line of defense that prevents unauthorized access to the group communication and protects the value of application data.

Group access control is usually achieved through encrypting the content using an encryption key, known as the session key (SK) that is shared by all legitimate group members. Since the group membership will most likely be dynamic with users' joining and departure, the encryption keys should be updated to prevent the leaving/joining user from accessing the future/prior communications [3]. The issues of establishing and updating the group keys are addressed by *Group Key Management* schemes [3–5]. Encryption and key management together ensure data confidentiality because unauthorized entities do not possess the group key and cannot decrypt group communication.

Existing key management schemes, such as in [4–22], address the access control issues in a single multicast session. They focus on establishing and updating keys with dynamic membership and provide all group members the same level of access privilege. That is, the users who possess the decryption keys have the full access to the content, and the users who do not have the decryption keys cannot interpret the data. In practice, many group applications contain multiple related data streams and have the members with various access privileges. These applications are prevalent in various scenarios.

- Multimedia applications distributing data in a multi-layer coding format [23]. For example, in video broadcast, users with a normal TV receiver can receive the normal format, while others with HDTV receivers can receive the normal format and the extra information needed to achieve HDTV resolution.
- Multicast programs containing several related services, such as weather, news, traffic and stock quote.
- Communications in hierarchically managed organizations, such as military group communications where participants have various access authorization.

Since group members subscribe to different data steams, or possibly multiple of them, it is necessary to develop an access control mechanism that supports multi-level access privilege, which shall be referred to as the *hierarchical group access control*.

Traditional multicast key management schemes are not designed to handle key management issues associated with multiple services occurring concurrently with correlated memberships. Although access control for individual data stream can be managed separately using existing key management schemes, this leads to inefficient use of keys and does not scale well when the number of data streams increases, as we will demonstrate in the later sections.

In this work, we develop a *multi-group key management* scheme that addresses the generalized hierarchical group access control problem. Particularly, we design an *integrated key graph* that maintains keying material for all members with different access privileges. It also addresses new functionalities that are not present in conventional multicast key management, such as user relocation on the key graph. The proposed multi-group key management scheme achieves forward and backward secrecy [20] when users (1) join the group communication with certain ac-

[1]This work is supported in part by the Army Research Office under Award No. DAAD19-01-1-0494.
[2]Earlier version of this work was presented in IEEE INFOCOM, Hong Kong, March 2004

cess privilege; (2) leave the group; and (3) add or drop the subscription to one or several data streams (change access privilege). The idea of the integrated key graph can be used in both centralized and contributory environments. This paper will focus on the centralized multi-group key management scheme and then discuss its extension in the distributed scenarios. Compared with using single-session access control solutions, such as a variety of tree-based key management schemes [6, 20], the proposed scheme reduces the usage of the communication, computation and storage overhead, and is scalable when the number of access levels increases.

The rest of the paper is organized as follows. Section II discusses related work. The hierarchical group access control problem is formulated in Section III. The centralized multi-group key management is presented in Section IV-VI. Section IV describes the integrated key graph and the rekey algorithm. Section V analyzes the performance of the proposed scheme. Section VI provides the simulation results. The contributory key management scheme using the integrated key graph is discussed in Section VII, followed by the conclusion in Section VIII.

II. Related work

Popular key management protocols can be divided into two categories: centralized and contributory [9]. The centralized key management relies on a centralized server, referred to as the key distribution center (KDC), which generates and distributes encryption keys. In the contributory key management, there is no explicit KDC, and group members contribute independent keying materials and all participate in the process of group key establishment. Besides particular key management protocols, decentralized key management architectures are also developed. In decentralized architecture, such as [12], the group is divided into subgroups and the task of key management is divided among subgroup managers.

In the current literature, key management schemes are designed for a single multicast session, where all group members have the same access privilege. For the group applications containing multiple related data streams and members with various access privileges, directly applying the existing schemes can lead to inefficient solutions. In the rest of the section, we briefly review some representative key management schemes. Among them, tree-based schemes will serve as a basic building block in developing the proposed hierarchical access control solution.

An important class of centralized key management protocols employ logical tree structures to maintain keying materials and coordinate key generation [4–10]. This type of protocols are considered to be scalable in terms of communication, computation and storage overhead. In [4], the logical key hierarchy (LKH) is introduced. The KDC maintains a key tree, where each node on the tree corresponds to a user's privacy key, the group key, or a key-encrypted-key (KEK). This work achieves scalable rekeying, which requires $O(2 \cdot \log(n))$ rekeying overhead for user joining and departure, where n is the group size. Later, an algorithm proposed in [6] improves the user joining operation such that new keys can be calculated through a one-way function without sending rekeying messages. Another improvement is the one-way function tree (OFT) proposed in [24]. In this approach, the keys on the key tree are generated through one-way functions, rather than arbitrarily determined by the KDC. This approach reduces the rekeying overhead from $O(2 \cdot \log(n))$ to $O(\log(n))$. Later, a slightly different approach that achieves the same communication overhead is proposed in [9]. Instead of using one-way functions, the ELK protocol, presented in [10], uses pseudo-random functions to build and manipulate the keys on key tree. ELK also introduces *hints*, a small piece of information that improves reliability of rekeying.

In many scenarios, it is not preferred to rely on a centralized server that arbitrates the establishment of the group key. This might occur in applications where group members do not explicitly trust a single entity, or there are no servers or group members who have sufficient resources to maintain, generate and distribute keying information. Thus, the distributed solutions of the key management problem have seen considerable attention [6, 14–22]. Many contributory schemes are inspired by the Diffie-Hellman (DH) key exchange protocol [25]. To extend two-party DH protocol to the group scenario, the schemes presented in [20–22] use logical tree structures such that the number of rounds for the formation of the group key is reduced to the logarithm of the group size.

Because of their scalability, the tree-based schemes are selected as the basic building blocks to address the hierarchical group access control problem in both centralized and contributory environments.

III. Hierarchical Group Access Control

There are many specific group applications containing multiple data streams and users with different access privileges. In order to develop a generic solution, we formulate the hierarchical group access control [26] problem for group communication in this section.

A. System description

Let $R = \{r_1, r_2, \cdots\}$ denote the set of *resources* in the system. From the resource points of view, a *Data Group* (DG) is defined as all users who have access to a particular resource. It is clear that the DGs may have overlapped membership because users may subscribe to multiple resources. From access control points of view, a *Service Groups* (SG) is defined as a set of users who can access the exactly same set of resources. SGs do not have overlapped membership. In this section, the DGs are denoted by $\{D_1, D_2, \cdots, D_M\}$, where M is the total number resources, and the SGs are denoted by $\{S_1, S_2, \cdots, S_I\}$, where I is the total number SGs. It is easy to prove that $I \leq 2^M - 1$.

The access relationship between the resources and the SGs can be described by a *capability list*. Here are two examples illustrating typical access relationship in group communication.

Example 1. Multimedia applications distributing data in multi-layer format [23].
• *Resources*: {base layer (r_1), enhancement layer 1 (r_2), enhancement layer 2 (r_3)}.
• *Service Groups*: {users subscribing basic quality (S_1), users subscribing moderate quality (S_2), users subscribing high quality (S_3)}.
• *Capability lists*: S_1 access $\{r_1\}$; S_2 access $\{r_1, r_2\}$; S_3 access $\{r_1, r_2, r_3\}$.
Example 2. Multicast programs containing several related services.
• *Resources*: {News (r_1), Stock quote (r_2), Traffic/Weather (r_3)}.
• *Service Groups*: Users can subscribe to any combination of the resources. Thus, there are total 7 SGs, denoted by $S_1, S_2, \cdots, S_7$.
• *Capability lists*: S_1 access $\{r_1\}$; S_2 access $\{r_2\}$; S_3 access $\{r_3\}$; S_4 access $\{r_1, r_2\}$; S_5 access $\{r_1, r_3\}$; S_6 access $\{r_2, r_3\}$; S_7 access $\{r_1, r_2, r_3\}$.

Besides the capability list, *access matrix* is also used to describe access relationship. In particular, the element on the i^{th} row and m^{th} column of the access matrix, denoted by $a_{i,m}$, is

$$a_{i,m} = \begin{cases} 1, & \text{if SG } S_i \text{ can access resource } r_m \\ 0, & \text{otherwise} \end{cases},$$

where $i = 1, \cdots, I$ and $m = 1, \cdots, M$.

Based on those definitions, the group size of SGs and DGs must satisfy:

$$n(D_m) = \sum_{i=1}^{I} a_{i,m} \cdot n(S_i), \tag{1}$$

where $n(S_i)$ is the number of users in SG S_i and $n(D_m)$ is the number of users in DG D_m.

B. Security requirements

In the applications containing multiple multicast sessions, users not only join or leave service, as addressed in the single multicast session scenario, but also may switch between the SGs by subscribing or dropping data streams. Thus, the security requirements are more complicated than these for a single multicast session.

We introduce the notation $S_i \rightarrow S_j$ that represents a user switching from the SG S_i to the SG S_j. To simplify future notations, S_0 is defined as a virtual service group containing users who cannot access any resources. Thus, $S_0 \rightarrow S_i$ represents a user joining the SG S_i, and $S_i \rightarrow S_0$ represents a user leaving the group communication from the SG S_i.

Similar to the single session access control problem addressed by traditional key management schemes [3], the hierarchical group access control should guarantee the following security requirements.
• The users in the SG S_i have and only have access to the resources $\{r_m, \forall\ m : a_{i,m} = 1\}$.
• When a user $S_i \rightarrow S_j$,
– This user cannot access the future content of the resources $\{r_m, \forall\ m : a_{i,m} = 1 \text{ and } a_{j,m} = 0\}$. This property can be referred to as the forward secrecy [20].
– This user cannot access the previous content of the resources $\{r_m, \forall\ m : a_{i,m} = 0 \text{ and } a_{j,m} = 1\}$. This property can be referred to as the backward secrecy [20].

C. Data encryption and hierarchical key management

To formulate the hierarchical access problem, the ways of encrypting multiple data streams need to be clarified first. In the hierarchical access control scenario, there are two ways to encrypt and distribute multicast data. In the first method, resources are encrypted using separate keys, which are called *Data Group Keys*. The data group key used to encrypt resource r_m, denoted by K_m^D, is shared among the users in DG D_m. In this case, each resource is distributed in a single multicast session, and the users may subscribe to one or several multicast sessions according to their access privilege. The task of key management is to securely update and distribute $\{K_m^D,\ \forall m : a_{i,m} = 1\}$ to the users in S_i, where $i = 1, 2, \cdots, I$.

In the second method, the users in each SG share a secrete key called the *Service Group Key* and the multicast sessions are formed based on SGs. In particular, the users in S_i share the service group key K_i^S and form one multicast session. In this multicast session, the resources $\{r_m, \forall m : a_{i,m} = 1\}$ are encrypted by K_i^S and transmitted to the users in S_i. In this case, one resource may be distributed in several multicast sessions while being encrypted by different service group keys. The task of key management is to securely distribute and update K_i^S for the users in SG S_i.

We compare these two methods using Example 1 in section III-A.

In the first method, data are transmitted in three multicast sessions. The first session contains all users, and distributes resource r_1 encrypted by K_1^D. The second session contains users in S_2 and S_3, and distributes resource r_2 encrypted by K_2^D. The third session contains users in S_3, and distribute resource r_3 encrypted by K_3^D. The communication overhead of a multicast session can be described as $DR(r_i)G(x)$, where $DR(r_i)$ denotes the data rate of resource r_i, and $G(x)$ is the cost of sending unit data to x users through multicast. The first method has communication overhead as $CommCost_{method_1} = DR(r1)G(n(S_1) + n(S_2) + n(S_3)) + DR(r2)G(n(S_2) + n(S_3)) + DR(r3)G(n(S_3))$.

When using the second method, there are three multicast sessions also. The first session contains users in S_1, and distributes resource r_1 encrypted by K_1^S. The second session contains users in S_2, and distributes resource r_1 and r_2 encrypted by K_2^S. The third session contains users in S_3, and distribute all three resources encrypted by K_3^S. The communication overhead is $CommCost_{method_2} = DR(r1)G(n(S_1)) + (DR(r1) + DR(r2))G(n(S_2)) + (DR(r1) + DR(r2) + DR(r3))G(n(S_3))$. Using the fact the $G(x + y) < G(x) + G(y)$ in multicast communications, it can be seen that $CommCost_{method_2} >$

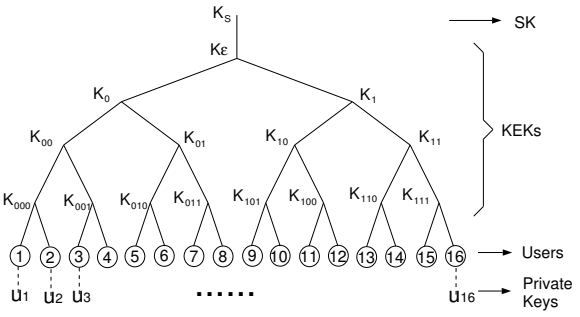


Fig. 1. A typical key management tree

$CommCost_{method_1}$.

On the other hand, users in the second method only subscribe to one multicast session. Thus, the task of key management for the second method can be solved by applying traditional key management for each SG separately.

In this work, we adopt the first encryption method because of its low data communication overhead. In order to guarantee forward and backward secrecy, when a user switches from SG S_i to S_j, the proposed key management scheme should

- update $\{K_m^D, \forall\ m : a_{i,m} = 0 \text{ and } a_{j,m} = 1\}$, such that this user cannot access the previous communication in corresponding DGs;
- and update $\{K_m^D, \forall\ m : a_{i,m} = 1 \text{ and } a_{j,m} = 0\}$, such that this user cannot access the future communication in corresponding DGs.

The focus of this work is to solve this hierarchical group key management problem efficiently.

IV. Centralized Multi-group Key Management Scheme

Hierarchical group access control can be achieved in either centralized or contributory manner. While the contributory solution will be discussed in Section VII, this section and the following two sections will be dedicated to the centralized schemes.

A. Employing independent key trees to achieve hierarchical access control

One way to solve the hierarchical access control problem is to use the existing tree-based key management schemes. Those schemes use a logical tree structure to maintain keying materials. The key tree is also a convenient way to represent the ownership of the keys.

As illustrated in Figure 1, each node of the key tree is associated with a key. The root of the key tree is associated with the session key (SK), K_s, which is used to encrypt the multicast content. Each leaf node is associated with a user's private key, u_i, which is only known by this user and the KDC. The intermediate nodes are associated with key-encrypted-keys (KEK), which are auxiliary keys and only for the purpose of protecting the session key and other KEKs. To make concise presentation, we do not distinguish the node and the key associated with this node in the remainder of the paper.

The key tree represents the ownership of the keys. The KDC knows all keys on the key tree. Each user knows his private key, the session key, and a set of KEKs on the path from himself to the root of the key tree. In the example shown in Figure 1, user 16 possesses $\{u_{16}, K_s, K_\epsilon, K_1, K_{11}, K_{111}\}$.

Each key contains the secrete material that is the content of the key and a *key selector* that is used to distinguish the key. The key selector consists of: 1) a unique ID that stays the same even if the key content changes and 2) a version and revision field, reflecting update in the keying material. The version number is increased whenever new keying material is sent out by the group manager upon user departure, while the revision number is increased whenever the key is passed through a one-way function. The usage of the version and revision numbers will be explained in the description of the key updating process.

When a user leaves the service, all his keys need to be updated in order to prevent him from accessing the future communication. Here we use the scheme presented in [6] to demonstrate the key updating process. When user 16 leaves, the KDC generates new keys and conveys new keys to the remaining users through a set of rekeying messages as: $\{K_{111}^{new}\}_{u_{15}}$, $\{K_{11}^{new}\}_{K_{111}^{new}}$, $\{K_{11}^{new}\}_{K_{110}^{old}}$, $\{K_1^{new}\}_{K_{11}^{new}}$, $\{K_1^{new}\}_{K_{10}^{old}}$, $\{K_\varepsilon^{new}\}_{K_1^{new}}$, $\{K_\epsilon^{new}\}_{K_0^{old}}$, $\{K_s^{new}\}_{K_\epsilon^{new}}$. Here, the notation x^{old} represents the old version of key x, x^{new} represents the new version of key x, and $\{y\}_x$ represents the key y encrypted by key x. The version numbers are increased for all new keys. This key updating procedure guarantees that all remaining users obtain the new session key and KEKs, while user 16 is unable to acquire the new keys. Since the rekeying messages are transmitted in the multicast channel [4], every user receives all rekeying messages. The session key, KEKs and users' private keys usually have the same length. The communication overhead associated with key updating can be described by *rekeying message size*, defined as the amounts of rekeying messages measured in the unit as the same size as the SK or KEKs. In this example, the rekeying message size is 8 when user 16 leaves the service.

When a user joins the service, the KDC chooses a leaf position on the key tree to put the joining user. The KDC updates the keys along the path from the new leaf to the root by generating the new keys from the old keys using a one-way function and increasing the revision numbers of the new keys. The joining user obtains the new keys through the unicast channel. Other users in the group 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 one-way function. No additional rekeying messages are necessary.

When using tree-based schemes to achieve hierarchical group access control, a separate key tree must be constructed for each DG, with the root being the data group key and the leaves being the users in this DG. This approach is referred to as the *Independent-tree* key management scheme. This scheme does not exploit the relationship among the subscribers and makes inefficient use of keys be-

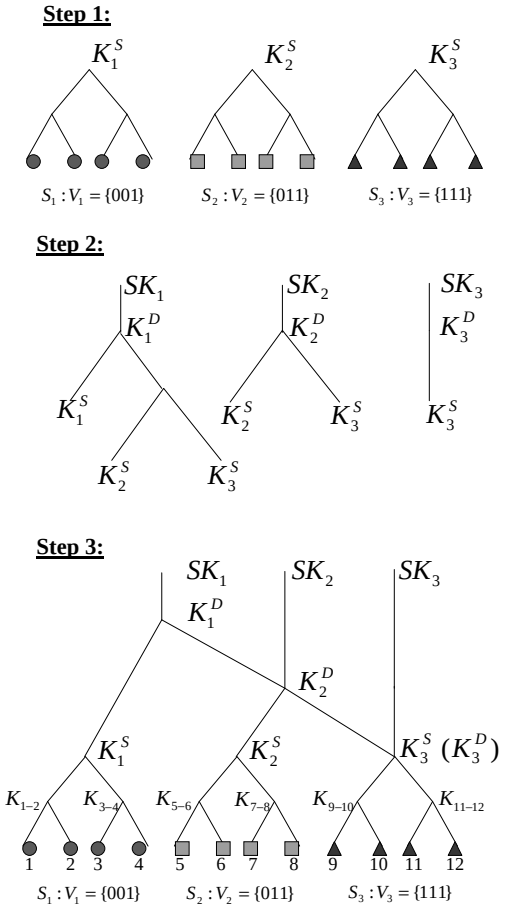


Fig. 2. Multi-group key management graph construction

cause of the overlapped DG membership. As an extreme example, if a user who subscribes to all data streams leaves the service, key updating has to take place on all key trees.

B. Multi-group key management scheme

We propose a *multi-group* key management scheme that employs one integrated key graph accommodating key materials of all users. This key graph consists of several subtrees, and is constructed in three steps.

Step1: For each SG S_i, construct a subtree having the leaf nodes as the private keys of users in S_i and the root node as the service group key K_i^S. These subtrees are referred to as the *SG-subtrees*.

Step2: For each DG D_m, construct a subtree whose root is the DG key K_m^D and whose leaves are $\{K_i^S, \forall i : a_{i,m} = 1\}$. These subtrees are referred to as the *DG-subtrees*.

Step3: Generate the key graph by connecting the leaves of the DG-subtrees and roots of SG-subtrees.

This 3-step procedure is formally described in Procedure 1 and illustrated in Figure 2 for the service containing 3 layers and 4 users in each SG. In the first and the second step, there is no constraint on the tree structures that can be used for the SG- and DG-subtrees. Binary subtrees are used in this paper to demonstrate the performance of the Multi-group key management, while there is a lot of flexibility in subtree design. For example, when considering the heterogeneity among SGs, the DG-subtrees can

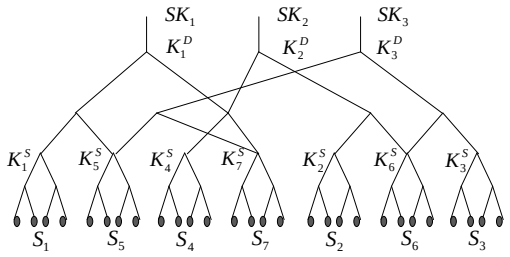


Fig. 3. An integrated key graph for multiple service scenario (Example 2, Section II-A)

be designed as unbalanced trees similar to those proposed in [22]. In the third step, some duplicated structures may appear on DG-subtrees. In Figure 2, the DG-subtrees of D_2 and D_1 have the same structures that connect K_2^S and K_3^S. Duplicated structures can be merged. In this example, the parent node of K_2^S and K_3^S on DG-subtree of D_2 are merged with K_2^D. This merging operation can further reduce the number of keys on the key graph, but the effect of merging is very small especially when the group size is large. Therefore, the performance analysis in Section V does not consider the positive effect of merging, and thus provides the performance upper bound.

This multi-group key graph can also be interpreted as M overlapped key trees, each of which has K_m^D as the root and the users in DG D_m as the leaves. Obviously, these M key trees can be used in the independent-tree scheme. This reveals the fact that the multi-group key graph removes the "redundancy" presented in the independent-tree scheme. Therefore, it can reduce overhead.

As another example, Figure 3 shows a multi-group key graph for the multiple service scenario described in Example 2 in Section III. It is noted that neither the design of the DG-subtrees nor the merging operation is unique. Although the graph between the DG keys and the SG keys can be optimized to minimize the number of keys on the graph, this optimization introduces little gain but high computational complexity. Therefore, the proposed scheme does not specify how to merge the DG-subtrees. As we will show in the performance analysis, the proposed scheme can significantly reduce key management overhead even without removing redundance on the DG-subtrees.

Procedure 1 Integrate Key Graph Generation

for $i = 1 : I$ **do**
  Construct a tree, called SG-subtree of S_i, with $n(S_i)$ leaf nodes.
  Assign users in S_i to leaf nodes.
end for
for $m = 1 : M$ **do**
  Construct a tree, called DG-subtree of D_m, with $\sum_i a_{i,m}$ leaf nodes.
  Assign key $\{K_i^S, \forall i : a_{i,m} = 1\}$ to leaf nodes, and K_m^D to the root node.
end for
for $i = 1 : I$ **do**
  Search all leaf nodes of DG subtrees and find these are associated with K_i^S
  Merge these nodes and the root of the SG-subtree of S_i into one node.
end for

As defined in [4], *keyset* refers to the set of keys associated with an edge node on the key graph and possessed by the user located at this edge node. In our key graph, the keyset of a user in SG S_i is the keys on the pathes from himself to the roots of the DG-subtrees of D_m for $\{m : a_{i,m} = 1\}$. It is noted that the keyset of users in S_0 is just an empty set.

Besides user join and departure, the rekey algorithm in the multi-group key management scheme must address users' relocation on the key graph. We describe the rekey algorithm for $S_i \rightarrow S_j$, which includes the cases for user join, departure, and switching. First, the switching user is moved from the SG-subtree of S_i to a new location on the SG-subtree of S_j. Let ϕ_i denote the keyset associated with the user's previous position, and ϕ_j denote the keyset associated with the user's new position. Then,

- the KDC updates the keys in $\overline{\phi_i} \cap \phi_j$ using one-way functions, similar to the procedure for user join described in Section IV-A,
- and, the KDC generates new versions of the keys in $\phi_i \cap \overline{\phi_j}$ and distributes these new keys encrypted by their children node keys from bottom to up, similar to the procedure for user departure described in Section IV-A.

We illustrate the rekey algorithm of the multi-group key management scheme based on the sample key tree shown in Figure 4. Let user 8 switches from SG S_2 to S_1. The key tree is updated as shown in Figure 4. On the SG-subtree of S_1, the leaf node associated with user 4 is split to accommodate user 8. Then, user 4 and 8 will share a new KEK, denoted by K_{4-8}. On the SG-subtree of S_2, user 7 will be moved up and occupy the node that is previously associated with K_{7-8}. In this case, ϕ_2 is $\{K_{7-8}, K_2^S, K_2^D, SK_2, K_1^D, SK_1\}$ and ϕ_1 is $\{K_{4-8}, K_{3-4}, K_1^S, K_1^D, SK_1\}$.

The KDC generates the new keys, K_{3-4}^{new} and $K_1^{S,new}$, from the old keys using a one-way function, and increases the revision numbers of those new keys. Thus, the user 1,2,3,4 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. No rekeying messages are necessary for K_{3-4}^{new} and $K_1^{S,new}$.

Then, the KDC generates new keys, K_{4-8}^{new}, $K_2^{S,new}$, $K_2^{D,new}$, and SK_2^{new}, and distributes them through a set of rekeying messages as: $\{K_{4-8}^{new}\}_{u_8}$, $\{K_{4-8}^{new}\}_{u_4}$, $\{K_2^{S,new}\}_{K_{5-6}}$, $\{K_2^{S,new}\}_{u_7}$, $\{K_2^{D,new}\}_{K_2^{S,new}}$, $\{K_2^{D,new}\}_{K_3^S}$, $\{SK_2^{new}\}_{K_2^{D,new}}$. In this case, the rekeying message size is 7.

It is noted that $\overline{\phi_i} \cap \phi_j$ may contain the new KEKs that are created for accommodating the switching user. These new KEKs are encrypted by users' private keys and distributed through rekeying messages. In addition, $\phi_i \cap \overline{\phi_j}$ may contain KEKs that do not exist any more after the relocation of the switching user. Obviously, these keys are discarded.

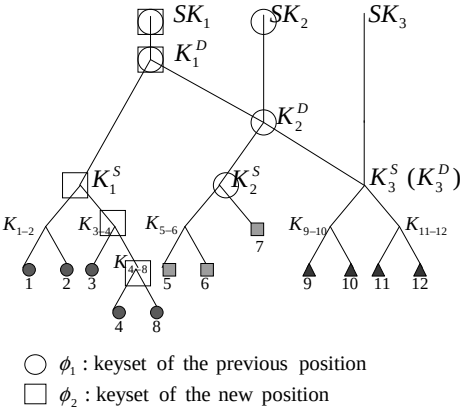


Fig. 4. User relocation on the key graph

V. Performance Measures and Analysis

Communication, computation and storage overhead associated with key updating are major performance criteria for key management schemes [3–5]. To measure the performance of hierarchical access control schemes, we define the performance measures as:

- *Storage overhead at the KDC*: denoted by R_{KDC} and defined as the expected number of keys stored at the KDC.
- *Rekey overhead at the KDC*: denoted by M_{KDC} and defined as the expected amount of rekeying messages transmitted by the KDC.
- *Storage overhead of users*: denoted by $R_{u \in S_i}$ and defined as the expected number of keys stored by the users in SG S_i.
- *Rekey overhead of users*: denoted by $M_{u \in S_i}$ and defined as the expected amount of rekeying messages received by the users in SG S_i.

Here, R_{KDC} and $R_{u \in S_i}$ describe the storage overhead, while M_{KDC} and $M_{u \in S_i}$ reflect the usage of communication and computation resources. For example, given the group size and network topology, M_{KDC} is proportional to the total amount of key management data forwarded on the network.

A. Storage overhead

We first consider the storage overhead of a single key tree. Similar to most key management schemes [3–6, 9], the key tree investigated in this work is fully loaded and maintained as balanced as possible by putting the joining users on the shortest branches.

Let $f_d(n)$ denote the length of the branches and $r_d(n)$ denote the total number of keys on the key tree when the key tree has degree d and accommodates n users. Since the key tree is balanced, $f_d(n)$ is either L_0 or $L_0 + 1$, where $L_0 = \lfloor \log_d n \rfloor$. Particularly,

- the number of users who are on the branches with length L_0 is $d^{L_0} - \lceil \frac{n - d^{L_0}}{d-1} \rceil$,
- and, the number of users who are on the branches with length $L_0 + 1$ is $n - d^{L_0} + \lceil \frac{n - d^{L_0}}{d-1} \rceil$.

Thus, the total number of keys on this key tree is calculated as: $r_d(n) = n + 1 + \frac{d^{L_0} - 1}{d-1} + \lceil \frac{n - d^{L_0}}{d-1} \rceil$. Using the fact that

$\frac{n-d^{L_0}}{d-1} \leq \lceil \frac{n-d^{L_0}}{d-1} \rceil < \frac{n-d^{L_0}}{d-1} + 1$, we have

$$\frac{dE[n]-1}{d-1} + 1 \leq E[r_d(n)] < \frac{dE[n]-1}{d-1} + 2, \quad (2)$$

where the expectation, $E[.]$, is taken over the distribution of $n(D_m)$ and the length of the branches on the key trees. The left-hand-side equality is achieved when $\log_d(n)$ is an integer. In addition, since $\log_d(n)$ is a concave function and $\lfloor \log_d n \rfloor \leq \log_d n$, we see that

$$E[f_d(n)] \leq E[\log_d n] + 1 \leq \log_d E[n] + 1. \quad (3)$$

With (2) and (3), we are ready to analyze the storage overhead. When using the separate key trees, the KDC stores all keys on total M key trees, and users in S_i store subsets of keys on the key trees that are associated with D_m, for $\{m : a_{i,m} = 1\}$. Thus,

$$R^{ind}_{KDC} = \sum_{m=1}^{M} E\left[r_d(n(D_m))\right], \quad (4)$$

$$R^{ind}_{u \in S_i} = \sum_{m=1}^{M} a_{i,m} \left(E[f_d(n(D_m))] + 1\right). \quad (5)$$

In the multi-group key management scheme, the DG-subtree of D_m has $c_m = \sum_i a_{i,m}$ leaf nodes. Before removing the redundancy on DG-subtrees, there are in total $\sum_{m=1}^{M} r_d(c_m)$ keys on DG-subtrees. Also, the total number of keys on the SG-subtrees is $\sum_{i=1}^{I} r_d(n(S_i))$. Merging duplicated structures on DG-subtrees only reduces the number of keys on the key graph. Therefore, the storage overhead at the KDC is

$$R^{mg}_{KDC} \leq \sum_{i=1}^{I} E[r_d(n(S_i))] + \sum_{m=1}^{M} E\left[r_d(c_m)\right]. \quad (6)$$

A user in SG S_i stores $f_d(n(S_i))$ keys on the SG-subtree and up to $\sum_{m=1}^{M} a_{i,m}(f_d(c_m)+1)$ keys on the DG-subtrees. Therefore, the users' storage overhead of the multi-group scheme is:

$$R^{mg}_{u \in S_i} \leq E[f_d(n(S_i))] + \sum_{m=1}^{M} a_{i,m}(E[f_d(c_m)] + 1). \quad (7)$$

We next investigate the storage overhead of the independent-tree and the multi-group key management in the applications containing multiple layers, as described in Example 1 in Section III-A. In this case, $a_{i,m} = 1$ for $m \leq i$ and $a_{i,m} = 0$ for $m > i$. We also assume that each layer contains the same amount of users, denoted by $n(S_i) = n_0$. Thus, $n(D_m) = (M - m + 1)n_0$. Using (5) and (7), the users' storage overhead is calculated as:

$$R^{ind}_{u \in S_i} = \sum_{m=1}^{i} \left(E[f_d((M-m+1) \cdot n_0)] + 1\right), \quad (8)$$

$$R^{mg}_{u \in S_i} \leq E[f_d(n_0)] + \sum_{m=1}^{i} \left(E[f_d(M-m+1)] + 1\right). \quad (9)$$

When the group size is large, i.e. $n_0 \to \infty$, (3)(8) and (9) lead to

$$R^{ind}_{u \in S_i} \sim O(i \cdot \log(n_0)),\ R^{mg}_{u \in S_i} \sim O(\log(n_0)). \quad (10)$$

Using (4) and (6), the storage overhead at the KDC is calculated as:

$$R^{ind}_{KDC} = \sum_{m=1}^{M} E[r_d(m \cdot n_0)], \quad (11)$$

$$R^{mg}_{KDC} \leq M \cdot E[r_d(n_0)] + \sum_{m=1}^{M} E[r_d(m)]. \quad (12)$$

From (2), it is seen that $\lim_{n \to \infty} r_d(n) = \frac{d}{d-1} n$. Thus,

$$R^{ind}_{KDC} \sim O(\frac{d}{d-1} \frac{M(M+1)}{2} n_0);$$
$$R^{mg}_{KDC} \sim O(\frac{d}{d-1} M \cdot n_0). \quad (13)$$

By using the integrated key graph instead of the separate key trees, the multi-group key management scheme reduces the storage overhead of both the KDC and the users. As indicated in (13), the storage advantage of the proposed scheme becomes larger when the system contains more SGs, i.e. requiring more levels of access control. The proposed scheme in fact scales better when the number of layers (M) increases.

B. Rekey overhead

In this section, we calculate the amount of rekeying messages transmitted by the KDC when one user switches from S_i to S_j, denoted by $C_{i,j}$. It is noted that the rekey overhead, M_{KDC} and $M_{u \in S_i}$, can be calculated from $C_{i,j}$, as long as the users' statistical joining/leaving/switching model is given.

Switching from S_i to S_j is equivalent to adding the subscription to $\{D_m, \forall m : a_{i,m} = 0 \text{ and } a_{j,m} = 1\}$ and dropping the subscription to $\{D_m, \forall m : a_{i,m} = 1 \text{ and } a_{j,m} = 0\}$. When using the tree-based key management schemes, the rekeying message size is:

$$C^{ind}_{ij} = \sum_{m=1}^{M} \max(a_{i,m} - a_{j,m}, 0) \cdot (d \cdot f_d(n(D_m))). \quad (14)$$

We can see that the term $(\max(a_{i,m} - a_{j,m}, 0))$ equals to 1 only when $a_{i,m} = 1$ and $a_{j,m} = 0$. When this term equals to 1, $d \cdot f_d(n(D_m)$ rekeying messages are necessary to update keys on the key tree associated with the DG D_m.

In the multi-group key management scheme, when a user switches from S_i to S_j and $i \neq j$, the amount of messages needed to update keys on the SG-subtree of S_i is up to $(d \cdot f_d(n(S_i)) - 1)$. The amount of messages needed to convey the KEK created for accommodating the switching/join user on the SG-subtree of S_j is no more than 2. If this user drops the subscription of the DG D_m, i.e. $(\max(a_{i,m} - a_{j,m}, 0)) = 1$, the amount of rekeying messages that update

keys on the DG-subtree of D_m is up to $(d \cdot f_d(c_m)+1)$. If this user remains the subscription of the DG D_m, i.e. $a_{i,m} = a_{j,m} = 1$, we need up to $(d \cdot f_d(c_m))$ rekeying messages to update keys on the DG-subtree of D_m. Therefore, when using the multi-group scheme and $i \neq j$, we have

$$C_{ij}^{mg} \leq \sum_{m=1}^{M} (\max(a_{i,m} - a_{j,m}, 0) \cdot (d \cdot f_d(c_m) + 1) + a_{i,m} a_{j,m} d \cdot f_d(c_m)) + d \cdot f_d(n(S_i)) + 1. \quad (15)$$

Similar to that in Section V-A, we analyze the rekey overhead in a multi-layer scenario with $n(S_i) = n_0$. In this case, the rekeying message size for one user departure, i.e. $S_j \rightarrow S_0$, is computed from (14) and (15) as:

$$C_{0j}^{ind} = \sum_{m=1}^{j} d \cdot E[f_d((M-m+1)n_0)], \quad (16)$$

$$C_{0j}^{mg} \leq d \cdot E[f_d(n_0)] + 1 + \sum_{m=1}^{j} (d \cdot E[f_d(M-m+1)] + 1). \quad (17)$$

When $n_0 \rightarrow \infty$, we can see that

$$C_{0j}^{ind} \sim O(j \cdot d \cdot \log(n_0)),\ C_{0j}^{mg} \sim O(d \cdot \log(n_0)). \quad (18)$$

The comprehensive comparison between the proposed scheme and the independent-tree scheme will be presented in Section VI.

VI. Simulations and Performance Comparison

A. Statistical dynamic membership model

Before performing simulations, it is important to model the dynamic behavior of group users.

In [27] [28], it has been shown that the users' arrival process and membership duration of MBone multicast sessions can be modelled by Poisson and exponential distribution respectively, in a short period of time. In this work, we use this Poisson arrival and exponential distribution duration model, and assume that when a user switches between SGs, the SG that he switches to depends only on his current SG.

Therefore, the users' statistical behavior can be described by an embedded Markov chain [29]. Particularly, there are a total of $I+1$ states, denoted by $\tilde{S}_i,\ i = 0, \cdots, I$. When a user is in the SG S_i, he is in the state $\tilde{S}_i$. After a user enters state $\tilde{S}_i$, i.e. subscribes or switches to SG S_i, this user stays at state $\tilde{S}_i$ for time T_i, which is governed by an exponential random variable. When time is up, the user moves to state $\tilde{S}_j$. The selection of $\tilde{S}_j$ only depends on the current state $\tilde{S}_i$ and is not related to previous states.

In practice, it is usually not necessary to update keys immediately after membership changes. Many applications allow the join/departure users receive limited previous/future communications [30]. For example, in video streaming applications, a joining user may receive a complete group-of-picture (GOP) [23] although partial of this GOP already been transmitted before his subscription. Those situations prefer *batch rekeying* [30] that postpones key updating such that the rekeying overhead is reduced by adding or removing several users altogether.

In this work, batch rekeying is implemented as updating keys periodically. The time between key updates is fixed and denoted by B_t. For the users who join/leave/switch SGs in the time interval $((k-1)B_t, kB_t]$, the key updating will take place at time kB_t, where k is a positive integer. From the key updating points of view, with batch rekeying, we can prove that the previous continuous Markov model can be simplified as a discrete Markov chain model [29]. In this model,

- The transition matrix is denoted by $P = [p_{ij}]_{(I+1)\times(I+1)}$, where p_{ij} is the probability that one user moves from SG S_i to S_j in the time interval $(kB_t, (k+1)B_t]$ given that this user is in S_i at time kB_t.
- The n-step transition probability matrix is denoted by $P(n)$, and obviously, $P(n) = P^N$. The element at the i^{th} row and j^{th} column of $P(n)$ is denoted by $p_{ij}(n)$.
- The stationary state probability is a 1-by-$(I+1)$ vector, denoted by $\pi = [\pi_0, \pi_1, \cdots, \pi_I]$.

In practical group applications, users can subscribe to (or leave from) any SGs and the expected time for a user staying in the group communication is finite. Thus, we can prove that this Markov chain is irreducible, aperiodic and positive recurrent. As a result, the stationary state probability mass function (pmf) exists [29] and is the unique solution of $\pi P = \pi$ and $\sum_i \pi_i = 1$.

B. Simulation Flow

A simulator based on C is built to study the performance of the multi-group scheme and the independent-tree scheme. The simulation is performed according to the following procedure.

1. Initialization determines the number of data groups, the number of service groups, access relationship, and the transition matrix. Then, the stationary state probability π is calculated based on P. The initial group size is set as $N_0\pi$ for SG S_i, where N_0 is the total number of potential users.
2. Construct the integrated key graph using Procedure 1.
3. Simulate user joining/depature/switching according to the transition matrix, and obtain storage and rekey overhead. One round of key updating is performed in one time unit. In each round,

– According to the rekeying protocol proposed in Section IV-B, update the key graph structure by removing departure users, adding joining users and relocating switching users on the key tree. Meanwhile, mark the intermediate keys that need to be updated. In particular, if a key needs to be updated due to user departure or switching out of a SG, it is marked as 'L'. If a key needs to be updated due to user joining or switching into a SG, it is marked as 'J'. For the same key, a 'L' mark overwrites a 'J' mark.

– Update the keys with 'J' marks using one-way function. Create rekeying messages according to 'L' marks, and update corresponding keys.

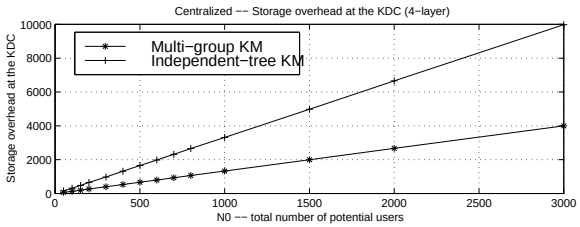


Fig. 5. Storage overhead at the KDC

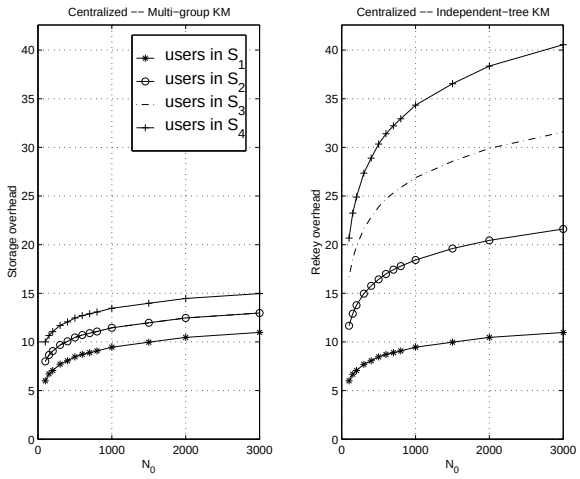


Fig. 6. Storage overhead at the users in each SG

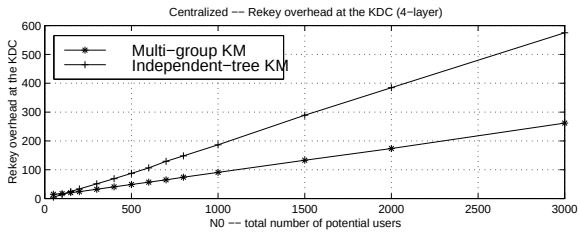


Fig. 7. Rekey overhead at the KDC

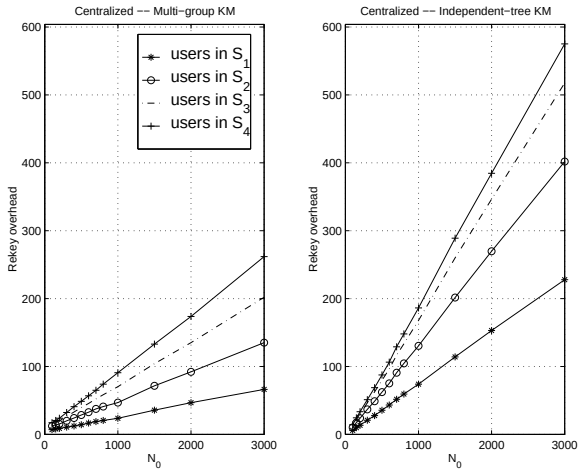


Fig. 8. Rekey overhead at the users in each SG

C. Performance with different group size

We first study the applications containing multiple layers (see Example 1 in SectionIII-A) where users in SG S_i can access DG $D_1, D_2, \cdots, D_i$. In the simulation, the transition matrix is chosen as follows.

- Users join different SGs with the same probability, i.e. $P_{0j} = \alpha, \forall j > 0$.
- Users leave different SGs with the same probability, i.e. $P_{i0} = \beta, \forall i > 0$.
- While a user is in the service, he adds/drops only one DG at a time, i.e. $P_{i,j} = 0, \forall i, j > 0$ and $|i - j| > 1$. Also, users switch between SGs with the same probability, i.e. $P_{i,j} = \gamma, \forall i, j > 0$ and $|i - j| = 1$.

Thus, the transition matrix is described by only three variables. In all simulations, batch rekeying is applied and the key trees are binary.

In Figure 5, 6, 7 and 8, the multi-group scheme and the independent-tree scheme are compared. The horizontal axis is the total number of potential users, N_0. The vertical axis is the storage overhead or rekeying overhead defined in Section V. The results are averaged over 300 realizations, and the number of layers is 4. In those simulations, we chose $\alpha = 0.005$, $\beta = 0.01$, and $\gamma = 0.001$.

Figure 5 shows that the number of keys stored at the KDC, R_{KDC}, increases linearly with the group size. This result can be verified by (2)(4) and (6). In the case when $M = 4$, the multi-group scheme reduces R_{KDC} by more than 50%. For example, when the group size is 3000 and the key size 128 bits, the independent-tree scheme requires about 160Kbyte secure storage at KDC, while the multi-group scheme requires about 64Kbyte.

Figure 6 shows that the users' storage overhead, $R_{u \in S_i}$, increases linearly with the logarithm of the group size. This can be verified by (8) and (9). The users who subscribe only to one layer have the similar storage overhead in both schemes. For the uses who subscribe to multiple layers, the multi-group scheme results in less storage overhead than the independent-tree scheme. For example, when the group size is 3000, a layer-4 user needs to store 656 byte keys in the independent-tree scheme and to store 240 byte keys in the multi-group scheme.

Since secrete keys need to be stored in a secure storage space, the reduction in key storage is desirable. More importantly, when the secure storage space is limited, the multi-group scheme can support more users.

The KDC's rekey overhead, R_{KDC} and the users' rekey overhead, $R_{u \in S_i}$ are shown in Figure 7 and 8, respectively. In both cases, the multi-group scheme reduces the rekey overhead by more than 50%.

In some group communications, the rekeying messages only contribute to a small percentage of the overall data rate, as long as the group size is not overwhelmingly large. Thus, reducing rekey overhead has limited advantage of saving bandwidth. The major advantage, however, is to improve reliability.

The rekeying messages must be transmitted reliably and in a timely manner. If a user does not receive keys correctly, he cannot decrypt the content even if he receives the content correctly. Assume there are N users in a multicast group, C rekeying packets need to be transmitted, packet

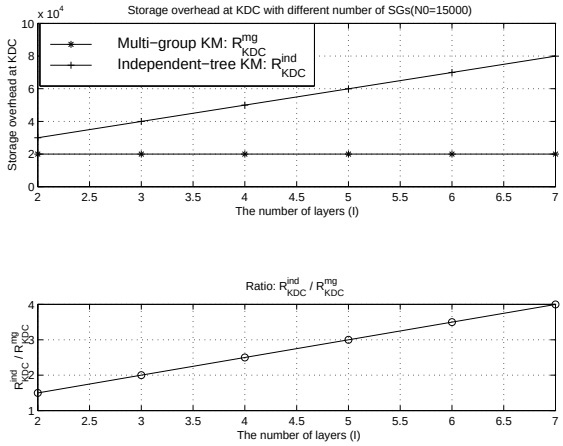


Fig. 9. Storage overhead at the KDC with different number of SGs

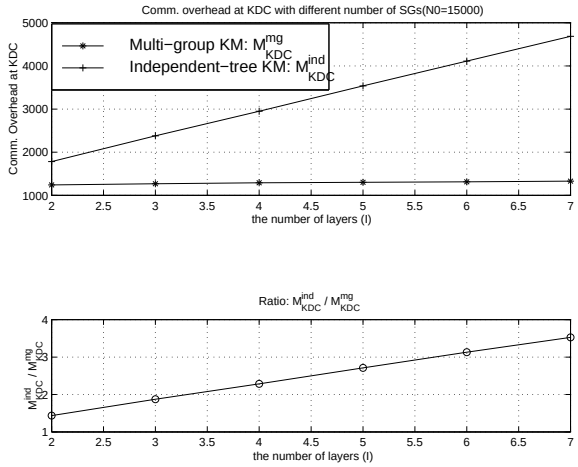


Fig. 10. Rekey overhead at the KDC with different number of SGs

loss among different users is independent, and packet loss ratio is p. Then, the reliability of rekeying protocol can be described by the probability that all users receive all rekeying packets correctly. This can be calculated as $(1-p)^{NC}$, which approximately equals to $1-NCp$. Therefore, from the reliability points of view, a 50% reduction in rekeying overhead (i.e. reduction in C) is significant.

D. Scalability

Next, we change the number of layers (M) while maintaining roughly the same number of users in the service by choosing the join probability α as $0.02/M$. The values of β and γ are the same as those in Section VI-C.

Figure 9(a) and Figure 10(a) show the storage and rekey overhead at the KDC, respectively. When M increases, the storage and rekey overhead of the multi-group scheme do not change much, while the overhead of the independent-tree scheme increases linearly with M. It is clear that the multi-group scheme scales better when M increase. By removing the redundancy in DG membership, the scale of the key graph mainly depends on the group size, not the number of layers or services. On the other hand, by constructing M separate key trees, the independent-tree scheme requires larger storage and rekey overhead when M increases even when N_0 is fixed.

Figure 9(b) shows that the ratio between R^{ind}_{KDC} and R^{mg}_{KDC} increases linearly with M, which agrees with (13). Similarly, the ratio between M^{ind}_{KDC} and M^{mg}_{KDC} increases linearly with M, as shown in Figure 10(b).

E. Performance with different transition probability

In the previous simulations, we set $\gamma = 0.1\beta$, which means that the users are more likely to leave the service than to switch SGs. Figure 11 shows the rekey overhead with different values of γ. Remember that γ describes the probability of user switching between SGs. In this simulation, $M = 4$, $N_0 = 1000$, and the values of α and β are the same as those in the previous experiments.

When γ is very small, the multi-group scheme reduces the rekey overhead by about 50%, as we have shown in the

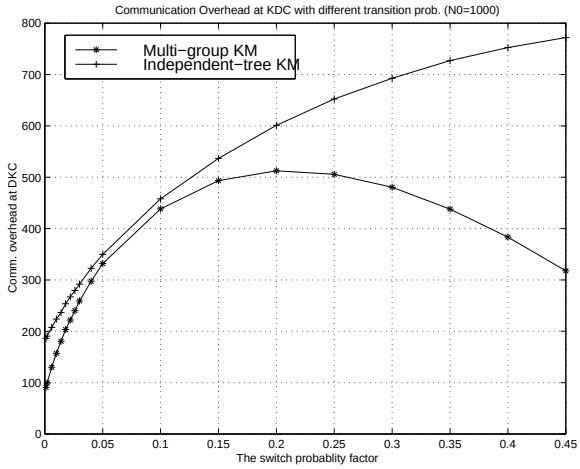


Fig. 11. Rekey overhead at the KDC with different transition probability

previous simulations. When γ is less than 2β, the advantage of the multi-group scheme decreases with the increase of γ. This is because the multi-group scheme introduces larger rekey overhead when users switch SGs by simply subscribing more DGs. To see this, let a user move from SG S_1 to SG S_2. When using the independent-tree scheme, this user only needs to be added to the key tree associated with the DG D_2 and no rekeying messages are necessary. When using the multi-group scheme, we need to update keys on the SG-subtree of S_1 and the DG-subtree of D_1. Therefore, the performance gain reduces when more users tend to switch SGs.

When γ continues to increase, however, the rekey overhead of the multi-group scheme decreases. Particularly, when $\gamma = 0.45$, which describes the scenario where users are much more likely to switch SGs than to stay in the current SG or leave the service, the performance gain of the multi-group scheme is about 50% again. This phenomena is due to the fact that the size of the SG-subtree is greatly reduced when a significant potion of users are switching away from this SG. In this case, removing a large potion of users from the key tree using batch rekeying requires less

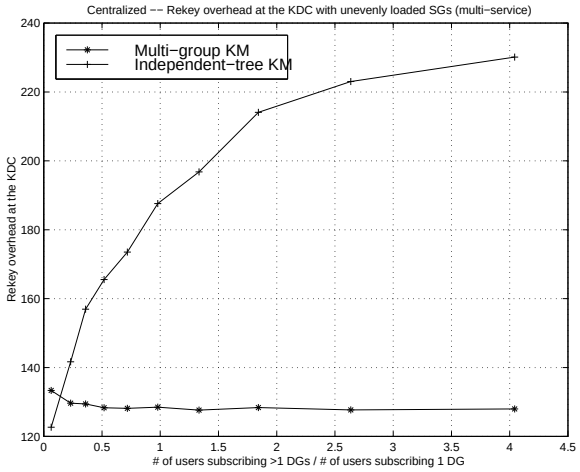


Fig. 12. Rekey overhead at the KDC with unevenly loaded SGs in multi-service applications

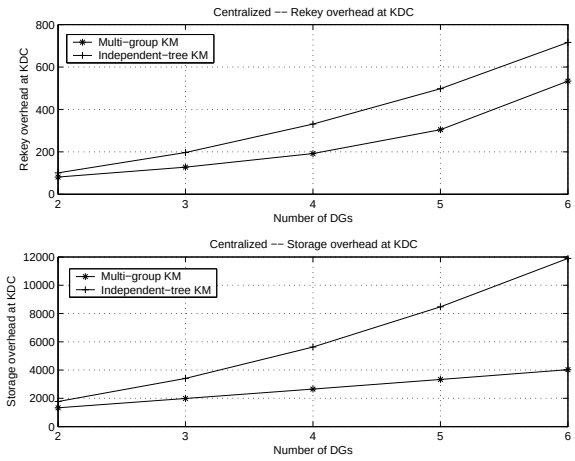


Fig. 13. Rekey overhead and storage overhead at the KDC with the different number of DGs in multi-service applications

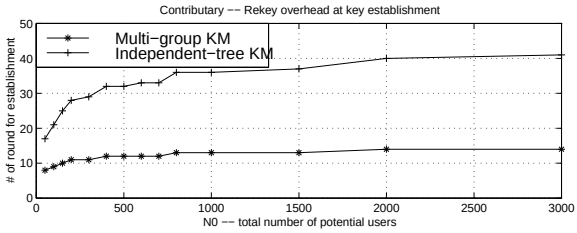


Fig. 14. The total number of rounds performed to establish the group key

rekeying messages than just removing several users.

F. Simulation of multi-service applications

We simulated the multi-service scenario illustrated in Example 2 (Section III-A). In the first experiment, there are 3 DGs and 7 SGs. The users can subscribe to any combination of DGs and switch to any SGs. Here, the transition matrix is 8 by 8, with $P_{j0} = 0.01, \forall j > 0$ and $P_{i,j} = 0.00017, \forall i, j > 0$ and $i \neq j$. N_0 is fixed to be 1500. The values of $P_{0i}, \forall i > 0$, are adjusted such that the SGs contain varying number of users while $(\sum_{i=1}^{I} P_{0i})$ is maintained to be the same. The horizontal axis in Figure 12 is the ratio between the number of users subscribing more than one DGs and the number of users subscribing only one DG. This ratio describes the overlap in DG membership. Larger is this ratio, more overlap is in DG membership. It can be seen that when there is no overlap, the performance of the proposed scheme is slightly worse than that of the independent-tree scheme. This is because additional tree structure connecting the SG subtrees and the DG keys is used in the proposed scheme. When the overlap increases, the overhead of the proposed scheme remains almost the same, but the overhead of the independent-tree scheme increases greatly. Thus, the advantage of the multi-group scheme is larger when more users subscribe to multiple DGs.

In the second experiment, we test different multi-service applications by varying the number of DGs (M). Given M, the number of SGs is $I = 2^M - 1$ and the potential group size is chosen as $500M$. The transition matrix is chosen similar to that in Section VI-C, with $\alpha = 0.02/I$, $\beta = 0.01$, and $\gamma = 0.1\beta$. Figure 13 shows the rekey overhead and the storage overhead at KDC for different M values. Three observations are made. First, the multi-group scheme requires less overhead than the independent-tree scheme. Second, the advantage in storage of the multi-group scheme becomes larger when there are more DGs. Third, if we divide the rekey overhead of the independent-tree scheme by that of the multi-group scheme, we can obtain the relative advantage of the multi-group scheme. This advantage increases with M for $2 \leq M \leq 4$, but decreases for $4 < M \leq 6$. This is because the size of the DG-subtrees is proportional to I, which increases exponentially with M. This interesting observation implies that the proposed multi-group scheme can be further improved by reducing the complexity of the DG-subtrees. This will be investigated in our future work.

VII. Extension to Contributory Hierarchical Access Control

In this section, we first briefly introduce tree-based contributory key management, and then discuss how to extend the multi-group key management to contributory environments.

A. Tree-based contributory key management schemes

The tree-based scheme in [22] is based on applying two-party DH protocol amongst two subgroups of users. In particular, the users in the first subgroup, who share a common subgroup key K_i, send $\{g^{K_i} \ mod \ p\}$ to users in the second subgroup; and the users in the second subgroup, who share a common subgroup key K_j, send $\{g^{K_j} \ mod \ p\}$ to users in the first subgroup. Here, g is the exponential base and p is modular based in the DH protocol [25]. Then, users in two subgroups compute a new key: $K_{ij} = g^{K_i K_j} \ mod \ p$. By doing so, these two subgroups can be merged into a larger subgroup that share the common key K_{ij}.

The key tree used in [20] [22] is similar to that in the

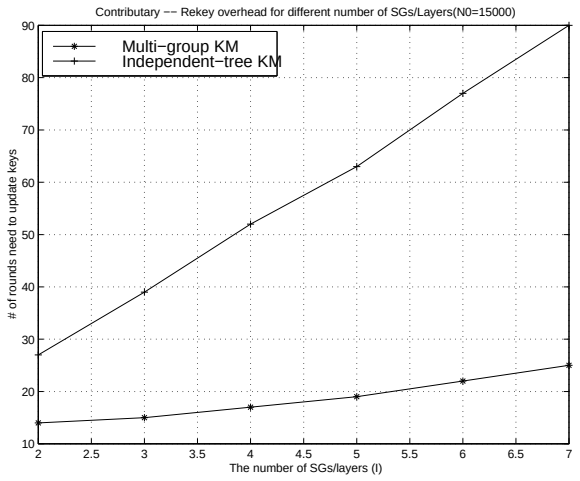


Fig. 15. The number of rounds performed to establish the group key with different number of SGs/layers

centralized schemes, as shown in Figure 1. The intermediate keys and the group key are generated from bottom to up as follows. In the first round, users are grouped into pairs and perform two-party DH. Thus, two users form a subgroup. In each of the following rounds, the subgroups formed in the previous round are paired up and each pair of subgroups perform DH and are merged into a larger subgroup with a shared key. Finally, all users are merged into one group that share the group key K_ϵ. When a user joins or leaves the service, the group key are regenerated in the similar fashion except that some existing intermediate keys do not need to be recalculated [20, 22]. In the example shown in Figure 1, K_ϵ is established in 4 rounds. When user 16 leaves the service, user 15 generates a new private key and 3 rounds should be performed to compute K_{11}^{new}, K_1^{new}, and K_ϵ^{new}.

B. Contributory Multi-group key management scheme

The multi-group key management schemes can be extended to the contributory environment by using the same graph construction procedure presented in Section IV-B. Similar to these in the centralized environments, separate key trees for each DG must be constructed when using existing tree-based contributory schemes [20–22], and the multi-group contributory schemes maintains one integrated key graph for all users.

The key establishment protocols are straightforward extensions from the existing protocols in tree-based contributory schemes [20–22]. When users join/leave/switch, the set of keys that need to be recalculated is the same as that need to be updated presented in Section IV-B. The new keys are recalculated by applying the DH protocol between the users who are under the left child node and the users who are under the right child node from bottom to up.

For contributory key management schemes, the number of rounds is usually used to measure the communication, computation, and latency [31] associated with key establishment and updating [19–21].

With the same simulation setup as that in Section VI-C, the performance of the independent-tree and multi-group contributory key management schemes are compared for varying group size. Figure 14 shows the total number of rounds to establish the group key, which reflects the latency in key establishment [31]. With the same simulation setup as that in Section VI-C, Figure 15 shows the number of rounds for key updating for with different number of layers. Compared with the tree-based contributory schemes, the multi-group contributory scheme significantly reduces the computation and latency associated with key establishment and updating. The advantage of the multi-group contributory scheme is larger when M increases.

VIII. Conclusion

This paper presented a multi-group key management scheme that achieves hierarchical group access control in secure group communications, where multiple data streams are distributed to group members with various access privileges. We designed an integrated key graph, as well as the rekey algorithms, which allow users to change access levels while maintaining the forward and backward secrecy. Compared with using the existing tree-based key management schemes that are designed for a single multicast session, the proposed scheme can greatly reduce the overhead associated with key management. In the multi-layer services containing 4 layers, we observed more than 50% reduction in the usage of storage, computation, and communication resources in the centralized environments, and the number of rounds to establish and update keys in the contributory environments. More important, the proposed scheme scales better than the existing tree-based schemes, when the group applications contain more data streams and require the mechanism to manage more levels of access control.

References

[1] S. Paul, *Multicast on the Internet and its applications*, Kluwer Academic Publishers, 1998.

[2] A. Perrig and J. D. Tygar, *Secure Broadcast Communication: In Wired and Wireless Networks*, Kluwer Academic Publishers, 2002.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] W. Trappe, J. Song, R. Poovendran, and K.J.R. Liu, "Key distribution for secure multimedia multicasts via data embedding," *Proc. IEEE ICASSP'01*, pp. 1449–1452, May 2001.

[8] D. McGrew and A. Sherman, "Key establishment in large dynamic groups using one-way function trees," Technical Report 0755, TIS Labs at Network Associates, Inc., Glenwood, MD, May 1998.

[9] R. Canetti, J. Garay, G. Itkis, D. Miccianancio, M. Naor, and B. Pinkas, "Multicast security: a taxonomy and some efficient constructions," *Proc. IEEE INFOCOM'99*, vol. 2, pp. 708–716, March 1999.

[10] A. Perrig, D. Song, and D. Tygar, "ELK, a new protocol for efficient large-group key distribution," in *Proc. IEEE Symposium on Security and Privacy*, 2001, pp. 247 –262.

[11] G. H. Chiou and W. T. Chen, "Secure broadcasting using the secure lock," *IEEE Trans. Software Eng.*, vol. 15, pp. 929–934, Aug 1989.

[12] S. Mittra, "Iolus: A framework for scalable secure multicasting," in *Proc. ACM SIGCOMM '97*, 1997, pp. 277–288.

[13] 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.

[14] I. Ingemarsson, D. Tang, and C. Wong, "A conference key distribution system," *IEEE Transactions on Information Theory*, vol. 28, pp. 714–720, Sep. 1982.

[15] D. G. Steer, L. Strawczynski, W. Diffie, and M. Wiener, "A secure audio teleconference system," in *Proceedings on Advances in cryptology*. 1990, pp. 520–528, Springer-Verlag New York, Inc.

[16] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution scheme," *Advances in Cryptology- Eurocrypt*, pp. 275–286, 1994.

[17] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-hellman key distribution extended to group communication," in *Proceedings of the 3rd ACM conference on Computer and communications security*. 1996, pp. 31–37, ACM Press.

[18] M. Steiner, G. Tsudik, and M. Waidner, "CLIQUES: a new approach to group key agreement," in *Proceedings of the 18th International Conference on Distributed Computing Systems*, May 1998, pp. 380 –387.

[19] M. Steiner, G. Tsudik, , and M. Waidner, "Key agreement in dynamic peer groups," *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, vol. 11, no. 8, pp. 769–780, Aug 2000.

[20] G. Tsudik Y. Kim, A. Perrig, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *Proceedings of the 7th ACM conference on Computer and communications security*, November 2000.

[21] L.R. Dondeti, S. Mukherjee, and A. Samal, "DISEC: a distributed framework for scalable secure many-to-many communication," in *Proceedings of Fifth IEEE Symposium on Computers and Communications*, 2000, pp. 693 –698.

[22] W. Trappe, Y. Wang, and K.J.R. Liu, "Establishment of conference keys in heterogeneous networks," in *proceedings of IEEE International Conference on Communications*, 2002, vol. 4, pp. 2201 –2205.

[23] A. Puri and T. Chen, *Multimedia Systems, Standards, and Networks*, Marcel Dekker Inc, 2000.

[24] D. Balenson, D. McGrew, and A. Sherman, "Key management for large dynamic groups: one-way function trees and amortized initialization," Internet Draft Report.

[25] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. on Information Theory*, vol. 22, pp. 644–654, 1976.

[26] Y. Sun and K.J.R. Liu, "Scalable hierarchical access control in secure group communications," in *Proc. of IEEE INFOCOM'04*, March 2004.

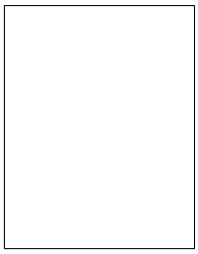
[27] K. Almeroth and M. Ammar, "Collecting and modeling the join/leave behavior of multicast group members in the mbone," in *Proc. High Performance Distributed Computing (HPDC'96), Syracuse, New York*, 1996, pp. 209–216.

[28] K. Almeroth and M. Ammar, "Multicast group behavior in the internet's multicast backbone (MBone)," *IEEE Communications*, vol. 35, pp. 224–229, June 1999.

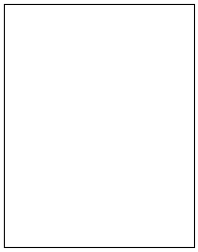
[29] A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, Addison Wesley, 2nd edition, 1994.

[30] Y. R. Yang, X. S. Li, X. B. Zhang, and S. S. Lam, "Reliable group rekeying: a performance analysis," *Proc. of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pp. 27 – 38, August 2001.

[31] B. Sun, W. Trappe, Y. Sun, and K.J.R. Liu, "A time-efficient contributory key agreement scheme for secure group communications," *Proc. of IEEE International Conference on Communication*, vol. 2, pp. 1159 –1163, 2002.



Yan Sun(S00M04) received her B.S. degree with the highest honor from Beijing University, Beijing, China, in 1998, and the Ph.D. degree in electrical and computer engineering from the University of Maryland in 2004. She is currently an NSF ADVANCE assistant professor in the Electrical and Computer Engineering Department, University of Rhode Island. Her research interests include network security and wireless communications and networking. Dr. Sun received the Graduate School Fellowship at the University of Maryland from 1998 to 1999, and the Excellent Graduate Award of Beijing University in 1998. She is a member of the IEEE Signal Processing, Communication, and Computer Societies.



K. J. Ray Liu(F03) received the B.S. degree from the National Taiwan University in 1983 and the Ph.D. degree from the University of California, Los Angeles (UCLA) in 1990, both in electrical engineering. He is a Professor and Associate Chairman and Director of Graduate Studies and Research of the Electrical and Computer Engineering Department, University of Maryland, College Park. His research contributions encompass broad aspects of wireless communications and networking, information forensics and security, multimedia communications and signal processing, bioinformatics and biomedical imaging, and signal processing algorithms and architectures.

Dr. Liu is the recipient of numerous honors and awards, including Best Paper Awards from the IEEE Signal Processing Society, the IEEE Vehicular Technology Society, and EURASIP; the IEEE Signal Processing Society Distinguished Lecturer; the EURASIP Meritorious Service Award; and the National Science Foundation Young Investigator Award. He also received the Poole and Kent Company Senior Faculty Teaching Award from the A. James Clark School of Engineering and the Invention of the Year Award, both from the University of Maryland. He is Vice-PresidentPublications and on the Board of Governors of the IEEE Signal Processing Society. He was Editor-in-Chief of the IEEE SIGNAL PROCESSING MAGAZINE and the founding Editor-in-Chief of the EURASIP Journal on Applied Signal Processing.