A Key Management Scheme for Hierarchical Access Control in Group Communication*

Qiong Zhang[1], Yuke Wang[2], and Jason P. Jue[2]
(Corresponding author: Qiong Zhang)

Advanced Signals Laboratory, School of Engineering Design & Technology[1]
University of Bradford, BD7 1DP, UK
Department of Computer Science, The University of Texas at Dallas[2]
PO Box 830688, EC 31, Richardson, TX 75083-0688
(Email: {ademahmo, S.J.Shepherd}@bradford.ac.uk)

(Received Feb. 1, 2007; revised and accepted Apr. 22, 2007)

Abstract

In group communication, users often have different access rights to multiple data streams. Based on the access relation of users and data streams, users can form partially ordered relations, and data streams can form partially ordered relations. In this paper, we propose a key management scheme for hierarchical access control, which considers both partially ordered user relations and partially ordered data stream relations. We also propose an algorithm for constructing a logical key graph, which is suitable even when users and data streams have complex relations. Simulation results show that our scheme can significantly improve the efficiency of key management.

Keywords: Group communication, hierarchical access control, key management, logical key graph

1 Introduction

Many emerging Internet applications, such as teleconferencing, e-newspaper, and IPTV, are based on group communication, in which a user or a subgroup of users sends (or receives) data streams to (or from) other users. In order to widely commercialize these applications, the issue of access control must be addressed. Access control for group communication must ensure that legitimate users are able to access authorized data streams, while preventing non-legitimate users from gleaning any unauthorized data stream.

Access control for users having different access rights to multiple data streams is referred to as *hierarchical access control*. Some examples of hierarchical access control include e-newspaper subscription and video multicast services. For an e-newspaper subscription service, there may be multiple data streams to send: Top News, Weather, Financial News, Stock, and Sports News. The service provider classifies users into four membership groups: Gold, Silver Sports, Silver Finance, and Basic. The access rights of each membership group to the data streams can be shown in Table 1. In a video multicast service, a multicast video may be encoded into three types of data streams: Base Layer (BL), Enhancement Layer 1 (EL1), and Enhancement Layer 2 (EL2), using a multilayer coding format [11]. Users can subscribe to services with different video qualities. Users that receive the best video quality are able to access all three data streams, users that receive the moderate video quality are able to access the BL and EL1 data streams, and users that receive the basic video quality are able to access only the BL data stream. Table 2 shows the access rights of each user group in the multicast video service. More applications of hierarchical access control can be found in areas such as online banking and military group communications.

To implement access control for group communication, data encryption keys are often used to encrypt data streams. A user is able to access the set of data streams only if the user possesses the data encryption keys that are used to encrypt the corresponding set of data streams. When a user dynamically joins or leaves a group, data encryption keys must be updated to ensure *backward secrecy* and *forward secrecy* [12]. Backward secrecy is a property to prevent a new user from decoding the data exchanged before it joined the group. Forward secrecy is a property to prevent a leaving or expelled user from continuing to access data exchanged after it left the group.

Key management schemes aim to update the data encryption keys in order to ensure backward secrecy and forward secrecy. Key management schemes can be broadly classified to two categories: centralized and distributed. The distributed schemes [1, 4, 5, 6, 13, 14, 15, 18] have no centralized group controller and generate group keys based on the contribution of users in the group. In centralized schemes [2, 7, 8, 9, 10, 16, 21, 22, 23], a centralized

*Portions of this paper have appeared in IEEE GLOBECOM 2004, Dallas, TX, Dec. 2004.

Table 1: E-newspaper subscription service

Access Relation	Sports	Financial	Stock	Top News	Weather
Gold	√	√	√	√	√
Silver Sport	√			√	√
Silver Finance		√	√	√	√
Basic				√	√

key server controls the entire group; it generates keys and distributes keys to legitimate users via *rekey messages.* In this paper, we focus on a centralized key management scheme.

Table 2: Video multicast service

Access Relation	EL2	EL1	BL
Best Video Quality	√	√	√
Moderate Video Quality		√	√
Basic			√

For systems with a large number of users, the efficiency of a centralized key management scheme is a non-trivial issue due to a single point of key management. As in [12, 17], the efficiency of a centralized key management scheme is primarily measured by: 1) *rekey overhead at the key server*, defined as the average number of rekey messages transmitted by the key server to users per key updating. 2) *rekey overhead at users*, defined as the average number of rekey messages received by the users per key updating, 3) *storage overhead at the key server*, defined as the average number of keys stored at the key server, and 4) *storage overhead at the users*, defined as the average number of keys stored at the users. It is critical to minimize rekey overhead in order to reduce the cost for communication and computation at the key server and users. Storage overhead is directly associated with the scalability of key management schemes. For users that are memory constrained, such as cell phones and PDAs, it is desirable to minimize the storage overhead at each user.

Previous key management schemes for hierarchical access control have failed to take into consideration either user relations or data stream relations. In this paper, we propose a centralized *hierarchical access control* (HAC) key management scheme that considers the relations of both users and data streams. The contributions of the paper are:

1) The HAC scheme encrypts equivalent data streams with a single data encryption key based on data stream relations, which simplifies the constructed key graph, thereby improving the efficiency of key management.

2) The paper provides a greedy algorithm for constructing a key subgraph based on the combined relations of users and data streams. The proposed algorithm is suitable for complex access relations. The related studies do not provide such an algorithm.

The rest of this paper is organized as follows. In Section 2, we introduce the background and the related work of our study. In Section 3, we present a key management scheme for constructing a logical key graph based on partially ordered relations of users and data streams. Section 4 describes the rekey algorithm. Section 5 analyzes storage overhead and rekey overhead of the proposed scheme. Section 6 evaluates the performance of our proposed scheme. Section 7 concludes the paper.

2 Background and Related Work

In this section, we introduce the formalization of partially ordered relations of users and data streams. We also present the existing key management schemes for hierarchical access control.

2.1 Formalization of Partially Ordered Relations

In a hierarchical access control system, users may have different access rights to multiple data streams. The work in [3] has shown that users can form a partial order and data streams can also form a partial order based on access relations. The following notation is introduced:

- U: A set of users $\{u_1, u_2, \cdots\}$

  R - a set of data streams $\{r_1, r_2, \cdots\}$, or overall resources.

- A: An access relation, where $A \subseteq U \times R$.

- $R(u_i)$: A set of data streams that user u_i can access.

- $U(r_i)$: A set of users that can access data stream r_i.

- U_i: Membership group i consisting of a subset of users.

- R_i: Resource group i consisting of a subset of data streams.

A partial order of users is defined as a binary relation over a set of users which is reflexive, antisymmetric, and transitive. In an access relation A, if the data streams that user u_i can access is a subset of data streams that user u_j can access, then u_i is *smaller* than u_j (reflexivity property). We have

$$u_i \leq_U u_j \ iff \ R(u_i) \subseteq R(u_j).$$

If users u_i and u_j can access exactly the same subset of data streams, both users are *equivalent.* Then,

$$u_i \equiv_U u_j \ iff \ R(u_i) = R(u_j),$$

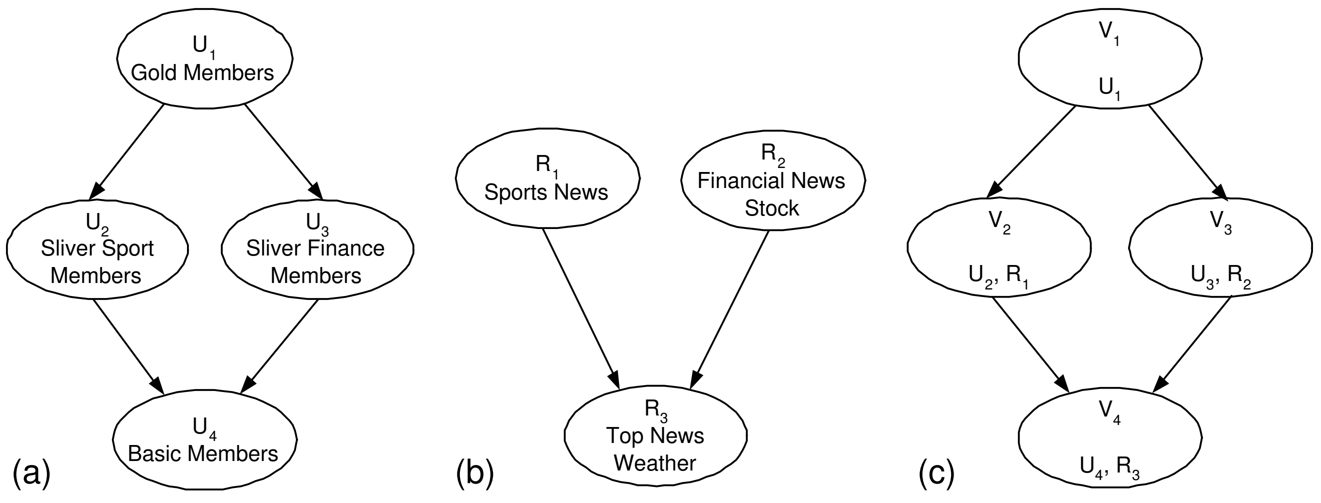

Figure 1: (a)Membership group DAG, (b) resource group DAG, and (c) unified DAG

that is

$$if\ \ u_i \leq_U u_j\ \ and\ \ u_j \leq_U u_i,\ \ then\ \ u_i \equiv_U u_j.$$

Here, the equivalent users u_i and u_j are merged into a *membership group* (antisymmetric property).

If user u_i can access a subset of data streams that user u_j can access, and user u_j can access a subset of data streams that user u_k can access, then u_i is smaller than u_k (transitivity property). We have

$$if\ \ u_i \leq_U u_j\ \ and\ \ u_j \leq_U u_k,\ \ then\ \ u_i \leq u_k.$$

Similarly, based on the access relation A, the data streams also form partially ordered relations. If the set of users that can access data stream r_j is a subset of users that can access data stream r_i, then r_i is smaller than r_j (reflexivity property). We have

$$r_i \leq_R r_j\ \ iff\ \ U(r_j) \subseteq U(r_i).$$

Note that this formalization is reversed compared to formalizing partially ordered relation of users.

If the set of users that can access data stream r_i is exactly same as the set of users that can access data stream r_j, the two data streams are equivalent. The relation of r_i and r_j is:

$$r_i \equiv_R r_j\ \ iff\ \ U(r_i) = U(r_j).$$

The equivalent data streams r_i and r_j will be grouped into a *resource group* (antisymmetric property).

If the set of users that can access data stream r_j is a subset of users that can access data stream r_i, and the set of users that can access data stream r_k is a subset of users that can access data stream r_j, then r_i is smaller than r_k (transitivity property).

$$if\ \ r_i \leq_R r_j\ \ and\ \ r_j \leq_R r_k, then\ \ r_i \leq_R r_k.$$

In [3], the partially ordered relations of membership groups and the partially ordered relation of resource groups can each be represented by a directed acyclic graph (DAG). In the DAG, a vertex represents a membership group or a resource group. We say that Vertex V_j is *adjacent* to Vertex V_i if (V_j, V_i) is an edge in the DAG. Moreover, we say that Vertex V_j is *reachable* to Vertex V_i if and only if there exists a directed path from V_j to V_i.

The connectivity of a DAG is determined as follows. For a DAG that represents the partially ordered relation of membership groups, if $u_i \leq_U u_j$ for $u_i \in U_i$ and $u_j \in U_j$, then the vertex that represents U_j is reachable to the vertex that represents U_i, which means that membership group U_j has more access rights than membership group U_i. For a DAG that represents the partially ordered relation of resource groups, if $r_i \leq_R r_j$ for $r_i \in R_i$ and $r_j \in R_j$, then the vertex that represents R_j is reachable to the vertex that represents R_i, which means that resource group R_j can be accessed by fewer membership groups than resource group R_i. Based on the access relation shown in Table 1, the DAGs that represent the partially ordered relation of membership groups and the partially ordered relation of resource groups are shown in Figure 1 (a) and (b) respectively.

A unified DAG, defined in [3], combines the membership group DAG and the resource group DAG by merging vertices in the two DAGs. The unified DAG must satisfy the following conditions: 1) it must maintain the partial orders of the membership group DAG and the resource group DAG; 2) a user $u \in U$ has access to a resource $r \in R$ if and only if the vertex representing U is the same as the vertex representing R or is reachable to the vertex representing R in the unified DAG; and 3) the unified DAG is the smallest partial order satisfying the above conditions. Figure 1 (c) shows the unified DAG for the access relation shown in Table 1.

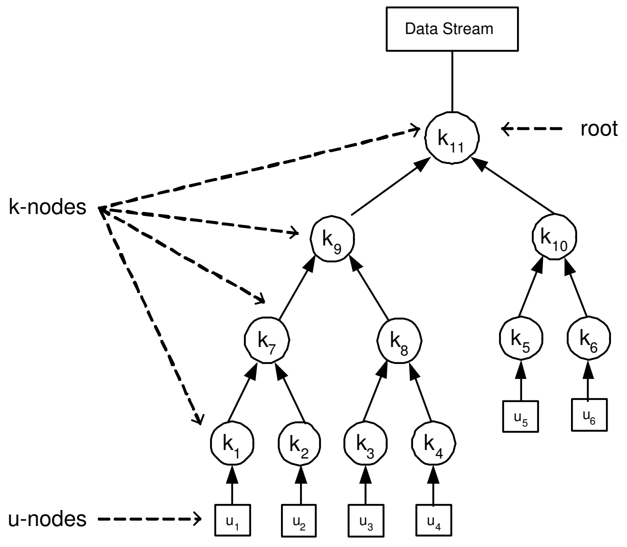

Figure 2: A logical key graph

2.2 Related Work

A logical key graph is an important data structure to improve the efficiency of key management. A logical key graph [22] is a DAG consisting of *k-nodes*, each of which represents a key, and *u-nodes*, each of which represents a user. Figure 2 shows a logical key graph. A u-node has one or more outgoing edges but no incoming edges, while a k-node may have both outgoing edges and incoming edges. A k-node that has no outgoing edges is called a *root*. There are two types of keys represented by k-nodes, data encryption keys and key encryption keys. Data encryption keys are used to encrypt data streams, while key encryption keys are used to encrypt data encryption keys and other key encryption keys. K-nodes that are roots in the logical key graph normally represent data encryption keys. The k-node that represents Key K_{11} is a root in Figure 2. Each user u_i stores a set of keys that is notated as $Keyset(u_i)$. $Keyset(u_i)$ includes all keys represented by the k-nodes along the route from User u_i to the root of the key graph. In Figure 2, $Keyset(u_1) = \{ k_1, k_7, k_9, k_{11}\}$. The set of users that possess Key k is notated as $Userset(k)$. Users in $Userset(k)$ consist of u-nodes that have a route in the key graph to the k-node that represents Key k. In Figure 2, $Userset(k_9) = \{ u_1, u_2, u_3, u_4\}$. Key k is said to *cover* users in $Userset(k)$.

In centralized key management schemes, logical key graphs are maintained at key servers in order to efficiently distribute keys to dynamically joining or leaving users. When a user joins or leaves the group, the key server sends rekey messages to update the keys affected by the joining or leaving user [22]. The number of rekey messages sent by the key server for a user joining or leaving increases linearly with the logarithm of group size. The scheme proposed in [20] employs a different joining operation that requires only one rekey message: when a user joins the group, the key server chooses a leaf position for the user, and all keys along the route from the new leaf to the root are passed through a one-way function. Every user that already knew the old key can calculate the new key locally. Hence, when a user joins, the key server needs to send only one rekey message for a joining user that is used to inform the joining user the updated keys. In this paper, we adopt the second approach for a user joining.

Many key management schemes [2, 7, 8, 10, 16, 22, 23] have been proposed to construct a logical key graph and to update keys in the logical graph efficiently. However, these schemes only provide key management for equivalent users and equivalent data streams in which all users in the group have the same access right to the same set of data streams.

An intuitive key management scheme for hierarchical access control, in which users have different access rights to different data streams, is one in which a separate data encryption key encrypts each data stream. For each data encryption key, the centralized key server must maintain a separate logical key tree containing the users that can access the corresponding data stream. When this scheme is used, a user that is able to access multiple data streams will be contained in multiple logical key trees, which may lead to key management inefficiency.

In order to improve key management efficiency, several key management schemes [17, 19, 24] have been proposed for constructing a single logical key graph for hierarchical access control. In [19], users have different access levels, and higher-level users can access more data streams than lower-level users. Each level is associated with a level key. A single unbalanced logical key tree is constructed, in which higher-level users possess all level keys that are possessed by lower-level users. However, this scheme only provides key management for the case in which users form linearly ordered relation, that is, the higher-level users are able to access all data streams that the lower-level users can access. Other researches [24] have proposed a Chinese Reminder Theorem based hierarchical access control scheme. However, this scheme is only suitable for users having a tree-based partially ordered hierarchy.

The *Multi-Group* (MG) key management scheme considers the case in which users form a partially ordered relation, while the data streams are not partially ordered [17]. In this scheme, each data stream i is encrypted by a data encryption key, dk_i. Users that can access a data stream i form a *data group*, and these users share the data encryption key dk_i. Users that can access the same set of data streams form a *service group*, and share a service group key, sk_i. In order to construct a single logical key graph, the scheme initially constructs a logical key subtree, referred to as a service-group subtree, for every service group. Then, the scheme constructs a logical key subtree, referred to as a data-group subtree, to connect the service-group subtrees to the data encryption keys. The duplicated structures on the data-group subtrees are merged in order to improve key management efficiency. Using the MG scheme, Figure 3 (a) shows the logical key graph constructed for the access relation in Table 1.

In the MG scheme, the algorithm for merging dupli-

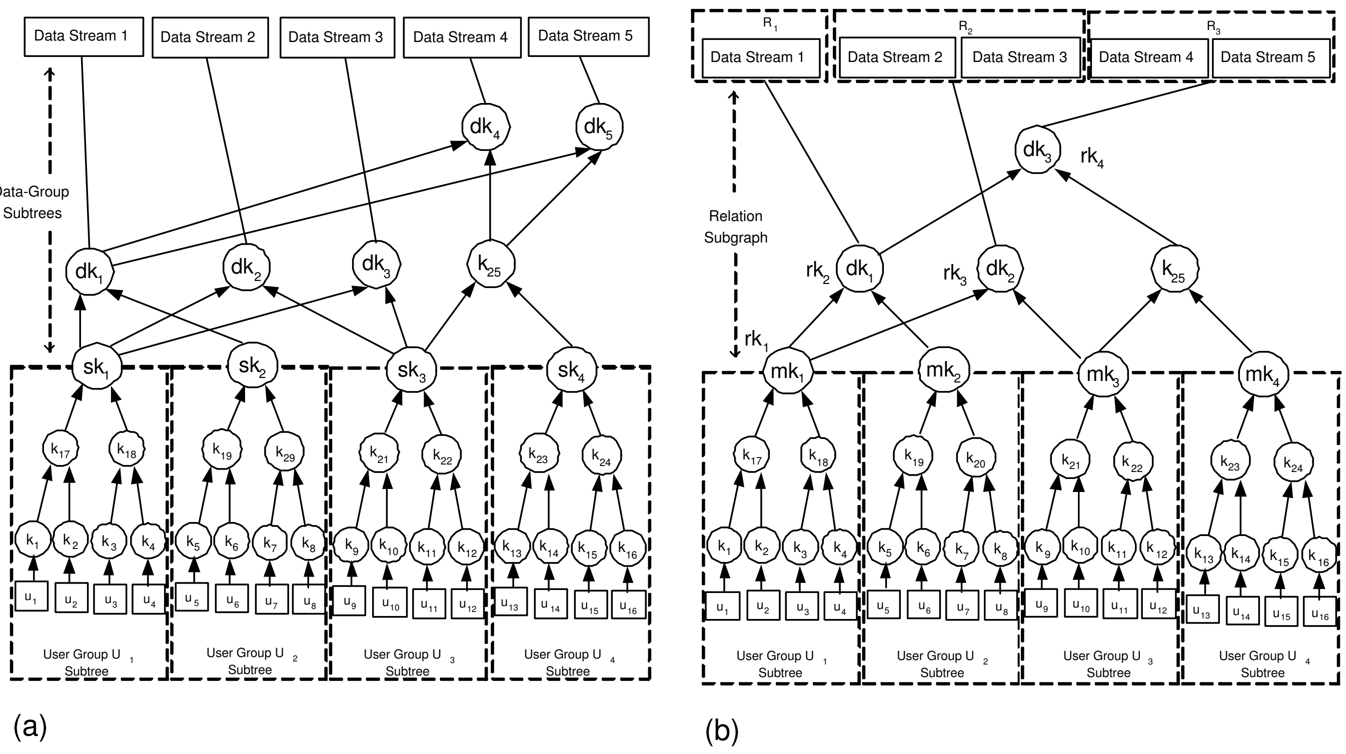

Figure 3: The key graphs constructed by (a) the MG scheme and (b) the HAC scheme

cated structures is not provided, which makes the merging difficult when users have complex partially ordered relations. Furthermore, the MG scheme adopts a flat relation of data streams, that is, each data stream is encrypted by an individual data encryption key, which leads to key redundancy. For example, in Figure 1 (b), Top News and Weather are equivalent data streams and can be grouped into the same resource group; Financial News and Stock are also equivalent data streams and can be grouped into the same resource group. All data streams within a single resource group can share the same data encryption key, which results in fewer data encryption keys. Hence, it is beneficial to consider not only user relation, but also data stream relation.

The HAC scheme that we propose in this paper focuses on constructing a simplified key graph, thereby improving the key management efficiency. The periodic batch rekeying technique [23] can be applied to the HAC scheme in order to further improve the key management efficiency, which is the ongoing research of this study. Similar to the study in [17], the constructed key graph can be used for contributory key management schemes [8, 18] in distributed environments.

3 The HAC Scheme

In this section, we describe our proposed HAC scheme for constructing a logical key graph. There are four steps to construct the logical key graph.

Step 1. For each resource group, encrypt all data streams in the resource group with a single data encryption key, called the *resource-group key.*

Step 2. For each membership group, construct a balanced logical key tree called the *membership-group subtree*, where each user is represented by a u-node and the root of the subtree is associated with a key, called the *membership-group key.*

Step 3. Construct a *relation subgraph* to connect the resources-group keys based on a unified DAG.

Step 4. Connect the roots of membership-group subtrees to the corresponding resource-group keys.

Figure 3 (b) shows the key graph constructed by the HAC scheme based on the unified DAG in Figure 1 (c). We can see that the key graph constructed by the HAC scheme results in a simpler key graph compared to the MG scheme. In the key graph, for each membership group U_i, users in U_i form a balanced binary tree, in which the root represents the membership-group key, mk_i. For each resource group R_i, all data streams in R_i are encrypted by a resource-group key, dk_i. The membership-group keys are connected with resource-group keys by the relation subgraph.

Constructing the relation subgraph is an important component in the HAC scheme, especially for complex access relations. The algorithm for constructing the subgraph is to explore the unified DAG and use a greedy algorithm for *cover*ing all membership groups that are reachable to and equivalent to a resource group. In the

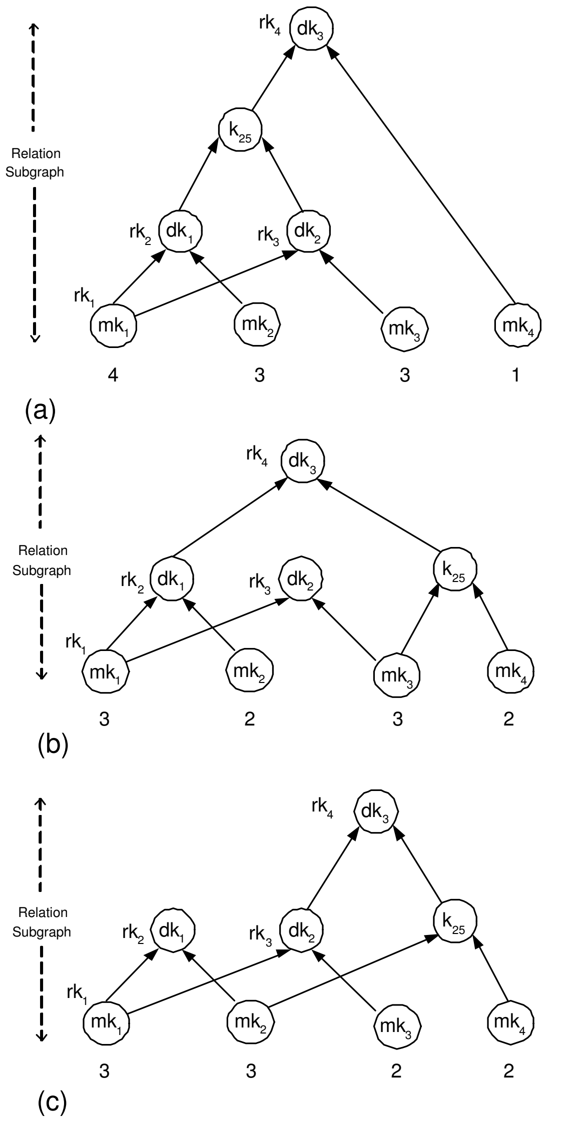

Figure 4: Different cover operations when vertex V_4 is visited

process of exploring the unified DAG, the algorithm colors each vertex white or black. All vertices are colored white initially. A vertex is colored black only if it has been visited and all vertices that are adjacent to it are colored black. The algorithm starts by selecting a vertex that has no incoming edge, and colors the selected vertex black. While the unified DAG is explored in a breadth-first search manner, the algorithm constructs the relation subgraph from bottom to the top. When a vertex V_i is colored black, a cover operation is performed, in which a balanced tree is constructed for the membership groups that are in all vertices reachable to and equivalent to V_i. In the resulting balanced tree, the root is called the *representative k-node* of V_i, notated as rk_i, and the leaves are the membership groups that are in V_i and in all vertices reachable to V_i. If there is only a single membership group in V_i and in all vertices reachable to V_i, then rk_i will be the same as the root of the membership-group subtree. If V_i contains a resource group R_i, K-node rk_i will represent the resource-group key dk_i, so that dk_i covers all membership groups that are in V_i and in all vertices reachable to V_i.

For example, in Figure 1 (c), Vertex V_1 will be visited first and will be colored black since no vertex is adjacent to it. After Vertex V_1 is colored black, the algorithm explores Vertex V_2. V_2 will be colored black since only Vertex V_1 is adjacent to it, and V_1 is colored black. A cover operation will then cover the membership groups in Vertices V_1 and V_2, resulting in a balanced tree with the root rk_1, shown in Figure 3 (b). Similarly, a cover operation will cover the membership groups in Vertices V_1 and V_3 when V_3 is visited, resulting in a balanced tree with the root rk_3.

When the remaining Vertex V_4 is visited and is colored black, the cover operation for Vertex V_4 has to cover the membership groups in Vertices V_1, V_2, V_3, and V_4. One simple approach is to join the roots rk_1 and rk_3 using a new key node in order to cover membership groups in Vertices V_1, V_2, and V_3, and then graft the membership group in Vertex V_4 as shown in Figure 4 (a). The root of the resulting binary tree is the resource-group key for R_3, dk_3. In this cover operation, the number of keys along the routes from mk_1 to the root dk_3 is, $|Keyset(mk_1)| = 4$, and the total number of keys from all mk_i to dk_3 is $\sum_{i=1}^{4} |Keyset(mk_i)| = 11$. Figure 4 (b) and (c) show two binary trees after a cover operation that result in $\sum_{i=1}^{4} |Keyset(mk_i)| = 10$. Note that one less key node in the relation subgraph may result in many membership groups possessing fewer key nodes, thereby improve the efficiency of key management. We also observe that the overlap of $Userset(rk_2)$ and $Userset(rk_3)$ causes higher $\sum_{i=1}^{4} |Keyset(mk_i)|$ in Figure 4 (a).

We provide a greedy algorithm for an optimal cover operation that results in a binary tree with minimum number of k-nodes and with each k-node covering disjoint user sets. In the cover operation, Vertex V_i is visited. Let M be the set of the membership groups that are to be covered, i.e., the set of the membership groups in V_i and in all vertices reachable to V_i. And let K be a set of k-nodes that covers disjoint sets of membership groups, C be a set of membership groups in M that has been covered by the k-nodes in K, and U be the set of the uncovered membership groups in M. Let RK be the set of representative k-nodes that has been generated and $Userset(rk_j)$ be the set of membership groups covered by a representative k-node rk_j. Initially, $K = \{mk_i\}$, $C = \{U_i\}$, and $U = M - C$. In each step, the algorithm selects an existing representative k-node, say rk_j, that contains the membership groups in U and covers most uncovered membership groups in U, notate as $Max(|Userset(rk_j)|)$, where $rk_j \in RK$ and $Userset(rk_j) \subseteq U$. Then remove remove rk_j from RK, $Userset(rk_j)$ from U, insert the k-node rk_j to K, and insert $Userset(rk_j)$ into C. If $M = C$, the algorithm constructs a balanced binary tree for all the k-nodes in K.

We summarize the algorithm for an optimal cover operation when Vertex V_i is visited.

```
Input: $M$ and $RK$
Output: a balanced binary tree for k-nodes in $K$
begin
$K = \{mk_i\}$, $C = \{U_i\}$, and $U = M - C$
while $C \neq M$ {
        Find $Max(|Userset(rk_j)|)$, where $rk_j \in RK$ and $Userset(rk_j) \subseteq U$
        $RK = RK - \{rk_j\}$
        $U = U - Userset(rk_j)$
        $C = C + Userset(rk_j)$
        $K = K + \{rk_j\}$
}
Construct a balanced tree for the k-nodes in $K$
end
```

The algorithm to explore the unified DAG and to construct the relation subgraph is as follows. The input unified DAG is represented by a collection of lists $Outgoing[v_i]$, where v_i is a vertex in the unified DAG. Each $Outgoing[v_i]$ contains the vertices connected from v_i by outgoing edges. The algorithm also maintains several additional data structures with each vertex v_i: a list $incoming[v_i]$ that contains all vertices connected to v_i by incoming edges, an array $color[v_i]$ that stores the color of Vertex v_i, and a First-In First-Out queue Q to manage the set of vertices that will be visited by the algorithm.

```
Input: the unified DAG
Output: the relation subgraph
// Check if all vertices that are adjacent to $v_i$ are colored black
Function bool Check-Black (Vertex $v_i$) {
        for every vertex $v_j \in incoming[v_i]$ {
            if $color[v_j] \neq$ BLACK { return FALSE };
        }
        return TRUE;
}

begin
for every vertex $v_i$ in the unified DAG {
        $color[v_i]$ = WHITE;
        Obtain $incoming[v_i]$ based on adjacency lists;
}
Find S, a set of vertices that has no incoming edge in the unified DAG;
for each $s \in$ S {
        EnQueue($Q$, $s$);
        while $Q \neq \emptyset$ {
            $v$ = head($Q$);
            if Check-Black($v$) {
                $color[v]$ = BLACK;
                Do cover operation for Vertex $v$;
                for each $v_i \in Outgoing[v]$ {
                    EnQueue($Q$, $v_i$);
                }
            }
            DeQueue($Q$);
        }
}
end
```

4 Rekey Algorithm

A rekey process is required to update the keys in the key graph if users join, leave, or switch membership groups dynamically, or if the service provider changes access relations dynamically.

The rekey algorithm for a user joining, leaving, or switching between membership groups is similar to the rekey algorithm in [17]. Let $\theta_{i,k} = Keyset(u_k)$, where User $u_k \in U_i$. If User u_k switches from a position in a membership group U_i to a new position in a membership group U_j, the key server must update the keys in $\theta_{j,k} \cap \overline{\theta_{i,k}}$ using one-way functions [20] and must update keys in $\theta_{i,k} \cap \overline{\theta_{j,k}}$ by distributing rekey messages [20, 22]. If User u_k leaves U_i, then $\theta_{j,k} = \emptyset$. If User u_k joins U_j, then $\theta_{i,k} = \emptyset$.

For example, in Figure 3 (b), User u_8 switches from U_2 to U_1. The key server chooses a position for u_8 that is the sibling of u_1, and creates a new k-node representing Key k_{26} that is shared between u_8 and u_1. Some related key sets are listed as follows:

$$\theta_{1,8} = \{\ k_8,\ k_{26},\ k_{17},\ mk_1,\ dk_1,\ dk_2,\ dk_3\ \},$$
$$\theta_{2,8} = \{\ k_8,\ k_{20},\ mk_2,\ dk_1,\ dk_3\ \},$$
$$\theta_{2,8} \cap \overline{\theta_{1,8}} = \{\ k_{20},\ mk_2\ \}, \text{ and}$$
$$\theta_{1,8} \cap \overline{\theta_{2,8}} = \{\ k_{26},\ k_{17},\ mk_1,\ dk_2\ \}.$$

We notate k_i as the individual key of User u_i. Since k_{26} is a new key, the key server distributes k_{26} via rekey messages $\{k_{26}\}k_8$ (sent to User u_8) and $\{k_{26}\}k_1$ (sent to User u_1). The remaining keys in $\theta_{1,8} \cap \overline{\theta_{2,8}}$ are updated using one-way functions. Since the k-node representing k_{20} no longer exists after User u_8 leaves the group, the key server updates the keys in $\theta_{2,8} \cap \overline{\theta_{1,8}}$ except k_{20} by distributing rekey messages. In this case, the key server only needs to generate a new key mk'_2 and to distribute mk'_2 through rekey messages $\{mk'_2\}k_{19}$ (sent to Users u_5 and u_6 simultaneously) and $\{mk'_2\}k_7$ (sent to User u_7).

The service provider may update the access relation, such as adding a new data stream or a new membership group. Updating access relations may result in reformalization of membership group relation and resource group relation, such as splitting and combining membership groups and resource groups, or adding and removing membership groups and resource groups. Hence, the relation subgraph must be updated. When we update the relation subgraph, we reuse the representative k-nodes to reduce the number of rekey messages required to be sent.

For example, the access relation in Table 1 is changed by adding a new Flight Schedule data stream, and a new Silver Frequent Traveler membership group with four users. Only the Gold and Silver Frequent Traveler membership groups can access the Flight Schedule data stream. The new unified DAG and the new key graph are shown in Figure 5. For the new relation between V_1 and V_5 (V_1 is adjacent to V_5) in Figure 5 (a), Key dk_4, represented by K-node rk_5, must cover U_1 and U_5 as shown in Figure 5 (b). For the new relation between V_4 and V_5 (V_5 is adjacent to V_4) in Figure 5 (a), we reuse the existing Key

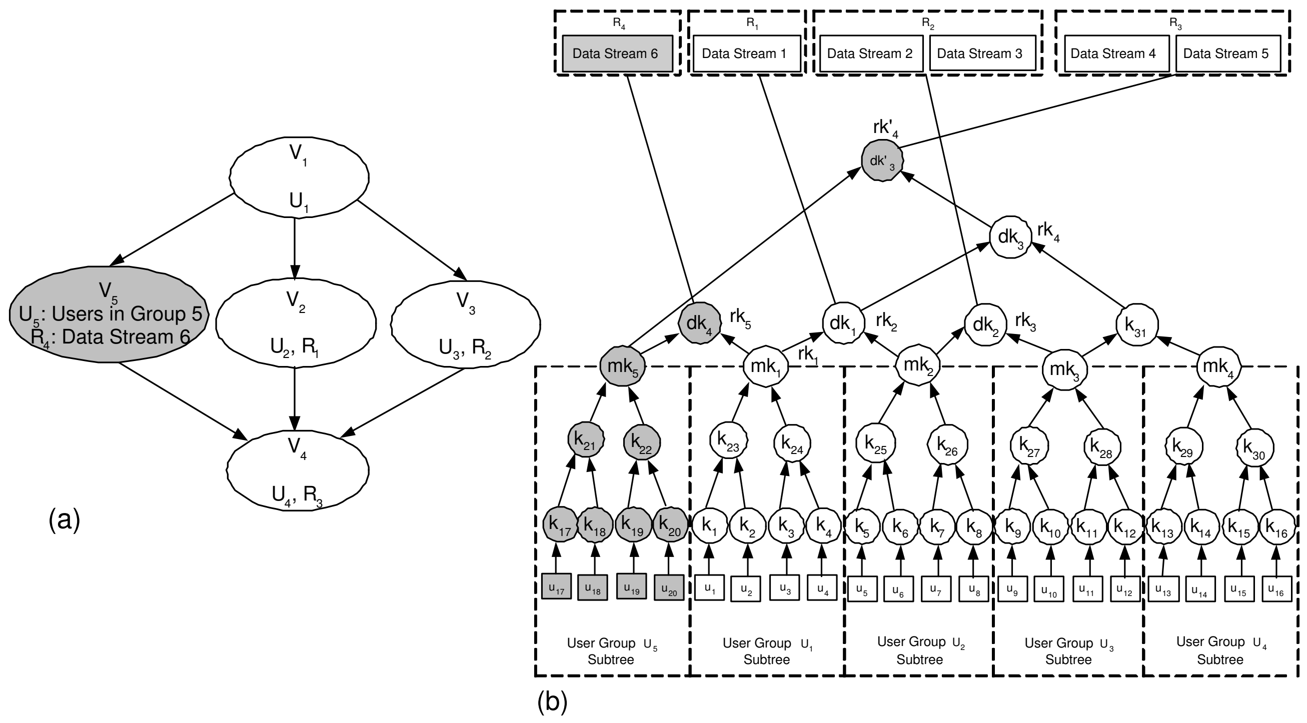

Figure 5: (a) The new unified DAG and (b) the new key graph constructed by the HAC scheme

dk_3, and generate a new key dk'_3. Key dk'_3, represented by the new k-node rk'_4, covers the new membership group V_5 and $Userset(dk_3)$, as shown in Figure 5 (b). The set of rekey messages for updating the new keys, colored grey in Figure 5 (b), are $\{k_{21}\}k_{17}$, $\{k_{21}\}k_{18}$, $\{k_{22}\}k_{19}$, $\{k_{22}\}k_{20}$, $\{mk_5\}k_{21}$, $\{mk_5\}k_{22}$, $\{dk_4\}mk_5$, $\{dk_4\}mk_1$, $\{dk'_3\}mk_5$, and $\{dk'_3\}dk_3$.

5 Performance Analysis

Storage overhead reflects the scalability of a key management scheme and rekey overhead is proportional to the cost of communication and computation for a key management scheme. Hence, in this section, we analyze the storage overhead and rekey overhead of the HAC scheme with a large number of users.

5.1 Storage Overhead

In the key graph constructed by the HAC scheme, each membership group forms a balanced membership-group subtree. For membership group U_i, let m_i notate the number of users and k_{U_i} notate the number of keys in the subtree. We assume that the membership-group subtree is balanced and fully loaded with degree d. Hence, the total number of keys in the subtree is

$$k_{U_i} = m_i \sum_{k=0}^{\log_d m_i} \frac{1}{d^k} = \frac{dm_i - 1}{d-1}. \tag{1}$$

In the construction of the relation subgraph, for a selected vertex with a resource group R_i, the cover operation constructs a balanced tree in which the membership groups that are in the selected vertex and in all vertices reachable to the selected vertex become the leaves of the balanced tree, and the resource-group key of R_i becomes the root of the balanced tree. Let N be the total number of resource groups and M be the total number of membership groups in a system. In the worst case, if keys are not able to be reused in any cover operation, the relation subgraph consists of N non-overlapped balanced trees, each of which has a resource-group key as the root and has M leaves. Let k_{R_i} be the number of keys in the balanced tree constructed for R_i and $k_{\sum R_i}$ be the total number of keys in the relation subgraph. In each of the balanced trees, since some leaves might be on the branches with length $(\lceil \log_d M \rceil - 1)$, the number of keys in the balanced tree is

$$k_{R_i} \le d^{\lceil \log_d M \rceil} \sum_{k=1}^{\lceil \log_d M \rceil} \frac{1}{d^k}. \tag{2}$$

The total number of keys in the relation subgraph is

$$k_{\sum R_i} \le N \times k_{R_i}.$$

Let S_{Server} be the storage overhead at the key server. We obtain S_{Server} by combining the number of keys on all membership-group subtrees and on the relation subgraph. Hence, we have

$$\begin{aligned} S_{Server} &= \sum_{i=1}^{M} k_{U_i} + k_{\sum R_i} \qquad (3) \\ &\le \sum_{i=1}^{M} \frac{dm_i - 1}{d-1} + N d^{\lceil \log_d M \rceil} \sum_{k=1}^{\lceil \log_d M \rceil} \frac{1}{d^k}. \end{aligned}$$

Let S_{U_i} be the storage overhead at a user in membership group U_i. In the worst case, S_{U_i} is the sum of the length of U_i subtree and the number of keys on the relation subgraph. Hence, we have

$$S_{U_i} \leq \log_d m_i + k_{\sum R_i} = \log_d m_i + N d^{\lceil \log_d M \rceil} \sum_{k=1}^{\lceil \log_d M \rceil} \frac{1}{d^k}. \quad (4)$$

Since $\lceil \log_d M \rceil \leq \log_d M + 1 = \log_d (dM)$, we have $d^{\lceil \log_d M \rceil} \leq d^{\log_d (dM)} = dM$. Also, $\sum_{k=1}^{\lceil \log_d M \rceil} \frac{1}{d^k} \leq 1$. Hence, Equations (4) and (4) can be further relaxed as

$$S_{Server} \leq \sum_{i=1}^{M} \frac{dm_i - 1}{d-1} + dMN. \quad (5)$$

$$S_{U_i} \leq \log_d m_i + dMN. \quad (6)$$

From Equations (5) and (6), if the number of users in a system is very large, the storage overhead at the key server is approximately proportional to the number of users in the system. If the number of users in a membership group is very large, the storage overhead at a user is approximately proportional to the logarithm of the number of users in its membership group.

5.2 Rekey Overhead

As described in Section 2.2, when a user joins a membership group, only one rekey message is sent out by the key server to notify the joining user of the updated keys using one-way functions [20]. When a user leaves from a membership group, the key server needs to update the keys possessed by the leaving user, some of which are on the membership subtree of the leaving user and some of which are on the relation subgraph. For a logical key tree, the number of rekey messages sent by the key server for a leaving user is $(d-1)(h-1)$, where h is the length of the path from the parent node of the leaving u-node to the root in the logical key tree [22]. Hence, when a user leaves from a membership group U_i, the number of rekey messages sent in order to update the keys from the parent node of the leaving u-node to the root of U_i subtree is up to $(d-1)(\lceil \log_d m_i \rceil - 1)$.

In the worst case, the relation subgraph consists of N non-overlapped balanced trees, each of which has a resource-group key as the root and M leaves that are the roots of membership-group subtrees. When a user leaves from a membership group U_i, the keys along N paths from the root of U_i subtree to N resource-group keys need to be updated. Hence, the number of rekey messages sent in order to update the keys on the relation subgraph possessed by the leaving user is up to $N(d-1)(\lceil \log_d M \rceil - 1)$.

Thus, we obtain the total number of rekey messages sent in order to update the keys possessed by the leaving user in a membership group U_i as

$$R_{U_i}^{Server} \leq (d-1)(\lceil \log_d m_i \rceil - 1) + N(d-1)(\lceil \log_d M \rceil - 1). \quad (7)$$

Table 3: Three cases in the simulation

No. of Data Streams	R_1	R_2	R_3	Total
Case I	1	1	1	3
Case II	1	2	2	5
Case III	2	3	3	8

Since $\lceil \log_d m_i \rceil \leq \log_d m_i + 1$, we can further relax Equation (7) as

$$R_{U_i}^{Server} \leq (d-1)(\log_d m_i + N \log_d M). \quad (8)$$

If the number of users in membership group U_i becomes very large, the rekey overhead at the key server is proportional to the logarithm of the number of users in membership group U_i. For a user leaving, the number of rekey messages received by a user is at most the number of rekey messages sent by the key server. Hence, the rekey overhead at a user is up to $R_{U_i}^{Server}$. When a user switches from membership group U_i to membership group U_j, the amount of rekey messages should be at most $R_{U_i}^{Server}$ since the keys on the relation subgraph that are possessed by users in both U_i and U_j do not need to be updated.

6 Performance Comparison

In this section, we compare the performance of the HAC scheme with the MG scheme. The performance of both schemes is measured by the storage overhead at the key server and at every user, as well as the rekey overhead of the key server and users. We develop a simulation model to construct logical key graphs with $d = 2$ based on access relations and to simulate user actions in the system.

We simulate four membership groups and three resource groups with relations as shown in Figure 1 (c). We consider three cases where there are equivalent data streams to group, and the number of data streams per resource group is shown in Table 3. Since, in the HAC scheme, all data streams in a resource group are encrypted by the same resource-group key, the logical key graphs for the three cases are the same. However, by using the MG scheme, the logical key graphs for Cases I, II, and III have 3, 5, and 8 data encryption keys respectively. The HAC scheme constructs a simpler key graph compared to the MG scheme due to the reduced data encryption keys. If there are no equivalent data streams to group in certain access relations, the number of data encryption keys required in the HAC scheme will be the same as that in the MG scheme, and then both schemes will have the same performance.

In each experiment, a sequence of user actions is generated, in which the ratio of the number of joining, leaving, and switching actions is 1:1:1. For each joining action, a user uniformly selects a membership group to join. For each switching action, a user is uniformly selected to switch from its current membership group to a membership group that is also uniformly selected. In the sim-

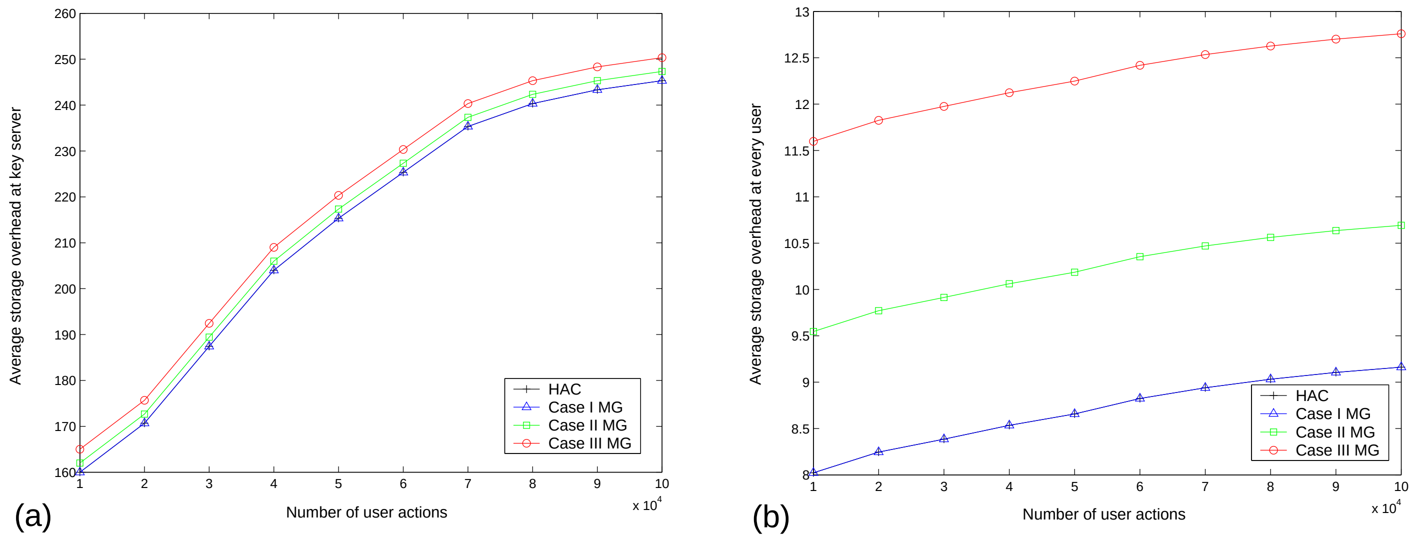

Figure 6: Storage overhead (a) at the key server and (b) at every user

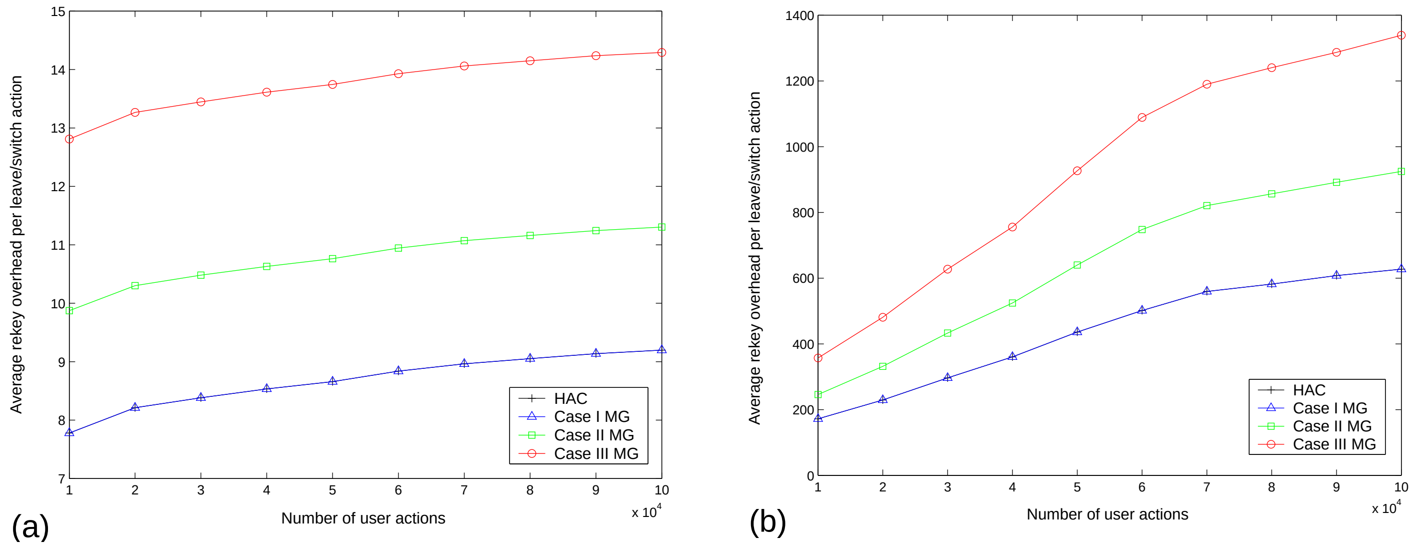

Figure 7: Rekey overhead (a) at the key server and (b) at the users

ulation, the initial number of users in each membership group is 16. The user actions are generated according to a Poisson process with a rate 0.1 action per minute. Hence, the number of users in the key graph changes over time, up to 550 users in the simulation.

Figure 6 plots the average storage overhead per user action versus the total number of user actions. Figure 6 (a) shows the average storage overhead at the key server. Since the difference of storage overhead at the key server between the two schemes primarily depends on the number of data encryption keys stored in the key server, both schemes have the same storage overhead at the key server for Case I. The HAC scheme has two fewer data encryption keys than the MG scheme for Case II, and five fewer data encryption keys than the MG scheme for Case III. Figure 6 (b) shows the average storage overhead at every user. We can see that both schemes have the same storage overhead at every user for Case I. This is because the number of data encryption keys in both schemes is three for Case I. For Cases II and III, the HAC scheme requires fewer data encryption keys, which results in less storage overhead at every user. We can also see that the storage overhead difference between the HAC scheme and Case III is much higher than the difference between the HAC scheme and Case II, which shows that the advantage of the HAC scheme becomes larger when more equivalent data streams are grouped.

Figure 7 plots rekey overhead for each leaving or switching action versus the total number of user actions. Figure 7 (a) shows the rekey overhead at the key server. Figure 7 (b) shows the rekey overhead at the users. We observe that the HAC scheme and the MG scheme have the same rekey overhead at the key server and at the users for Case I. However, the HAC scheme results in less rekey overhead than the MG scheme at the key server and at the users for Cases II and III. The rekey overhead comparison also shows that the advantage of the HAC scheme over the MG scheme becomes larger when more equivalent data streams are grouped.

7 Conclusion

In this paper, we proposed a hierarchical access control key management scheme for group communication. We employed an algorithm to construct a key graph based on a unified relation of membership groups and resource groups, which can handle complex access relations. In the key graph constructed by the hierarchical access control key management scheme, equivalent data streams are grouped in a resource group and are encrypted by a single data encryption key, which leads to fewer data encryption keys than the multi-group key management scheme. We compared the performance of the hierarchical access control key management scheme to the multi-group key management scheme by simulation. The simulation results showed that the hierarchical access control key management scheme is more efficient than the multi-group key management scheme when multiple equivalent data streams are grouped in a resource group. Also, the hierarchical access control key management scheme becomes more beneficial when more equivalent data streams are grouped. An area of future work is to employ the batch rekeying [23] scheme in order to further improve the key management efficiency. Another area of future work is to consider the distributed environments in the study.

References

[1] Y. Amir, et. al., "Secure group communication using robust contributory key agreement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 5, pp. 468-480, May 2004.

[2] S. Banerjee, and B. Bhattacharjee, "Scalable secure group communication over IP multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1511-1527, Oct. 2002.

[3] J. C. Birget, X. Zou, G. Noubir, and B. Ramamurthy, "Hierarchy-based access control in distributed environments," *proceedings, International Conference on Communication (ICC2001)*, vol. 1, pp. 229–233, June 2001.

[4] K. C. Chan, and S. H. G. Chan, "Key management approaches to offer data confidentiality for secure multicast," *IEEE Network*, vol. 17, no. 5, pp. 30-39, Sep. 2003.

[5] L. R. Dondeti, S. Mukherjee, and A. Samal, "DISEC: a distributed framework for scalable secure many-to-many communication," *Proceedings of 15th IEEE Symposium on Computers and Communications*, pp. 693–698, 2000.

[6] O. M. Erdem, "Efficient self-organized key management for mobile ad hoc networks," *IEEE Global Telecommunications Conference (GLOBECOM)*, vol. 4, pp. 2185–2190, Dec. 2004.

[7] M. G. Gouda, H. C. Tser, and E. N. Elnozahy, "Key trees and the security of interval multicast," *Proceedings of 22nd International Conference on Distributed Computing Systems*, pp. 467–468, 2002.

[8] Y. Kim, A. Perrig, G. Tsudik, "Tree-based group key agreement," *ACM Transactions on Information and System Security*, vol. 7, no. 1, pp. 60-94, Feb. 2004.

[9] S. Mittra, "Iolus: a framework for scalable secure multicasting," *Proceedings of ACM SIGCOMM*, pp. 277–288, Oct. 1997.

[10] A. Penrig, D. Song, and D. Tygar, "ELK: A new protocol for efficient large-group key distribution," *Proceedings of IEEE Symposium on Security and Privacy*, pp. 247–262, May 2001.

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

[12] S. Rafaeli, and D. Hutchison, "A survey of key management for secure group communication," *ACM Computing Surveys*, vol. 35, no. 3, pp 309-329, Sep. 2003.

[13] J. Raymond, and A. Stiglic, "Security Issues in the Diffie- Hellman Key Agreement Protocol," *IEEE Transactions on Information Theory*, pp. 1-7, 1998.

[14] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-Hellman key distribution extended to group communication," *proceedings, The 3rd ACM Conference on Computer and Communications Security*, pp. 31–37, 1996.

[15] M. Steiner, G. Tsudik, and M. Waidner, "Key agreement in dynamic peer groups, *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 8, pp. 769-80, Aug. 2000.

[16] Y. Sun, W. Trappe, and K. J. R. Liu, "A scalable multicast key management scheme for heterogenecous wireless networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 4, pp. 653-666, Aug. 2004.

[17] Y. Sun, and K. J. Ray Liu, "Scalable hierarchical access control in secure group communications" *IEEE INFOCOM*, vol. 2, pp. 1296–1306, March 2004.

[18] W. Trappe, Y. Wang, and K. J. R. Liu, "Resource-aware conference key establishment for heterogeneous networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp.134-146, Feb. 2005.

[19] J. Wang and X. Qi, "Key management for differentially secure multicast," *2nd IASTED International Conference on Communications, Internet & Information Technology*, pp. 226–231, Nov. 2003.

[20] 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.

[21] D. Wallner, E. Harder, and R. Agee, "Key management for multicast: issues and architecture," *Internet Draft*, draft-wallnet-key-arch-01.txt, Sep. 1998.

[22] C. Wong, M. Gouda, and S. Lam, "Secure group communications using key graphs," *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp.16–30, Feb. 2000.

[23] X. Zhang, S. Lam, D. Lee, and Y. Yang, "Protocol design for scalable and reliable group rekeying," *IEEE/ACM Transactions on Networking*, vol. 11, no. 6, pp. 908-922, Dec. 2003.

[24] X. Zou, *Secure Group Communications and Hierarchical Access Control*, PhD. Thesis, University of Nebraska-Lincoln, USA, 2000.

Qiong Zhang is an assistant professor of Mathematical Sciences & Applied Computing at Arizona State University. She joined the ASU faculty in 2005. She received her Ph.D. degree in Computer Science from the University of Texas at Dallas in 2005. Her current research interests include optical networks, network security, and grid computing. Her PhD dissertation earned honorable mention in the University of Texas at Dallas best dissertation competition in 2005. She was also a recipient of the Best Paper Award for the Photonic Technologies for Communications Symposium in GlobeCom 2005.

Yuke Wang received the Ph.D. degree in computer science from the University of Saskatchewan, Canada, in 1996. He is currently an Associate Professor with the University of Texas at Dallas. His research interests include network security, QoS, ASIC design and embedded processors for applications in DSP and communication systems. Dr. Wang has served as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS ART II, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, the EURASIP Journal of Applied Signal Processing, and the Journal of Circuits, Systems, and Signal Processing.

Jason P. Jue received the B.S. degree in electrical engineering from the University of California, Berkeley in 1990, the M.S. degree in electrical engineering from the University of California, Los Angeles in 1991, and the Ph.D. degree in computer engineering from the University of California, Davis in 1999.

He is an Associate Professor in the Department of Computer Science at the University of Texas at Dallas. His research interests include optical networks, network control and management, and network survivability.