Secure Group Communications Using Key Graphs

Chung Kei Wong, *Member, IEEE*, Mohamed Gouda, and Simon S. Lam, *Fellow, IEEE*

***Abstract*—Many emerging network applications (e.g., teleconference, information services, distributed interactive simulation, and collaborative work) are based upon a group communications model. As a result, securing group communications, i.e., providing confidentiality, authenticity, and integrity of messages delivered between group members, will become a critical networking issue. We present, in this paper, a novel solution to the scalability problem of group/multicast key management. We formalize the notion of a secure group as a triple (U, K, R) where U denotes a set of users, K a set of keys held by the users, and R a user-key relation. We then introduce key graphs to specify secure groups. For a special class of key graphs, we present three strategies for securely distributing rekey messages after a join/leave and specify protocols for joining and leaving a secure group. The rekeying strategies and join/leave protocols are implemented in a prototype key server we have built. We present measurement results from experiments and discuss performance comparisons. We show that our group key management service, using any of the three rekeying strategies, is scalable to large groups with frequent joins and leaves. In particular, the average measured processing time per join/leave increases linearly with the logarithm of group size.**

***Index Terms*—Confidentiality, group communications, group key management, key distribution, multicast, privacy, rekeying, security.**

I. INTRODUCTION

MOST network applications are based upon the client–server paradigm and make use of unicast (or point-to-point) packet delivery. Many emerging applications, on the other hand, are based upon a *group communications* model. In particular, they require packet delivery from one or more authorized sender(s) to a large number of authorized receivers. In the Internet, multicast has been used successfully to provide an efficient, best effort delivery service to large groups [7]. We envision that deployment of network applications requiring group communications will accelerate in coming years. As a result, securing group communications, i.e., providing confidentiality, authenticity, and integrity of messages delivered between group members, will become a critical networking issue in the near future.

While the technical issues of securing unicast communications for client–server computing are fairly well understood, the technical issues of securing group communications are not. Conceptually, since every point-to-multipoint communication can be represented as a set of point-to-point communications, the current technology base for securing unicast communications can be extended in a straightforward manner to secure group communications [12], [13]. However, such an extension is not scalable to large groups.

For a more concrete illustration of this point, we outline a typical procedure for securing unicast communications between a client and a server. Initially, the client and server mutually authenticate each other using an authentication protocol or service; subsequently, a symmetric key is created and shared by them to be used for pairwise confidential communications [4], [21], [23], [27]. This procedure can be extended to a group as follows. Let there be a trusted server which is given membership information to exercise group access control. When a client wants to join the group, the client and server mutually authenticate using an authentication protocol. Having been authenticated and accepted into the group, each member shares with the server a key,[1] to be called the member's *individual key*. For group communications, the server distributes to each member a *group key* to be shared by all members of the group.[2]

For a group of n members, distributing the group key securely to all members requires n messages encrypted with individual keys (a computation cost proportional to group size n). Each such message may be sent separately via unicast. Alternatively, the n messages may be sent as a combined message to all group members via multicast. Either way, there is a communication cost proportional to group size n (measured in terms of the number of messages or the size of the combined message).

Observe that for a point-to-point session, the costs of session establishment and key distribution are incurred just once, at the beginning of the session. A group session, on the other hand, may persist for a relatively long time with members joining and leaving the session. Consequently, the group key should be changed frequently. To achieve a high level of security, the group key should be changed after every *join* and *leave* so that a former group member has no access to current communications and a new member has no access to previous communications.

Manuscript received November 5, 1998; revised July 14, 1999; recommended by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. Partridge. This work was supported in part by the Texas Advanced Research Program under Grant 003658-063, in part by the NSA INFOSEC University Research Program under Grant MDA904-98-C-A901, and in part by the National Science Foundation under Grant ANI-9977267. Experiments were performed on equipment procured with the support of the NSF under Grant CDA-9624082. An earlier version of this paper was presented at ACM SIGCOMM'98, Vancouver, BC, Canada, September 1998.

The authors are with the Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712-1188 USA (e-mail: ckwong@hrl.com; gouda@cs.utexas.edu; lam@cs.utexas.edu).

Publisher Item Identifier S 1063-6692(00)01437-0.

---

[1] In this paper, *key* means a key from a symmetric cryptosystem, such as DES, unless explicitly stated otherwise.

[2] It is easy to see that sharing a group key enables confidential communications within a group. In addition to confidentiality, standard techniques such as digital signature and message digest can be used to provide authenticity, integrity, and nonrepudiation. We will not elaborate upon these techniques since the focus of this paper is key management.

Let there be a trusted server that creates a new group key after every join and leave. After a join, the new group key can be sent via unicast to the new member (encrypted with its individual key) and via multicast to existing group members (encrypted with the previous group key). Thus, changing the group key securely after a join is not too much work. After a leave, however, the previous group key can no longer be used and the new group key must be encrypted for each remaining group member using its individual key. Thus, we see that changing the group key securely after a leave incurs computation and communication costs proportional to n, the same as initial group key distribution. That is, large groups whose members join and leave frequently pose a scalability problem.

The topic of secure group communications has been investigated [1], [2], [11]–[13], [18]. Also the problem of how to distribute a secret to a group of users has been addressed in the cryptography literature [3], [5], [9], [22]. However, with the exception of [18], no one has addressed the need for frequent key changes and the associated scalability problem for a very large group. The approach proposed in Iolus [18] to improve scalability is to decompose a large group of clients into many subgroups and employ a hierarchy of group security agents.

A. Our Approach

We present in this paper a different hierarchical approach to improve scalability. Instead of a hierarchy of group security agents, we employ a hierarchy of keys. A detailed comparison of our approach and the Iolus approach [18] as well as an overview of related work, is given in Section VI.

We begin by formalizing the notion of a secure group as a triple (U, K, R), where U denotes a set of users, K denotes a set of keys, and $R \subset U \times K$ denotes a *user–key* relation, which specifies keys held by each user in U. In particular, each user is given a subset of keys which includes the user's individual key and a group key. We next illustrate how scalability of group key management can be improved by organizing the keys in K into a hierarchy and giving users additional keys.

Let there be a trusted server responsible for group access control and key management. In particular, the server securely distributes keys to group members and maintains the user–key relation.[3] To illustrate our approach, consider the following simple example of a secure group with nine members partitioned into three subgroups: $\{u_1, u_2, u_3\}$, $\{u_4, u_5, u_6\}$, and $\{u_7, u_8, u_9\}$. Each member is given three keys: its individual key, a key for the entire group, and a key for its subgroup. Suppose that u_1 leaves the group; the remaining eight members form a new secure group and require a new group key; also, u_2 and u_3 form a new subgroup and require a new subgroup key. To send the new subgroup key securely to u_2 (u_3), the server encrypts it with the individual key of u_2 (u_3). Subsequently, the server can send the new group key securely to members of each subgroup by encrypting it with the subgroup key. Thus by giving each user three keys instead of two, the server performs five encryptions instead of eight. As a more general example, suppose the number n of users is a power of d, and the keys in K are organized as the nodes of a full and balanced d-ary tree. When a user leaves the secure group, to distribute new keys, the server needs to perform approximately $d \log_d(n)$ encryptions (rather than $n-1$ encryptions). For a large group, say, 100 000, the savings can be very substantial.

This approach of a hierarchy of keys, organized as a rooted tree, was discovered independently by Wallner *et al.* of the National Security Agency and presented in an informational RFC [24] at about the same time as when this paper was first published as a technical report [25]. Additional contributions of this paper on protocol design, implementation, and performance analysis, not addressed in [24], are summarized below.

B. Other Contributions of This Paper

With a hierarchy of keys, there are many different ways to construct rekey messages and securely distribute them to users. We investigate three rekeying strategies: *user-oriented, key-oriented,* and *group-oriented*. We design and specify join/leave protocols based upon these rekeying strategies. For key-oriented and user-oriented rekeying, which use multiple rekey messages per join/leave, we present a technique for signing multiple messages (destined to different receivers) with a single digital signature operation. Compared to using one digital signature per rekey message, the technique provides a substantial reduction in the average server processing time of a join/leave.

The rekeying strategies and protocols are implemented in a prototype key server we have built. We performed experiments on two lightly loaded SGI Origin 200 machines, with the server running on one and up to 8192 clients on the other. From measurement results, we show that our group key management service, using any of the rekeying strategies with a key tree, is scalable; in particular, the average server processing time per join/leave increases linearly with the logarithm of group size. We found that the optimal key tree degree is four. Group-oriented rekeying provides the best performance of the three strategies on the server side but is worst of the three on the client side. User-oriented rekeying has the best performance on the client side but the worst on the server side.

The balance of this paper is organized as follows. In Section II, we introduce key graphs as a method for specifying secure groups. In Section III, we present protocols for users to join and leave a secure group as well as the three rekeying strategies. In Section IV, we present a technique for signing multiple rekey messages using a single digital signature operation. Experiments and performance results are presented in Section V. Related work and a comparison of our approach and the Iolus approach are given in Section VI. Our conclusions are in SectionVII.

II. SECURE GROUPS

A *secure group* is a triple (U, K, R) where:

- U is a finite and nonempty set of users;
- K is a finite and nonempty set of keys;
- R is a binary relation between U and K, $R \subset U \times K$, called the *user—key* relation of the secure group. User u has key k if and only if (u, k) is in R.

---

[3] In practice, such a server may be distributed or replicated to enhance reliability and performance.

Each secure group has a trusted *key server* responsible for generating and securely distributing keys in K to users in the group.[4] Specifically, the key server knows the user set U and the key set K and maintains the user–key relation R. Every user in U has a key in K, called its *individual key*, which is shared only with the key server and is used for pairwise confidential communication with the key server. There is a *group key* in K, shared by the key server and all users in U. The group key can be used by each user to send messages confidentially to other members of the group.

A. Key Graphs

A key graph is a directed acyclic graph G with two types of nodes: u*-nodes* representing users and k*-nodes* representing keys. Each u-node has one or more outgoing edges but no incoming edge. Each k-node has one or more incoming edges. If a k-node has incoming edges only and no outgoing edge, then this k-node is called a *root*. (A key graph can have multiple roots.)

Given a key graph G, it specifies a secure group (U, K, R) as follows.

1) There is a one-to-one correspondence between U and the set of u-nodes in G.
2) There is a one-to-one correspondence between K and the set of k-nodes in G.
3) (u, k) is in R if and only if G has a directed path from the u-node that corresponds to u to the k-node that corresponds to k.

As an example, the key graph in Fig. 1 specifies the following secure group:

$$\begin{aligned} U &= \{u_1, u_2, u_3, u_4\} \\ K &= \{k_1, k_2, k_3, k_4, k_{12}, k_{234}, k_{1234}\} \\ R &= \{(u_1, k_1), (u_1, k_{12}), (u_1, k_{1234}), \\ &\qquad (u_2, k_2), (u_2, k_{12}), (u_2, k_{234}), (u_2, k_{1234}), \\ &\qquad (u_3, k_3), (u_3, k_{234}), (u_3, k_{1234}), \\ &\qquad (u_4, k_4), (u_4, k_{234}), (u_4, k_{1234})\}. \end{aligned}$$

Associated with each secure group (U, K, R) are two functions, $\mathrm{keyset}()$ and $\mathrm{userset}()$, defined as follows:

$$\begin{aligned} \mathrm{keyset}(u) &= \{k \mid (u, k) \in R\} \\ \mathrm{userset}(k) &= \{u \mid (u, k) \in R\}. \end{aligned}$$

Intuitively, $\mathrm{keyset}(u)$ is the set of keys that are held by user u in U, and $\mathrm{userset}(k)$ is the set of users that hold key k in K. For examples, referring to the key graph in Fig. 1, we have $\mathrm{keyset}(u_4) = \{k_4, k_{234}, k_{1234}\}$ and $\mathrm{userset}(k_{234}) = \{u_2, u_3, u_4\}$.

We generalize the definition of function $\mathrm{keyset}()$ to any subset U' of U, and function $\mathrm{userset}()$ to any subset K' of K, in a straightforward manner, i.e., $\mathrm{keyset}(U')$ is the set of keys each of which is held by at least one user in U', and $\mathrm{userset}(K')$ is the set of users each of which holds at least one key in K'.

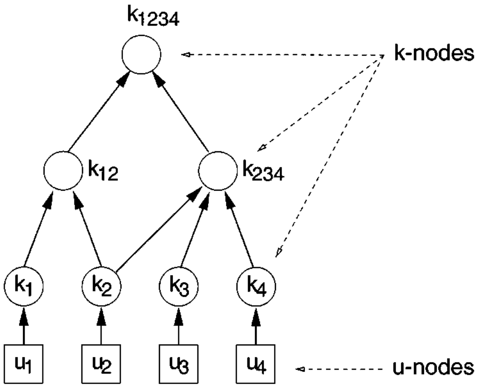


Fig. 1. A key graph.

When a user u leaves a secure group (U, K, R), every key that has been held by u and shared by other users in U should be changed. Let k be such a key. To replace k, the server randomly generates a new key k_{new} and sends it to every user in $\mathrm{userset}(k)$ except u. To do so securely, the server needs to find a subset K' of keys such that $\mathrm{userset}(K') = \mathrm{userset}(k) - \{u\}$ and use keys in K' to encrypt k_{new} for distribution. To minimize the work of rekeying, the server would like to find a minimal size set K'. This suggests the following *key-covering problem*: Given a secure group (U, K, R), and a subset S of U, find a minimum size subset K' of K such that $\mathrm{userset}(K') = S$. Unfortunately, the key-covering problem in general is NP-hard. (This is proved by showing that the NP-hard Set-Covering problem in [6] can be reduced to the Key-Covering problem in polynomial time.)

B. Special Classes of Key Graphs

We next consider key graphs with special structures for which the key covering problem can be easily solved.

Star: This is the special class of a secure group (U, K, R) where each user in U has only two keys: its individual key and a *group key* that is shared by every user in U.[5]

Tree: This is the special class of a secure group (U, K, R) whose key graph G is a single-root tree. A tree key graph (or *key tree*) is specified by two parameters.

- The *height* h of the tree is the length (in number of edges) of the longest directed path in the tree.
- The *degree* d of the tree is the maximum number of incoming edges of a node in the tree.

Note that since the leaf node of each path is a u-node, each user in U has at most h keys. Also the key at the root of the tree is shared by every user in U, and serves as the *group key*. Lastly, it is easy to see that *star* is a special case of *tree*.

Complete: This is the special class of a secure group (U, K, R), where for every nonempty subset S of U, there is a key k in K such that $\mathrm{userset}(k) = S$. Let n be the number of users in U. There are $2^n - 1$ keys in K, one for each of the $2^n - 1$ nonempty subsets of U. Moreover, each user u in U has 2^{n-1} keys, one for each of the 2^{n-1} subsets of U that contains

---

[4] Note that individual keys may have been generated and securely distributed by an authentication service and do not have to be generated by the key server.

[5] This is the base case where no additional keys are used to improve scalability of group key management.

TABLE I
NUMBER OF KEYS HELD BY THE SERVER AND BY EACH USER

	Star	Tree	Complete
Total # of keys	$n+1$	$\frac{d}{d-1}n$	2^n-1
# of keys per user	2	h	2^{n-1}

u. Since U is a subset of U, there is a key shared by every user in U, which serves as the *group key*.

The total number of keys held by the server and the number of keys held by a user are presented in Table I where n is the size of U. In particular, in the case of a complete key graph, each user needs to hold 2^n-1 keys, which is practical only for small n. Note that the number of keys in a key tree is $(d^h-1)/(d-1) \approx (d/(d-1))n$ when the tree is full and balanced (i.e., $n = d^{h-1}$).

III. REKEYING STRATEGIES AND PROTOCOLS

A user u who wants to join (leave) a secure group sends a join (leave) request to the key server, denoted by s. For a join request from user u, we assume that group access control is performed by server s using an access control list provided by the initiator of the secure group.[6] A join request initiates an authentication exchange between u and s. If user u is not authorized to join the group, server s sends a join-denied reply to u. If the join request is granted, we assume that the session key distributed as a result of the authentication exchange [10], [21], [27] will be used as the individual key k_u of u. To simplify protocol specifications below, we use the following notation:

$$s \Leftrightarrow u: \text{ authenticate } u \text{ and distribute } k_u$$

to represent the authentication exchange between server s and user u, and secure distribution of key k_u to be shared by u and s.

After each join or leave, a new secure group is formed. Server s has to update the group's key graph by replacing the keys of some existing k-nodes, deleting some k-nodes (in the case of a leave), and adding some k-nodes (in the case of a join). It then securely sends *rekey messages* containing new group/subgroup keys to users of the new secure group. (A reliable message delivery system, for both unicast and multicast, is assumed.) In protocol specifications below, we also use the following notation:

$$x \rightarrow y : z$$

to denote

- if y is a single user, the sending of message z from x to y;
- if y is a set of users, the sending of message z from x to every user in y (via multicast or unicast).

In the following subsections, we first present protocols for joining and leaving a secure group specified by a star key graph. These protocols correspond to conventional rekeying procedures informally described in the introduction. We then consider secure groups specified by tree key graphs. With a hierarchy of group and subgroup keys, rekeying after a join/leave can be carried out in a variety of ways. We present three rekeying strategies—*user-oriented*, *key-oriented*, and *group-oriented*—as well as protocols for joining and leaving a secure group.

(1) $u \rightarrow s$: join request
(2) $s \Leftrightarrow u$: authenticate u and distribute k_u
(3) s : randomly generate a new group key $k_{U'}$
(4) $s \rightarrow u$: $\{k_{U'}\}_{k_u}$
(5) $s \rightarrow U$: $\{k_{U'}\}_{k_U}$

Fig. 2. Join protocol for a star key graph.

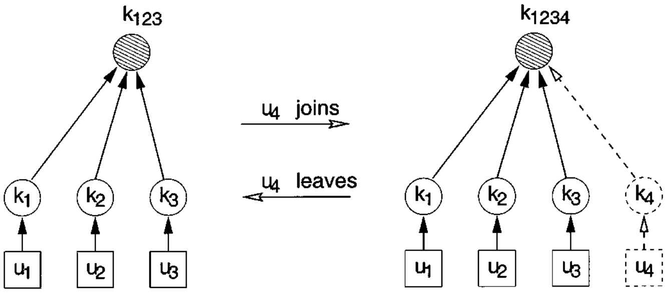


Fig. 3. Star key graphs before and after a join (leave).

A. Joining a Star Key Graph

After granting a join request from user u, server s updates the key graph by creating a new u-node for u and a new k-node for k_u, and attaching them to the root node. Server s also generates a new group key $k_{U'}$ for the root node, encrypts it with the individual key k_u of user u, and sends the encrypted new group key to u. To notify other users of the new group key, server s encrypts the new group key $k_{U'}$ with the old group key k_U, and then multicasts the encrypted new group key to every user in the group. (See Fig. 2.)

For example, as shown in Fig. 3, suppose user u_4 wants to join the left secure group in the figure, and it is allowed to join. After server s changes the group key from k_{123} to a new key k_{1234}, server s needs to send out the following two rekey messages:

$$\begin{aligned} s &\rightarrow \{u_1, u_2, u_3\} : \{k_{1234}\}_{k_{123}} \\ s &\rightarrow u_4 \qquad\qquad : \{k_{1234}\}_{k_4}. \end{aligned}$$

For clarity of presentation, we have assumed that rekey messages contain new keys only and secure distribution means that the new keys are encrypted for confidentiality only. In our prototype implementation, rekey messages have additional fields, such as, subgroup labels for new keys, server digital signature, message integrity check, etc.

B. Leaving a Star Key Graph

After granting a leave request from user u, server s updates the key graph by deleting the u-node for user u and the k-node for its individual key k_u from the key graph. Server s generates a new group key $k_{U'}$ for the new secure group without u, encrypts it with the individual key of each remaining user, and unicasts the encrypted new group key to the user. (See Fig. 4.)

---

[6] The authorization function may be offloaded to an authorization server. In this case, the authorization server provides an authorized user with a ticket to join the secure group [19], [28]. The user submits the ticket together with its join request to server s.

```
(1) $u \rightarrow s$ : { leave-request }$_{k_u}$
(2) $s \rightarrow u$ : { leave-granted }$_{k_u}$
(3)     $s$ : randomly generate a new group key $k_{U'}$
(4) for each user $v$ in $U$ except user $u$ do
        $s \rightarrow v$ : $\{k_{U'}\}_{k_v}$
```

Fig. 4. Leave protocol for a star key graph.

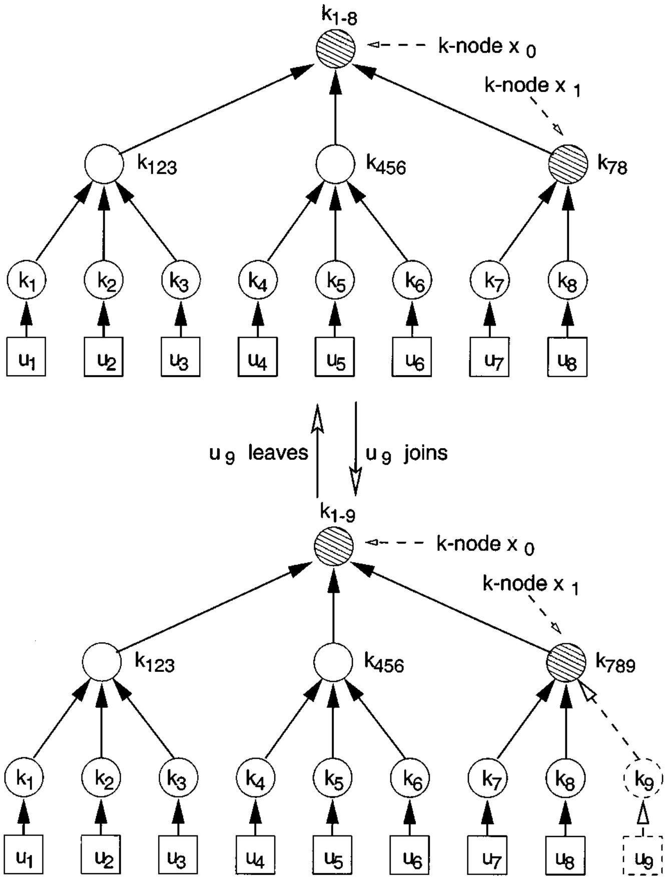


Fig. 5. Key trees before and after a join (leave).

C. Joining a Tree Key Graph

After granting a join request from u, server s creates a new u-node for user u and a new k-node for its individual key k_u. Server s finds an existing k-node (called the *joining point* for this join request) in the key tree and attaches k-node k_u to the joining point as its child.

To prevent the joining user from accessing past communications, all keys along the path from the joining point to the root node need to be changed. After generating new keys for these nodes, server s needs to securely distribute them to the existing users as well as the joining user. For example, as shown in Fig. 5, suppose u_9 is granted to join the upper secure group in the figure. The joining point is k-node k_{78} in the upper key graph, and the key of this k-node is changed to k_{789} in the lower key graph. Moreover, the group key at the root is changed from k_{1-8} to k_{1-9}. Users $u_1, \ldots, u_6$ only need the new group key k_{1-9}, while users u_7, u_8, and u_9 need the new group key k_{1-9} as well as the new subgroup key k_{789}.

To securely distribute the new keys to the users, the server constructs and sends rekey messages to the users. A *rekey message* contains one or more encrypted new key(s), and a user needs to decrypt it with appropriate keys in order to get the new keys. We next present three different approaches to construct and send rekey messages.

1) User-Oriented Rekeying: Consider each user and the subset of new keys it needs. The idea of user-oriented rekeying is that for each user, the server constructs a rekey message that contains precisely the new keys needed by the user and encrypts them using a key held by the user.

For example, as shown in Fig. 5, for user u_9 to join the upper secure group in the figure, server s needs to send the following three rekey messages:

$$\begin{aligned} s &\rightarrow \{u_1, \ldots, u_6\} &&: \{k_{1-9}\}_{k_{1-8}} \\ s &\rightarrow \{u_7, u_8\} &&: \{k_{1-9}, k_{789}\}_{k_{78}} \\ s &\rightarrow u_9 &&: \{k_{1-9}, k_{789}\}_{k_9}. \end{aligned}$$

Note that users $u_1, \ldots, u_6$ need to get the new group key k_{1-9}. There is no single key that is shared only by $u_1, \ldots, u_6$. However, key k_{1-8} can be used to encrypt the new key k_{1-9} for $u_1, \ldots, u_6$ without security breach since users u_7 and u_8 will also get this new group key from another rekey message.

User-oriented rekey messages can be constructed as follows. For each k-node x whose key has been changed, say, from k to k', the server constructs a rekey message by encrypting the new keys of k-node x and all its ancestors (up to the root) by the old key k. This rekey message is then sent to the subset of users that need precisely these new keys. Either unicast or *subgroup multicast* may be used.[7] Moreover, one rekey message is sent to the joining user, which contains all of the new keys encrypted by the individual key of the joining user.

This approach needs h rekey messages. Counting the number of keys encrypted, the encryption cost for the server is given by

$$1 + 2 + \cdots + h - 1 + h - 1 = \frac{h(h+1)}{2} - 1.$$

2) Key-Oriented Rekeying: In this approach, each new key is encrypted individually (except keys for the joining user). For each k-node x whose key has been changed, say, from k to k', the server constructs two rekey messages. First, the server encrypts the new key k' with the old key k and sends it to $\mathrm{userset}(k)$, which is the set of users that share k. All of the original users that need the new key k' can get it from this rekey message. The other rekey message contains the new key k' encrypted by the individual key of the joining user and is sent to the joining user.

As described above, a user may have to get multiple rekey messages in order to get all the new keys it needs. For example, as shown in Fig. 5, for user u_9 to join the upper secure group in the figure, server s needs to send the following four rekey

---

[7] A rekey message can be sent via multicast to a subgroup if a multicast address has been established for the subgroup in addition to the multicast address for the entire group. Alternatively, the method in [16] may be used in lieu of allocating a large number of multicast addresses for subgroups. See Section VII for more discussion.

```
(1) $u \rightarrow s$ : join request
(2) $s \Leftrightarrow u$ : authenticate $u$ and distribute $k_u$
(3)        $s$ : find a joining point and attach $k_u$;
           let $x_j$ denote the joining point,
             $x_0$ the root, and for $i = 1, \ldots, j$
             $x_{i-1}$ the parent of $x_i$;
           let $K_{j+1}$ denote $k_u$, and $K_0, \ldots, K_j$
             the old keys of $x_0, \ldots, x_j$;
           randomly generate new keys
             $K'_0, \ldots, K'_j$
(4) for $i = 0$ upto $j$ do
        $s \rightarrow (userset(K_i) - userset(K_{i+1}))$ :
             $\{K'_0\}_{K_0}, \ldots, \{K'_i\}_{K_i}$
(5) $s \rightarrow u$ : $\{K'_0, \ldots, K'_j\}_{k_u}$
```

Fig. 6. Join protocol for a tree key graph (key-oriented rekeying).

messages. Note that users u_7, u_8, and u_9 need to get two rekey messages each

$$\begin{array}{ll} s \rightarrow \{u_1, \ldots, u_8\} & : \{k_{1-9}\}_{k_{1-8}} \\ s \rightarrow u_9 & : \{k_{1-9}\}_{k_9} \\ s \rightarrow \{u_7, u_8\} & : \{k_{789}\}_{k_{78}} \\ s \rightarrow u_9 & : \{k_{789}\}_{k_9}. \end{array}$$

Compared to user-oriented rekeying, the above approach reduces the encryption cost of the server from $(h(h+1)/2)-1$ to $2(h-1)$, but it requires $2(h-1)$ rekey messages instead of h.

To reduce the number of rekey messages, all of the rekey messages for a particular user can be combined and sent as one message. Thus, server s can send the following three rekey messages instead of the four rekey messages shown above:

$$\begin{array}{ll} s \rightarrow \{u_1, \ldots, u_6\} & : \{k_{1-9}\}_{k_{1-8}} \\ s \rightarrow \{u_7, u_8\} & : \{k_{1-9}\}_{k_{1-8}}, \{k_{789}\}_{k_{78}} \\ s \rightarrow u_9 & : \{k_{1-9}, k_{789}\}_{k_9}. \end{array}$$

The join protocol based upon this rekeying strategy is presented in Fig. 6. Steps (4) and (5) in Fig. 6 specify how the combined rekey messages are constructed and distributed by server s.

Using combined rekey messages, the number of rekey messages for key-oriented rekeying is h (same as user-oriented rekeying), while the encryption cost is $2(h-1)$. From this analysis, key-oriented rekeying is clearly better for the server than user-oriented rekeying. (This conclusion is confirmed by measurement results presented in Section V.)

3) Group-Oriented Rekeying: In key-oriented rekeying, each new key is encrypted individually (except keys for the joining user). The server constructs multiple rekey messages, each tailored to the needs of a subgroup. Specifically, the users of a subgroup receive a rekey message containing precisely the new keys each needs.

An alternative approach, called group-oriented, is for the server to construct a single rekey message containing all new keys. This rekey message is then multicasted to the entire group. Clearly, such a rekey message is relatively large and contains information not needed by individual users. However, scalability is not a concern because the message size is $O(\log_d(n))$ for group size n and key tree degree d. The group-oriented approach has several advantages over key-oriented and user-oriented rekeying. First, multicast can be used instead of unicast or subgroup multicast. Second, with fewer rekey messages, the server's per rekey message overheads are reduced. Third, the total number of bytes transmitted by the server per join/leave request is much less than those of key-oriented and user-oriented rekeying which duplicate information in rekey messages. (See Sections V and VII for a more thorough discussion on performance comparisons.)

```
(1) - (3) (same as Figure 6)
(4) $s \rightarrow userset(K_0)$ : $\{K'_0\}_{K_0}, \ldots, \{K'_j\}_{K_j}$
(5)            $s \rightarrow u$ : $\{K'_0, \ldots, K'_j\}_{k_u}$
```

Fig. 7. Join protocol for a tree key graph (group-oriented rekeying).

For example, as shown in Fig. 5, for user u_9 to join the upper secure group in the figure, server s needs to send the following two rekey messages; one is multicasted to the group and the other is unicasted to the joining user:

$$\begin{array}{ll} s \rightarrow \{u_1, \ldots, u_8\} & : \{k_{1-9}\}_{k_{1-8}}, \{k_{789}\}_{k_{78}} \\ s \rightarrow u_9 & : \{k_{1-9}, k_{789}\}_{k_9}. \end{array}$$

The join protocol based upon group-oriented rekeying is presented in Fig. 7. This approach reduces the number of rekey messages to one multicast message and one unicast message, while maintaining the encryption cost at $2(h-1)$, which is the same as key-oriented rekeying.

D. Leaving a Tree Key Graph

After granting a leave request from user u, server s updates the key graph by deleting the u-node for user u and the k-node for its individual key from the key graph. The parent of the k-node for its individual key is called the *leaving point*.

To prevent the leaving user from accessing future communications, all keys along the path from the leaving point to the root node need to be changed. After generating new keys for these k-nodes, server s needs to securely distribute them to the remaining users. For example, as shown in Fig. 5, suppose u_9 is granted to leave the lower secure group in the figure. The leaving point is the k-node for k_{789} in the lower key graph, and the key of this k-node is changed to k_{78} in the upper key graph. Moreover, the group key is also changed from k_{1-9} to k_{1-8}. Users $u_1, \ldots, u_6$ only need to know the new group key k_{1-8}. Users u_7 and u_8 need to know the new group key k_{1-8} and the new subgroup key k_{78}.

To securely distribute the new keys to users after a leave, we revisit the three rekeying strategies.

1) User-Oriented Rekeying: In this approach, each user gets a rekey message in which all the new keys it needs are encrypted using a key it holds. For example, as shown in Fig. 5, for user u_9 to leave the lower secure group in the figure, server s needs to send the following four rekey messages:

$$\begin{array}{ll} s \rightarrow \{u_1, u_2, u_3\} & : \{k_{1-8}\}_{k_{123}} \\ s \rightarrow \{u_4, u_5, u_6\} & : \{k_{1-8}\}_{k_{456}} \\ s \rightarrow u_7 & : \{k_{1-8}, k_{78}\}_{k_7} \\ s \rightarrow u_8 & : \{k_{1-8}, k_{78}\}_{k_8}. \end{array}$$

User-oriented rekey messages for a leave can be constructed as follows. For each k-node x whose key has been changed, say,

```
(1) $u \to s$ : { leave-request }$_{k_u}$
(2) $s \to u$ : { leave-granted }$_{k_u}$
(3)     $s$ : find the leaving point (parent of $k_u$);
          remove $k_u$ from the tree;
          let $x_{j+1}$ denote the deleted k-node for
            $k_u$, $x_j$ the leaving point, $x_0$ the root,
            and for $i = 1, \ldots, j$
            $x_{i-1}$ the parent of $x_i$;
          randomly generate keys $K'_0, \ldots, K'_j$
            as the new keys of $x_0, \ldots, x_j$
(4) for $i = 0$ upto $j$ do
      for each child $y$ ($\neq x_{i+1}$) of $x_i$ do
        let $K$ denote the key at k-node $y$;
        $s \to userset(K)$ :
          $\{K'_i\}_K, \{K'_{i-1}\}_{K'_i}, \ldots, \{K'_0\}_{K'_1}$
```

Fig. 8. Leave protocol for a tree key graph (key-oriented rekeying).

from k to k', and for each unchanged child y of x, the server constructs a rekey message by encrypting the new keys of k-node x and all its ancestors (up to the root) by the key K of k-node y. This rekey message is then multicasted to $\mathrm{userset}(K)$.

This approach requires $(d-1)(h-1)$ rekey messages. The encryption cost for the server is given by

$$(d-1)(1+2+\cdots+h-1) = \frac{(d-1)h(h-1)}{2}.$$

2) Key-Oriented Rekeying: In this approach, each new key is encrypted individually. For example, as shown in Fig. 5, for user u_9 to leave the lower secure group in the figure, server s needs to send the following four rekey messages:

$$\begin{aligned} s &\to \{u_1, u_2, u_3\} : \{k_{1-8}\}_{k_{123}} \\ s &\to \{u_4, u_5, u_6\} : \{k_{1-8}\}_{k_{456}} \\ s &\to u_7 \quad : \{k_{1-8}\}_{k_{78}}, \{k_{78}\}_{k_7} \\ s &\to u_8 \quad : \{k_{1-8}\}_{k_{78}}, \{k_{78}\}_{k_8}. \end{aligned}$$

The leave protocol based upon key-oriented rekeying is presented in Fig. 8. Step (4) in Fig. 8 specifies how the rekey messages are constructed and distributed to users.

Note that by storing encrypted new keys for use in different rekey messages, the encryption cost of this approach is $d(h-1)$, which is much less than that of user-oriented rekeying. The number of rekey messages is $(d-1)(h-1)$, same as user-oriented rekeying.

3) Group-Oriented Rekeying: A single rekey message is constructed containing all new keys. For example, as shown in Fig. 5, for user u_9 to leave the lower secure group in the figure, server s needs to send the following rekey message:

let L_0 denote $\{k_{1-8}\}_{k_{123}}, \{k_{1-8}\}_{k_{456}}, \{k_{1-8}\}_{k_{78}}$
let L_1 denote $\{k_{78}\}_{k_7}, \{k_{78}\}_{k_8}$
$s \to \{u_1, \ldots, u_8\} : L_0, L_1$.

Note that for a leave, this single rekey message is about d times bigger than the rekey message for a join, where d is the average degree of a k-node.

The leave protocol based upon group-oriented rekeying is presented in Fig. 9. This approach uses only one rekey message which is multicasted to the entire group, and the encryption cost is $d(h-1)$, same as key-oriented rekeying.

```
(1) - (3) (same as Figure 8)
(4) for $i = 0$ upto $j$ do
      let $J_1, \ldots, J_r$ denote keys at the children
        of $x_i$ in the new key tree;
      let $L_i$ denote $\{K'_i\}_{J_1}, \ldots, \{K'_i\}_{J_r}$;
      $s \to userset(K'_0)$ : $L_0, \ldots, L_j$
```

Fig. 9. Leave protocol for a tree key graph (group-oriented rekeying).

TABLE II
COST OF A JOIN/LEAVE REQUEST

(a)	the requesting user		
	Star	Tree	Complete
join	1	$h-1$	2^n-1
leave	0	0	0
(b)	a non-requesting user		
	Star	Tree	Complete
join	1	$\frac{d}{d-1}$	2^{n-1}
leave	1	$\frac{d}{d-1}$	0
(c)	the server		
	Star	Tree	Complete
join	2	$2(h-1)$	$2^{n+1}-2$
leave	$n-1$	$d(h-1)$	0

TABLE III
AVERAGE COST PER REQUEST

	Star	Tree	Complete
Cost of server	$n/2$	$(d+2)(h-1)/2$	2^n
Cost of a user	1	$d/(d-1)$	2^n

E. Encryption and Decryption Costs

An approximate measure of the computational costs of the server and users is the number of key encryptions and decryptions required by a join/leave request. Let n be the number of users in a secure group. For each join/leave request, the user that requests the join/leave is called the *requesting user,* and the other users in the group are *nonrequesting users.* For a join/leave request, we tabulate the cost of a requesting user in Table II(a), the cost of a nonrequesting user in Table II(b), and the cost of the server in Table II(c). These costs are from the protocols described above for star and tree key graphs and from [25] for complete key graphs. (Key-oriented or group-oriented rekeying is assumed for tree key graphs.)

For a key tree, recall that d and h denote the degree and height of the tree, respectively. In this case, for a nonrequesting user u, the average cost of u for a join or a leave is less than $d/(d-1)$, which is independent of the size of the tree (see derivation in Appendix A).

Assuming that a request is equally likely to be a join or a leave, and the group size n is large, the average costs per request are tabulated in Table III for the server and a user in the group.

From Table III, it is obvious that complete key graphs should not be used. On the other hand, scalable group key management can be achieved by using tree key graphs. Note that for a full and balanced d-ary tree, the average server cost is $(d+2)(h-1)/2 = (d+2)(\log_d(n))/2$. However, each user has to do slightly more work [from 1 to $d/(d-1)$]. For $d = 4$, a user needs to do 1.33 key decryptions on the average instead of one. (It can be shown that the server cost is minimized for $d = 4$, i.e., the optimal degree of key trees is four.)

TABLE IV
AVERAGE REKEY MESSAGE SIZE AND SERVER PROCESSING TIME ($n = 8192$, DES, MD5, RSA)

key tree degree 4	one signature per rekey msg					one signature for all rekey msgs				
	msg size (byte)		proc time (msec)			msg size (byte)		proc time (msec)		
	join	leave	join	leave	ave	join	leave	join	leave	ave
user-oriented	263.1	233.8	76.7	204.6	140.6	312.8	306.9	13.6	17.1	15.3
key-oriented	303.0	270.9	76.3	203.8	140.1	352.8	344.0	13.1	15.9	14.5
group-oriented	525.5	1005.7	11.9	12.0	11.9	525.5	1005.7	11.9	12.0	11.9

IV. TECHNIQUE FOR SIGNING REKEY MESSAGES

In our join/leave protocols, each rekey message contains one or more new keys. Each new key, destined for a set of users, is encrypted by a key known only to these users and the server. It is possible for a user to masquerade as the server and send out rekey messages to other users. Thus if users cannot be trusted, then each rekey message should be digitally signed by the server.

We note that a digital signature operation is around two orders of magnitude slower than a key encryption using DES. For this reason, it is highly desirable to reduce the number of digital signature operations required per join/leave. If each rekey message is signed individually, then group-oriented rekeying, using just one rekey message per join/leave for all users, would be far superior to key-oriented (user-oriented) rekeying, which uses many rekey messages per join/leave.

Consider m rekey messages, $M_1, \ldots, M_m$, with message digests $d_i = h(M_i)$ for $i = 1, \ldots, m$, where $h()$ is a secure message digest function such as MD5. The standard way to provide authenticity is for the server to sign each message digest (with its private key) and send the signed message digest together with the message. This would require m digital signature operations for m messages.

We next describe a technique, implemented in our prototype key server, for signing a set of messages destined to different receivers using just a single digital signature operation. The technique is based upon a scheme proposed by Merkle [17].

Suppose there are four messages, $M_1, \ldots, M_4$, with message digests d_1, d_2, d_3, and d_4. Compute message digests $d_{12} = h(d_1, d_2)$, $d_{34} = h(d_3, d_4)$, and $d_{1-4} = h(d_{12}, d_{34})$. The server signs message digest d_{1-4} with its private key. The server then sends the signed message digest, $\mathrm{sign}(d_{1-4})$, together with d_3, d_{12}, and M_4 to a user that needs M_4. Upon receipt, the user computes d'_4 from M_4, and then computes d'_{34} from d_3 and d'_4. It computes d'_{1-4} from d_{12} and d'_{34}, and uses it to verify the received signature $\mathrm{sign}(d_{1-4})$. The above example can be easily extended to m messages in general (see [26]).

The benefits of this technique for signing rekey messages are demonstrated in Table IV for both key-oriented and user-oriented rekeying. (Note that it is not needed by group-oriented rekeying which uses one rekey message per join/leave.) The average rekey message size per join/leave is shown, as well as the server's processing time per join/leave (*ave* denotes the average of average join and leave processing times). The experiments were performed for an initial group size of 8192, with DES-CBC encryption, MD5 message digest, and RSA digital signature (512-bit modulus). Additional details of our experimental setup can be found in Section V. With the technique for signing rekey messages, the processing time reduction for key-oriented and user-oriented rekeying is about a factor of ten (for example, 14.5 ms versus 140.1 ms in the case of key-oriented rekeying). There is however a small increase (around 50–70 bytes) in the average rekey message size.

V. EXPERIMENTS AND PERFORMANCE COMPARISONS

We have designed and constructed a prototype group key server, as well as a client layer, which implement join/leave protocols for all three rekeying strategies in Section III and the technique for signing rekey messages in Section IV.

We performed a large number of experiments to evaluate the performance of the rekeying strategies and the technique for signing rekey messages. The experiments were carried out on two lightly loaded SGI Origin 200 machines running IRIX 6.4. The machines were connected by a 100-Mbps Ethernet. The key server process runs on one SGI machine. The server is initialized from a specification file, which determines the initial group size, the rekeying strategy, the key tree degree, the encryption algorithm, the message digest algorithm, the digital signature algorithm, etc. A client simulator runs on the other SGI simulating a large number of clients. Actual rekey messages, as well as *join, join-ack, leave, leave-ack* messages, are sent between individual clients and the server using UDP over the 100-Mbps Ethernet. Cryptographic routines from the publicly available CryptoLib library are used [14].

For each experiment with an initial group size n, the client simulator first sent n join requests, and the server built a key tree. Then the client simulator sent 1000 join/leave requests. The sequence of 1000 join/leave requests was generated randomly according to a given ratio (the ratio was 1:1 in all our experiments to be presented). Each experiment was performed with three different sequences of 1000 join/leave requests. For fair comparisons (between different rekeying strategies, key trees of different degrees, etc.), the same three sequences were used for a given group size. The server employs a heuristic that attempts to build and maintain a key tree that is full and balanced. However, since the sequence of join/leave requests is randomly generated, it is unlikely that the tree is truly full and balanced at any time.

To evaluate the performance of different rekeying strategies as well as the technique for signing rekey messages, we measured rekey message sizes (in bytes) and processing time (in ms) used by the server per join/leave request. Specifically, the processing time per join/leave request consists of the following components. First, the server parses a request, traverses the key graph to determine which keys are to be updated, generates new keys, and updates the key graph. Second, the server performs encryption of new keys and constructs rekey messages. Third,

TABLE V
NUMBER AND SIZE OF REKEY MESSAGES, WITH ENCRYPTION AND SIGNATURE, SENT BY THE SERVER (INITIAL GROUP SIZE 8192)

key tree	rekey msg size (byte)						number of rekey msgs					
degree 4	per join			per leave			per join			per leave		
	ave	min	max	ave	min	max	ave	min	max	ave	min	max
user-oriented	312.8	196	552	306.9	228	412	7.00	6	7	19.02	18	20
key-oriented	352.8	212	616	344.0	244	476	7.00	6	7	19.02	18	20
group-oriented	525.5	356	564	1005.7	968	1076	1.00	1	1	1.00	1	1
key tree	rekey msg size (byte)						number of rekey msgs					
degree 8	per join			per leave			per join			per leave		
	ave	min	max	ave	min	max	ave	min	max	ave	min	max
user-oriented	287.3	196	496	285.9	228	356	5.00	4	5	29.01	28	30
key-oriented	319.3	212	544	314.3	244	404	5.00	4	5	29.01	28	30
group-oriented	464.5	284	492	1293.1	1256	1364	1.00	1	1	1.00	1	1
key tree	rekey msg size (byte)						number of rekey msgs					
degree 16	per join			per leave			per join			per leave		
	ave	min	max	ave	min	max	ave	min	max	ave	min	max
user-oriented	274.0	180	452	282.4	244	344	4.00	3	4	46.01	45	47
key-oriented	302.0	196	492	306.6	260	384	4.00	3	4	46.01	45	47
group-oriented	427.8	248	456	1869.1	1832	1940	1.00	1	1	1.00	1	1

TABLE VI
NUMBER AND SIZE OF REKEY MESSAGES, WITH ENCRYPTION AND SIGNATURE, RECEIVED BY A CLIENT (INITIAL GROUP SIZE 8192)

key tree degree 4	rekey msg size (byte) per join average	per leave average	# of rekey msgs per join/leave
user-oriented	209.3	237.4	1
key-oriented	227.9	256.0	1
group-oriented	525.5	1005.7	1
key tree degree 8	rekey msg size (byte) per join average	per leave average	# of rekey msgs per join/leave
user-oriented	200.0	242.0	1
key-oriented	217.2	259.2	1
group-oriented	464.5	1293.1	1
key tree degree 16	rekey msg size (byte) per join average	per leave average	# of rekey msgs per join/leave
user-oriented	197.8	246.7	1
key-oriented	214.3	263.2	1
group-oriented	427.8	1869.1	1

if message digest is specified, the server computes message digests of the rekey messages. Fourth, if digital signature is specified, the server computes message digests and a digital signature as described in Section IV. Last, the server sends out rekey messages as UDP packets using socket system calls.[8]

Table VI presents the size and number of rekey messages received by a client. Only the *average* message sizes are shown, because the minimum and maximum sizes are the same as those in Table V. Note that each client gets exactly one rekey message for all three rekeying strategies. For key-oriented and user-oriented rekeying, the average message size is smaller than the corresponding average message size in Table V. This is because the average message size here was calculated over all clients, and many more clients received small rekey messages than clients that received large rekey messages. The results in this table show that group-oriented rekeying, which has the best performance on the server side, requires more work on the client side to process a larger message than key-oriented and user-oriented rekeying. The average rekey message size on the client side is the smallest in user-oriented rekeying.

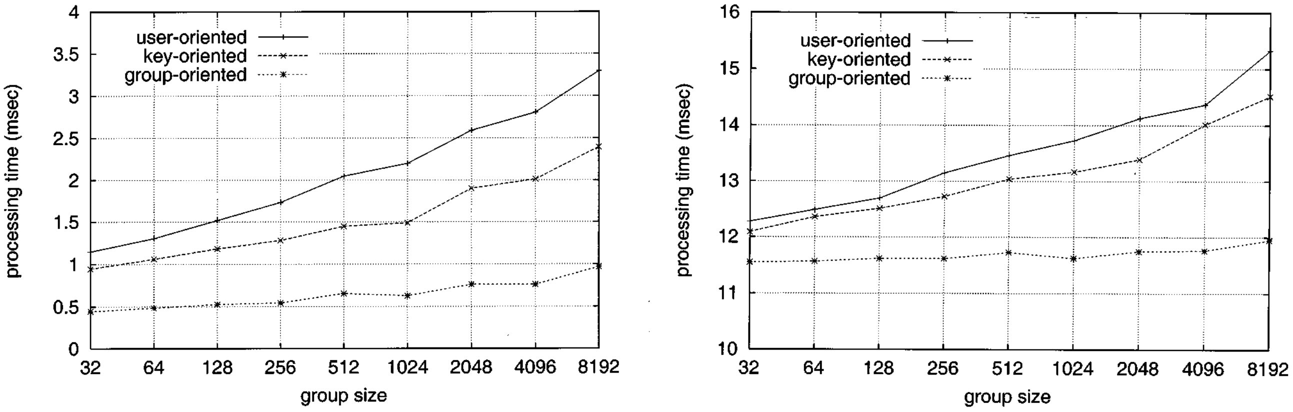
The server processing time per request (averaged over joins and leaves) versus group size (from 32 to 8192) is shown in Fig. 10. Note that the horizontal axis is in log scale. The left figure is for rekey messages with DES-CBC encryption only (no message digest and no digital signature). The right figure is for rekey messages with DES-CBC encryption, MD5 message digest, and RSA-512 digital signature. The key tree degree was four in all experiments. We conclude from the experimental results that our group key management service is scalable to very large groups since the processing time per request increases (approximately) linearly with the logarithm of group size for all three rekeying strategies. Other experiments support the same conclusion for key tree degrees of eight and 16.

The average server processing time versus key tree degree is shown in Figs. 11–13. The initial group size was 8192 in these experiments. The left-hand side of each figure is for rekey messages with DES-CBC encryption only (no message digest and no digital signature). The right-hand side of each figure is for rekey messages with DES-CBC encryption, MD5 message digest, and RSA-512 digital signature. These experimental results illustrate three observations. First, as shown in Fig. 13, the optimal degree for key trees is around four. Second, with respect to server processing time, group-oriented rekeying has the best performance, with key-oriented rekeying in second place. Third, signing rekey messages increases the server processing time by an order of magnitude (it would be another order of magnitude more for key-oriented and user-oriented rekeying without a special technique for signing multiple messages).

Table V presents the size and number of rekey messages sent by the server. Note that group-oriented rekeying uses a single large rekey message per request (sent via group multicast), while key-oriented and user-oriented rekeying use multiple smaller rekey messages per request (sent via subgroup multicast

---

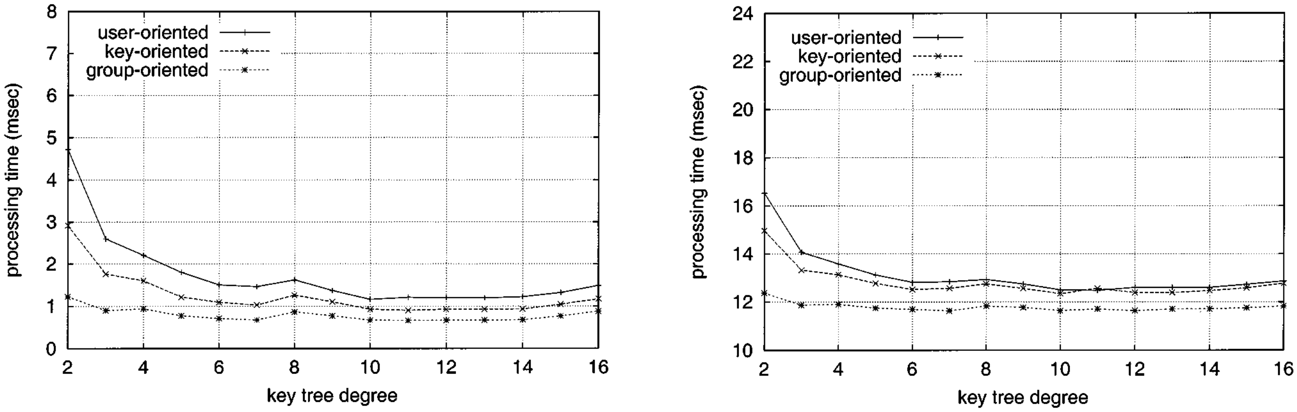
[8] The processing time is measured using the UNIX system call `getrusage()`, which returns processing time (including time of system calls) used by a process. In the results presented herein, the processing time for a join request does not include any time used to authenticate the requesting user [i.e., step (2) in the join protocols of Figs. 6 and 7]. We feel that any authentication overhead should be accounted for separately.



(a)

(b)

Fig. 10. Server processing time per request versus group size (key tree degree 4). (a) Encryption only and (b) encryption and signature.



(a)

(b)

Fig. 11. Server processing time per join versus key tree degree (initial group size 8192). (a) Encryption only and (b) encryption and signature.

or unicast).[9] Note that the total number of bytes per join/leave transmitted by the server is much higher in key-oriented and user-oriented rekeying than in group-oriented rekeying.

From the contents of rekey messages, we counted and computed the average number of key changes by a client per join/leave request, which is shown in Fig. 14. The left figure shows the average number of key changes versus the key tree degree, and the right figure shows the average number of key changes versus the initial group size of each experiment. Note that the average number of key changes by a client is small and is very close to the analytical result $d/(d{-}1)$ shown in Table III in Section III.

VI. Related Work

Various cryptographic techniques have been proposed to address the problem of distributing a secret from a source to a set of destinations. Chiou and Chen proposed a method called *secure lock* implemented using the Chinese remainder theorem [5]. The times to compute the lock and the length of the lock (size of transmission) are both proportional to the number n of destinations. (Hence it is not scalable for the purposes of this paper.)

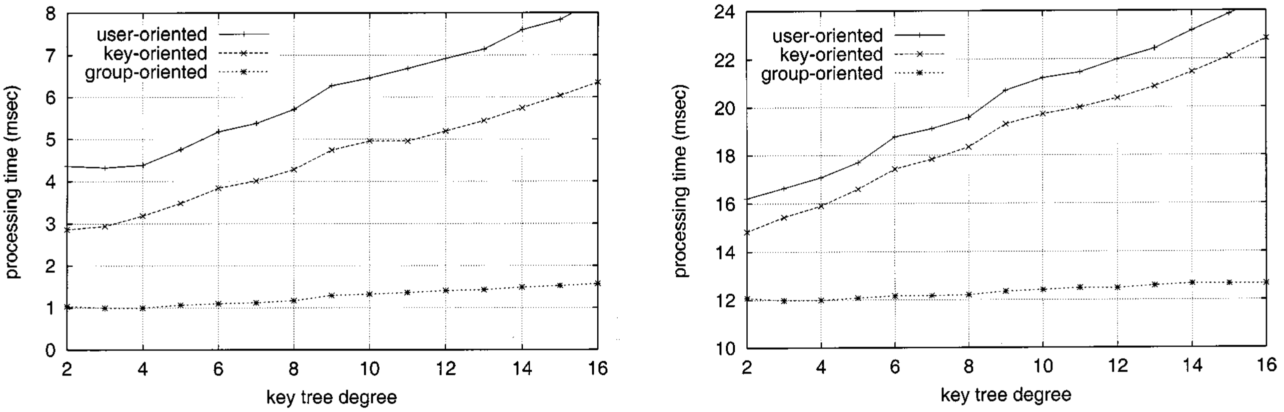
Berkovits [3] proposed the use of k out of m secret sharing. To distribute a new secret to n destinations, the source needs to compute at least n new "shares" and send them to each of the n destinations. Thus, the communication cost is proportional to n, the number of destinations. The computing cost is at least $O(n)$.

Deng *et al.* [8] proposed the use of systematic linear block codes to distribute a secret to n destinations. The transmission overhead of their approach is independent of the size of the secret but is still proportional to n, the number of destinations. The computing cost is at least $O(n)$.

Fiat and Naor [9] introduced the concept of k-resilient broadcast. In their approach, a secret distributed to a subset of n destinations is resilient to collusion by up to k other destinations. (Note that our approach is resilient to collusion by any number of destinations not belonging to the group of authorized receivers.) The most interesting scheme requires each destination to store $O(k\log(k)\log(n))$ keys and the source to broadcast $O(k^2\log^2(k)\log(n))$ messages to distribute a new secret. Some recent results on this approach by Stinson can be found in [22].

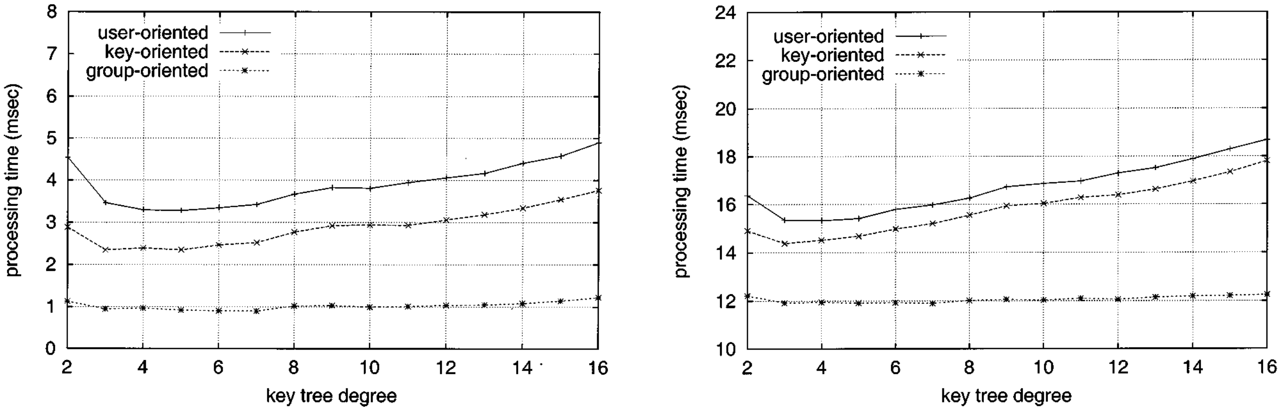
Another related area is process group security in distributed computing systems, e.g., protocols in Rampart [20]. These protocols, designed for highly secure and fault-tolerant systems, are very complicated and appropriate only for small groups.

[9] The experiments reported herein were performed with each rekey message sent just once by the server via subgroup multicast.



(a) (b)

Fig. 12. Server processing time per leave versus key tree degree (initial group size 8192). (a) Encryption only and (b) encryption and signature.



(a) (b)

Fig. 13. Server processing time per request versus key tree degree (initial group size 8192). (a) Encryption only and (b) encryption and signature.

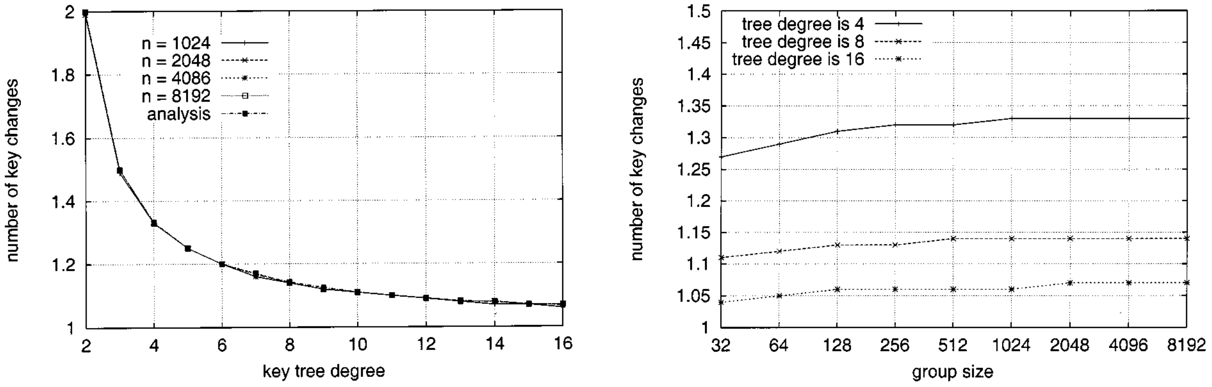


Fig. 14. Number of key changes by a client per request.

Security issues in the area of groupware or computer-supported cooperative work (CSCW) are also related. In particular, Enclaves [11] is a toolkit designed for building applications for secure collaboration over the Internet. However, the problem of scalable group key management was not addressed [11].

Security issues of IP multicast have been addressed to some extent [2]. A number of group key management protocols have been proposed for the Internet [1], [12], [13]. Their concern is the distribution of group keying material to users joining a group. No solution was proposed for changing the group keying material when users leave a group, except the obvious approach of establishing a new secure group, which is clearly not scalable.

The scalability problem of group key management for a large group with frequent joins and leaves was previously addressed by Mittra with his Iolus system [18]. Both Iolus and our approach solve the scalability problem by making use of a hierarchy. The similarity, however, ends here. The system architectures are very different in the two approaches. We next compare them by considering a tree hierarchy with a single root (i.e., a single secure group).

Iolus's tree hierarchy consists of clients at the leaves with multiple levels of group security agents (agents, in short) above. For each tree node, the tree node (an agent) and its children (clients or lower level agents) form a subgroup and share a subgroup key. There is no globally shared group key. Thus a join or a leave in a subgroup does not affect other subgroups; only the local subgroup key needs to be changed.

Our tree hierarchy consists of keys, with individual keys at leaves, the group key at the root, and subgroup keys elsewhere. There is a single key server for all the clients. There are no agents, but each client is given multiple keys (its individual key, the group key, and some subgroup keys).

In comparing the two approaches, there are several issues to consider: performance, trust, and reliability.

A. Performance

Roughly speaking, since both approaches make use of a hierarchy, both attempt to change a $O(n)$ problem into a $O(\log(n))$ problem where n denotes group size. They differ, however, in where and when work is performed to achieve secure rekeying when a client joins/leaves the secure group.

Secure rekeying after a leave requires more work than after a join because, unlike a join, the previous group key cannot be used and n rekey messages are required (this is referred to in [18] as a **1 does not equal n** type problem). This is precisely the problem solved by using a hierarchy in both approaches.

The main difference between Iolus and our approach is in how the **1 affects n** type problem [18] is addressed. In our approach, every time a client joins/leaves the secure group, a rekeying operation is required, which affects the entire group. Note that this is not a scalability concern in our approach because the server cost is $O(\log(n))$ and the client cost is $O(1)$.

In Iolus, there is no globally shared group key with the apparent advantage that whenever a client joins/leaves a subgroup, only the subgroup needs to be rekeyed. However, for a client to send a message confidentially to the entire group, the client needs to generate a *message key* for encrypting the message and the message key has to be securely distributed to the entire group via agents. Each agent decrypts using one subgroup key to retrieve the message key and reencrypts it with another subgroup key for forwarding [18].

That is, most of the work in handling the **1 affects n** type problem is performed in Iolus when a client sends a message confidentially to the entire group (rather than when a client joins/leaves the group). In our approach, most of the work in handling the **1 affects n** type problem is performed when a client joins/leaves the secure group (rather than when a client later sends messages confidentially to the entire group).

B. Trust

Our architecture requires a single trusted entity, namely, the key server. The key server may be replicated for reliability/performance enhancement, in which case, several trusted entities are needed. Each trusted entity should be protected using strong security measures (e.g., physical security, kernel security, etc.). In Iolus, however, there are many agents and all of the agents are trusted entities. Thus the level of trust required of such system components is much greater in Iolus than in our approach.

C. Reliability

In Iolus, agents are needed to securely forward message keys. When an agent fails, a backup is needed. It would appear that replicating a single key server (in our approach) to improve reliability is easier than backing up a large number of agents.[10]

VII. CONCLUSION

To address the scalability problem of group key management, we propose the use of key trees (or graphs, in general). We investigated three rekeying strategies, *user-oriented*, *key-oriented* and *group-oriented*, and specified join/leave protocols for them. The rekeying strategies and protocols are implemented in a prototype key server we have built. From measurement results of a large number of experiments, we conclude that our group key server using any of the three rekeying strategies is scalable to very large groups with frequent joins and leaves. In particular, the average server processing time per join/leave increases linearly with the logarithm of group size. We found that the optimal key tree degree is around four.

On the server side, group-oriented rekeying provides the best performance, with key-oriented rekeying in second place, and user-oriented rekeying in third place. On the client side, user-oriented rekeying provides the best performance, with key-oriented rekeying in second place, and group-oriented rekeying in third place. In particular, for a very large group whose clients are connected to the network via low-speed connections (modems), key-oriented or user-oriented rekeying would be more appropriate than group-oriented rekeying.

We have not investigated the amount of network traffic generated by the three rekeying strategies. With group-oriented rekeying, a single rekey message is sent per join/leave via multicast to the entire group. The network load generated would depend upon the network configuration (local area network, campus network, wide-area Internet, etc.) and the group's geographic distribution. With key-oriented and user-oriented rekeying, many smaller rekey messages are sent per join/leave to subgroups. If the rekey messages are sent via unicast (because the network provides no support for subgroup multicast), the network load generated would be much greater than that of group-oriented rekeying.

It is possible to enable subgroup multicast by the method in [16] or by allocating a large number of multicast addresses, one for each subgroup that shares a key in the key tree being used. A more practical approach, however, is to allocate just a small number of multicast addresses (e.g., one for each child of the key

---

[10] Craig Partridge observed that agents can be implemented in existing firewalls and derive their reliability and trustworthiness from those of firewalls.

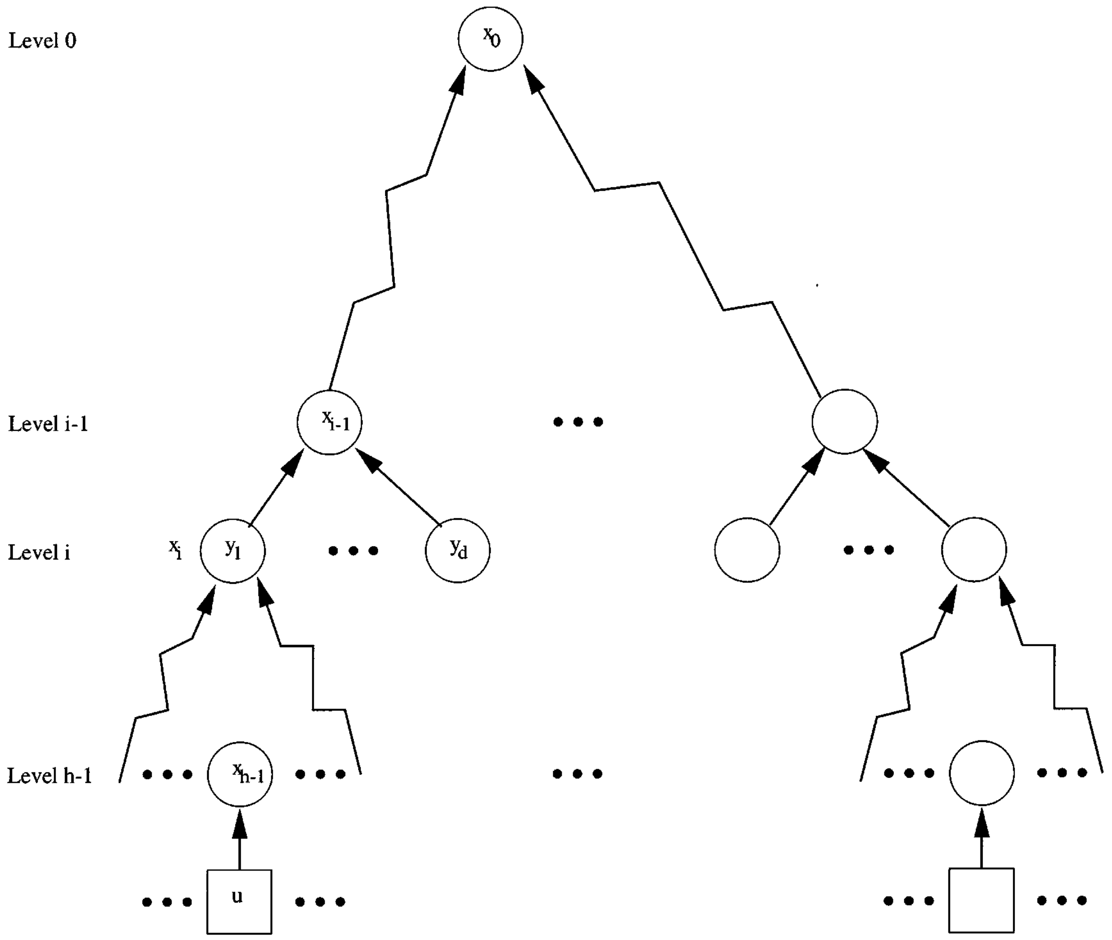


Fig. 15. Derivation of $\mathrm{Prob}[u$ changes i keys$]$.

tree's root node) and use a rekeying strategy that is a hybrid of group-oriented and key-oriented rekeying. It is straightforward to design such a hybrid strategy and specify the join/leave protocols. Furthermore a hybrid approach, involving the use of some Iolus agents at strategic locations, such as firewalls or border routers, would also be appropriate.

We have implemented a prototype system for group key management, called Keystone [15]. In designing the system, we had to deal with several practical issues not considered in this paper. To deliver rekey messages reliably to group members, an application program that uses Keystone can specify one of two options for rekey message delivery: reliable unicast (which is not scalable) or IP multicast with forward error correction. In the case of forward error correction, the application can specify the ratio of orginal to repair packets such that the loss probability of a rekey message is calculated to be at an acceptable level (such as 0.001). For rekey messages that are not recoverable, a client sends a resynchronization request to the Keystone server, which then resends the missing rekey messages by reliable unicast.

For applications characterized by very frequent joins/leaves, the amount of rekey message traffic can be substantially reduced by rekeying periodically instead of after each join/leave. This also allows batch processing of join/leave requests at the key server. An application that uses Keystone can specify a rekeying period. If one is specified, then the Keystone server performs rekeying periodically instead of after every join/leave. Any efficiency gain, however, is at the expense of allowing new members access to some past data and old members access to some future data. Such a tradeoff may be acceptable to some e-commerce applications, e.g., pay per view and information services, where a temporary breach in group confidentiality can be quantified in monetary terms.

Last, Keystone allows an application to have multiple secure groups, and clients to simultaneously join (or leave) a subset of these groups. For example, consider video conferencing where there is video stream and a set of audio streams in different languages. Each client joins (leaves) the video group and one audio group simultaneously. In this case, it is efficient for Keystone to use a single key graph for all groups instead of separate key trees (one for each group).

APPENDIX
NUMBER OF KEY CHANGES BY A USER

Consider a secure group with a key tree that is full and balanced with degree d and height h. Suppose each user is equally likely to be the one who is joining/leaving. First, we derive the probability that after a join or leave, a user, say, u, needs to change exactly i keys, denoted by $\mathrm{Prob}[u$ changes i keys$]$.

Suppose the individual key of user u is at k-node x_{h-1}. Let $x_{h-1}, \ldots, x_0$ denote the path from k-node x_{h-1} to the root x_0 of the tree (see Fig. 15). User u needs to change exactly i keys if and only if it needs to change keys at $x_{i-1}, \ldots, x_0$ but not $x_{h-1}, \ldots, x_i$. Let $y_1, \ldots, y_d$ denote the children of x_{i-1}. Without any loss of generality, we assume $y_1 = x_i$, that is, user u is in the subtree rooted at y_1. When the joining/leaving user is in one of the subtrees rooted at $y_2, \ldots, y_d$ (there are $d-1$ of them), the keys at $x_{i-1}, \ldots, x_0$ are precisely the ones to be changed by user u. Note that these subtrees are of the same height, and there are d^i subtrees of this height in the key tree. Therefore

$$\mathrm{Prob}[u \text{ changes } i \text{ keys}] = (d-1)/d^i.$$

For a join/leave, the average number of key changes (or the average number of key decryptions) by a nonrequesting user u, denoted by c_u, is given by the following expression:

$$\begin{aligned} c_u &= \sum_{i=1}^{h-1} i \times \mathrm{Prob}[u \text{ changes } i \text{ keys}] \\ &= \sum_{i=1}^{h-1} i \times (d-1)d^{-i} \\ &= \frac{d}{d-1} - \left(\frac{d}{d-1} + (h-1)\right) d^{-(h-1)} \\ &< \frac{d}{d-1} \end{aligned}$$

which depends only on the degree d of the key tree.

ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers and the editor, C. Partridge, for their constructive comments.

REFERENCES

[1] T. Ballardie, "Scalable multicast key distribution," *RFC 1949*, May 1996.

[2] T. Ballardie and J. Crowcroft, "Multicast-specific security threats and counter-measures," in *Proc. Symp. Network and Distributed System Security*, 1995.

[3] S. Berkovits, "How to broadcast a secret," in *Advances in Cryptology, EUROCRYPT'91*, D. W. Davies, Ed. Berlin, Germany: Springer Verlag, 1991, vol. 547, *Lecture Notes in Computer Science*, pp. 535–541.

[4] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "The kryptoknight family of light-weight protocols for authentication and key distribution," *IEEE/ACM Trans. Networking*, vol. 3, pp. 31–41, Feb. 1995.

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

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1989.

[7] S. E. Deering, "Multicast routing in internetworks and extended LANs," in *Proc. ACM SIGCOMM'88*, Aug. 1988, pp. 55–64.

[8] R. H. Deng, L. Gong, A. A. Lazar, and W. Wang, "Authenticated key distribution and secure broadcast using no conventional encryption: A unified approach based on block codes," in *Proc. IEEE Globecom'95*, Nov. 1995.

[9] A. Fiat and M. Naor, "Broadcast encryption," in *Advances in Cryptology, CRYPTO'93*, D. R. Stinson, Ed. Berlin, Germany: Springer Verlag, 1994, vol. 773, *Lecture Notes in Computer Science*, pp. 480–491.

[10] A. O. Freier, P. Karlton, and P. C. Kocher, The SSL Protocol Version 3.0, 1996. Work in progress, Netscape Communications.

[11] L. Gong, "Enclaves: Enabling secure collaboration over the internet," *IEEE J. Select. Areas Commun.*, pp. 567–575, Apr. 1997.

[12] H. Harney and C. Muckenhirn, "Group key management protocol (GKMP) architecture," *RFC 2094*, July 1997.

[13] ——, "Group key management protocol (GKMP) specification," *RFC 2093*, July 1997.

[14] J. B. Lacy, D. P. Mitchell, and W. M. Schell, "CryptoLib: Cryptography in software," in *Proc USENIX 4th UNIX Security Symp.*, Oct. 1993.

[15] Keystone: A Group Key Management Service, S. S. Lam and C. K. Wong. [Online]. Available: http://www.cs.utexas.edu/users/lam/NRL/

[16] B. N. Levine and J. J. Garcia-Luna-Aceves, "Improving internet multicast with routing labels," in *Proc. Int. Conf. Network Protocols*, 1997.

[17] R. C. Merkle, "A certified digital signature," in *Advances in Cryptology—CRYPTO'89*, 1989, pp. 241–250.

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

[19] B. C. Neuman, "Proxy-based authorization and accounting for distributed systems," in *Proc. 13th Int. Conf. Distributed Computing Systems*, May 1993, pp. 283–291.

[20] M. K. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in rampart," in *Proc. ACM Conf. Computer and Communications Security*, Nov. 1994, pp. 68–80.

[21] J. G. Steiner, C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems," in *Proc. USENIX Winter Conf.*, Feb. 1988, pp. 191–202.

[22] D. R. Stinson, "On some methods for unconditionally secure key distribution and broadcast encryption," *Designs, Codes Cryptography*, vol. 12, no. 3, pp. 215–243, 1997.

[23] J. J. Tardo and K. Alagappan, "SPX: Global authentication using public key certificates," in *Proc. 12th IEEE Symp. Research in Security and Privacy*, May 1991, pp. 232–244.

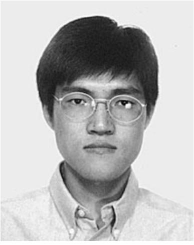
[24] D. M. Wallner, E. J. Harder, and R. C. Agee, "Key management for multicast: issues and architectures," *Informational RFC*, July 1997.

[25] C. K. Wong, M. Gouda, and S. S. Lam, "Secure Group Communications Using Key Graphs," Department of Computer Sciences, The Univ. of Texas at Austin, Tech. Rep. TR-97-23, July 1997.

[26] C. K. Wong and S. S. Lam, "Digital signatures for flows and multicasts," in *Proc. IEEE ICNP'98*, Oct. 1998. Revised version in *IEEE/ACM Trans. Networking*, vol. 7, pp. 502–513, Aug. 1999.

[27] T. Y. C. Woo, R. Bindignavle, S. Su, and S. S. Lam, "SNP: An interface for secure network programming," in *Proc. USENIX'94 Summer Technical Conf.*, Boston, MA, June 1994.

[28] T. Y. C. Woo and S. S. Lam, "Designing a distributed authorization service," in *Proc. IEEE INFOCOM'98*, San Francisco, CA, Mar. 1998, pp. 419–429.



Chung Kei Wong (S'88–M'00) received the B.Eng. degree from the University of Hong Kong, the M.Phil. degree from the Hong Kong University of Science and Technology, and the Ph.D. degree from the University of Texas at Austin.

He is currently a Research Staff Member at HRL Laboratories, Malibu, CA. His research interests include network security, multicast security, and multicast communication.

Mohamed G. Gouda (M'93) was born in Egypt. His received the B.Sc. degree in engineering and in mathematics from Cairo University, Cairo, Egypt. He received the M.A. degree in mathematics from York University, Toronto, Ont., Canada, and the master's and Ph.D. degrees in computer science from the University of Waterloo, Waterloo, Ont.

He worked for the Honeywell Corporate Technology Center in Minneapolis, MN, during 1977–1980. In 1980, he joined the University of Texas at Austin, where he currently holds the Mike A. Myers Centennial Professorship in Computer Sciences. He spent one summer at Bell Labs in Murray Hill, NJ, one summer at MCC in Austin, TX, and one winter at the Eindhoven Technical University in the Netherlands. His research area is distributed and concurrent computing. In this area, he has been working on abstraction, formality, correctness, nondeterminism, atomicity, convergence, stabilization, and efficiency. He has published more than 60 journal papers more than 100 conference papers. He was the founding Editor-in-Chief of the Springer-Verlag journal *Distributed Computing* during 1985–1989. He is the author of *Elements of Network Protocol Design* (New York: Wiley, 1998), the first ever textbook where network protocols are presented in abstract and formal setting.

Prof. Gouda received the Kuwait Award in Basic Sciences in 1993. He was the Program Committee Chairman of the ACM SIGCOMM Symposium in 1989. He was the first Program Committee Chairman of the IEEE International Conference on Network Protocols in 1993 and of the IEEE Symposium on Advances in Computers and Communications, which was held in Egypt in 1995. He was the Program Committee Chairman of IEEE International Conference on Distributed Computing Systems in 1999. He is on the steering committee of the IEEE International Conference on Network protocols and is an original member of the Austin Tuesday Afternoon Club.

Simon S. Lam (S'71–M'74–SM'80–F'85) received the B.S.E.E. degree (with distinction) from Washington State University, Pullman, in 1969 and the M.S. and Ph.D. degrees in engineering from the University of California at Los Angeles (UCLA) in 1970 and 1974, respectively.

From 1971 to 1974, he was a Postgraduate Research Engineer at the ARPA Network Measurement Center, UCLA, where he worked on satellite and radio packet switching networks. From 1974 to 1977, he was a Research Staff Member at the IBM T.J. Watson Research Center, Yorktown Heights, NY. Since 1977, he has been on the Faculty of the University of Texas at Austin, where he is a Professor of computer sciences. He holds two anonymously endowed professorships and served as Department Chair from 1992 to 1994. His research interests in networking include protocol and switch design, performance analysis, distributed multimedia, quality-of-service guarantees, and security.

Dr. Lam received the 1975 Leonard G. Abraham Prize Paper Award from the IEEE Communications Society for his paper on packet switching in a multiacess broadcast channel. He is a Fellow of the Association for Computing Machinery (ACM). He has served on the editorial boards of IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE TRANSACTIONS ON COMMUNICATIONS, PROCEEDINGS OF THE IEEE, and *Performance Evaluation*. He was Editor-in-Chief of IEEE/ACM IEEE/ACM TRANSACTIONS ON NETWORKING from 1995 to 1999. He organized and was Program Chair of the inaugural ACM SIGCOMM Symposium held at the University of Texas at Austin in 1983. He is a founding Steering Committee member of the IEEE International Conference on Network Protocols.