Dynamic Join-Exit Amortization and Scheduling for Time-Efficient Group Key Agreement

Yinian Mao, Yan Sun, Min Wu and K. J. Ray Liu
Department of Electrical and Computer Engineering
University of Maryland, College Park
Email: {ymao, ysun, minwu, kjrliu}@eng.umd.edu

***Abstract*—In this paper, we propose a time-efficient contributory key agreement framework for secure communications in dynamic groups. The proposed scheme employs a special join-tree/exit-tree topology in the logical key tree and effectively exploits the efficiency of amortized operations. We derive the optimal parameters and design an activation algorithm for the join and exit trees. We also show that the asymptotic average time cost per user join and leave event is $\Theta(\log(\log n))$, where n is the group size. Our experiment results on both simulated user activities and the real MBone data have shown that the proposed scheme outperforms the existing tree-based schemes.**

I. Introduction

The recent development in multicast communications has led to the deployment of many group-oriented applications [1], such as video conferencing, network games and data-collection in sensor networks [2]. In many group communication systems, communication security is an indispensable component [3]. One important aspect of group communication security is access control [4], which can be achieved by encrypting the communication content using a secret key known to all group members [5] [6]. Such a key is usually referred to as the *group key* [7] [8] [9].

To address the access control issue in group communications, many group key management protocols have been proposed [4] [7] [8] [10]–[22]. These schemes can be classified into two categories, centralized schemes [10] [7] [11], where a central key server is responsible for generating and distributing keys to all group members, and contributory schemes [12]–[14], where each group member contributes his/her own share to the group key. There are also key management schemes that emphasize the underlying communications technology, such as the Internet [18] [5] or wireless networks [15].

To establish and update a group key in a large dynamic group often require a considerable amount of effort. Consider a group of n users. Upon each user's join, the group key needs to be updated to prevent the joining user from accessing the past communications. Similarly, upon each user's departure, the group key needs to be updated to prevent the leaving user from accessing the future communications. The communication and computation overhead associated with key update is related to the group size n. When the group is large and the *join* and *leave* events are frequent, the key updates will incur a significant overhead for the entire group.

The scalability of a group key agreement in large and dynamic groups is one of the main concerns in the design of group key management protocols [23]. A number of existing works have addressed this problem both from theoretical points of view [24] [25], and from construction perspectives [8], [16]–[22], [26]. Previous literatures have also shown that, in group key management schemes, a logical *key tree* can be employed to organize the keys belonging to the group users [7] [13]. The usage of a logical key tree reduces the communications complexity associated with join and leave events to $\Theta(\log n)$, where n is the group size.

Since the two-party Diffie-Hellman (DH) protocol [27] was published in 1976, many contributory key agreements that extend the two-party DH to group scenarios have been proposed [13] [14] [28] [29]. In some applications, contributory key agreements are particularly attractive due to three reasons. First, it does not require the existence of a secure communication channel. Second, it does not put all trust in a third party (a key server) that generates and distributes keys for group members. Third, it does not need the establishment of a key server, which could be infeasible in some practical situations [13]. While the communication and computation complexity of contributory key agreements have drawn extensive attention [13] [14] [24] [25], the discussion on the time-efficiency issues of contributory schemes remains limited. Furthermore, cryptographic primitives of a contributory key agreement, such as modular multiplication and exponentiation [30], are computationally more expensive than their centralized counterparts [5], which poses a time-efficiency challenge to contributory key agreements.

In this paper, we investigate time efficiency issues of contributory key agreements. We first analyze the importance of time efficiency in contributory key agreements and propose two performance metrics for tree-based contributory schemes. To improve the time efficiency, we design a novel key tree topology with join and exit subtrees. Together with this key tree topology, we propose a set of algorithms to handle user join and leave events. We then integrate all the algorithms into a Dynamic Subtree Group Key Agreement. The proposed scheme employs amortization and scheduling techniques to improve the time efficiency in large dynamic groups. Our analytical results show that the proposed scheme achieves an asymptotic average time of $\Theta(\log(\log n))$ for a join event, and also $\Theta(\log(\log n))$ for a leave event when group dynamics are known *a priori*. In addition to the improved time efficiency, our scheme has low communications and computation complexity.

The rest of this paper is organized as follows. Section II reviews tree-based contributory key agreement and proposes two performance metrics for time efficiency. Section III discusses the join and exit tree topology and algorithms used in our scheme. In Section IV, we integrate these algorithms into a unified protocol. We present the simulation results in Section V and discuss several other performance aspects in Section VI. Finally, the conclusions are drawn in Section VII.

II. Background and Performance Metrics

In this section, we discuss the time efficiency issues of contributory key agreement and review a class of tree-based contributory key agreements. The performance metrics that measure the time efficiency are also formulated.

A. *Time-Efficiency Issues in Contributory Key Agreements*

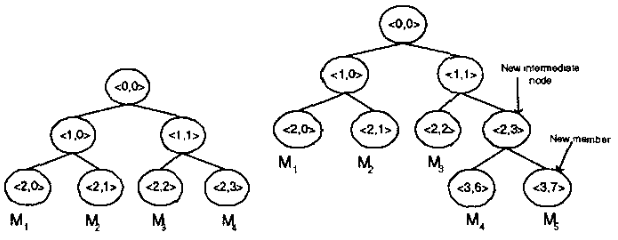
Time efficiency of contributory key agreements describes the processing time of key updating due to users' join and departure. After sending the join request, a join user has to wait until group keys are updated before being able to participate in the group communication. Since both computing cryptographic primitives and exchanging messages for a key update are time-involving, this waiting time is not negligible. Similarly, in the case of user departure, the amount of time needed to recompute a new group key reflects the latency in user revocation. In applications with large group size and highly dynamic membership, the time efficiency of the key management is an important quality-of-service concern.

Many contributory key agreements aim at extending the two-party DH protocol to the group scenario, such as [13], [14], [28], [29]. These schemes evaluate their time performance by the number of rounds needed to perform the protocol. In general, the number of rounds cannot always accurately reflect the time cost, especially when different rounds represent different operations. For example, in GDH.2 [29], one modular exponentiation is performed in the first round, while n modular exponentiations are performed in the n-th round. In this work, we will focus on the tree-based contributory schemes using DH, each round of which is to perform a two-party DH. In this scenario, we can use round as the basic time unit.

B. *Key Establishment and Update in Tree-based Contributory Schemes*

In this part, we briefly review rekeying operations for join and leave events in tree-based contributory key agreements [13], [14], [20], which can use two-party DH protocol as a basic module. These schemes satisfy the security requirements for group key distribution, namely, group key secrecy, forward secrecy, backward secrecy and key independence as defined in [13] and [8].

In a tree-based key agreement, three types of keys are organized in a logical key tree, as illustrated in Fig.1(a). The leaf nodes in a key tree represent the private keys held by group members. The root of the tree corresponds to the group



(a) a key tree (b) user join

Fig. 1. Notations for a key tree

key, which is held by all members in the group. All other inner nodes represent subgroup keys, each of which is held by the group members that are descendants of the corresponding inner node. We adopt the notations from [13] as follows:

M_i	i-th group member
$\langle l,v\rangle$	v-th node at level l in a key tree
$K_{\langle l,v\rangle}$	the key associated with the node $\langle l,v\rangle$
g	exponentiation base
p	modular base

To establish a group key, the keys in the key tree are computed in a bottom-up fashion. Users are first grouped into pairs and each pair performs a two-party DH to form a sub-group. These sub-groups will again pair up with each other and perform the two-party DH to form larger sub-groups. Continuing in this way, the final group key can be obtained. As an example shown in Fig.1(a), there are four members in the group. Denoting each member's private key as r_i, the group key $K_{\langle 0,0\rangle}$ is computed in 2 rounds as

$$K_{\langle 0,0\rangle} = g^{(g^{r_1 r_2} \bmod p)(g^{r_3 r_4} \bmod p)} \bmod p.$$

In a user join event, the new user will first be paired with an insertion node (which may represent a group of users) to perform a two-party DH. Then all the keys on the path from the insertion node to the tree root are updated recursively. An example is shown in Fig.1(b).

Upon a user's departure, the leaving user's node and its parent node will be deleted from the key tree. Its sibling node will assume the position of its parent node. Then all the keys on the path from the leaving user's grandparent node to the tree root are recalculated from the bottom to the top.

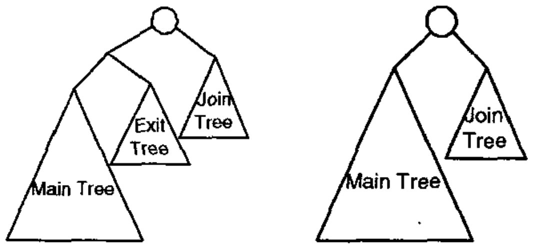
C. *Time Complexity Performance Metrics*

In this part we define two time complexity performance metrics.

1) Average User Join/Leave Time Metric: The *user join time* is defined as the number of rounds to process key updates for a user join event. The average user join time, denoted by ATC_{join}, is calculated as

$$ATC_{join} = \frac{R_{join}}{N_{join}}, \tag{1}$$

where R_{join} is the total number of DH rounds performed for N_{join} join events.



(a) join and exit tree (b) join tree only

Fig. 2. Join Exit Tree Topology

Similarly, the *user leave time* is defined as the number of rounds to process key updates for a user leave event. The average user leave time, denoted by ATC_{leave}, is calculated as

$$ATC_{leave} = \frac{R_{leave}}{N_{leave}}, \qquad (2)$$

where R_{leave} is the total number of DH rounds performed for N_{leave} leave events.

2) User Join/Leave Latency Metric: We define *user join latency* as the number of DH rounds needed for a joining user to acquire a group key, and *user leave latency* as the number of DH rounds needed to calculate a new group key that is unknown to the leaving user. The average user join and leave latency are denoted as AL_{join} and AL_{leave}, respectively.

In many existing key agreements [7], [13], [14], [28], [29], the user join time and latency are always the same. So does the user leave time and latency. In this paper, we present a contributory key agreement that aims at reducing average user join/leave time, while achieving a user join/leave latency that is even lower than the corresponding user join/leave time.

III. JOIN-EXIT TREE: THE TOPOLOGY AND ALGORITHMS

In this section, we present a logical key tree topology that consists of three parts: *join tree, exit tree,* and *main tree*, as shown in Fig.2(a). Similar to the key trees shown in [13] and [14], our proposed key tree is a binary tree. We define *join tree capacity* and *exit tree capacity* as the maximum number of users that can be held in the join and exit tree, respectively. Using the join-exit tree structure, we discuss how to choose the join and exit tree capacity dynamically such that the average user join and leave time are minimized.

The join tree and exit tree are designed to be considerably smaller than the main tree. The joining users will first be added to the join tree. Later on, when the join tree reaches its capacity, all users in the join tree will be relocated together into the main tree. In addition, when users' departure time is known, users that are likely to leave in the near future will be moved in batch from the main tree to the exit tree. The join-exit tree design rationale resembles that of memory hierarchy in computer design [31]. The join tree and exit tree are similar to the cache, and the main tree is similar to the main memory.

The join-exit tree topology can be reduced to a simpler form. For example, when there is no user in the exit tree,

TABLE I
JOIN-EXIT TREE NOTATIONS

N_M	number of users in main tree
C_J	join tree capacity
C_E	exit tree capacity
ATC_{join}	average join time cost(in rounds)
ATC_{leave}	average leave time cost(in rounds)

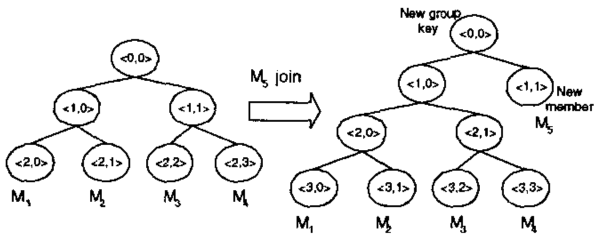


Fig. 3. User join at join tree root

the topology reduces to a main tree and join tree topology as shown in Fig.2(b). To distinguish the proposed key tree topology from those described in the existing schemes [13] [14], we call the key tree in Fig.2(a) a *join-exit tree* and a key tree without special structures a *simple key tree*. We specify the notations related to a join-exit tree in Table I and present the detailed join-exit tree algorithms.

A. The Join Tree Algorithm

We choose the average user join time as performance metric, and address the following four problems in the join tree algorithm:

1. If a join tree is used, where in the join tree do we insert the joining user?

2. When the join tree is full, how do we relocate users from the join tree to the main tree?

3. What is the optimal join tree capacity?

4. When should we choose to use a join tree?

1) Insertion Strategy for Joining Users: When the join tree is empty and a new user wants to join, the insertion node is chosen as the root of the current key tree. The insertion is done by treating the entire existing group as one logical user, and performing a two-party DH between this logical user and the new user. Therefore the new user forms the root of join tree. This process is shown in Fig.3. When there are already some users in the join tree, the insertion node is determined by the Algorithm 1, where $usernumber(x)$ returns the number of users under a given node x in the key tree.

Algorithm 1 Finding the insertion node
```
$x \leftarrow$ join-tree-root
while $usernumber(x) \neq 2^k$ for some integer $k$ do
    $x \leftarrow rightchild(x)$
end while
insertion-node $\leftarrow x$
```

Fig.4 shows the growth of the join tree from 1 user to 8 users using the proposed insertion strategy.

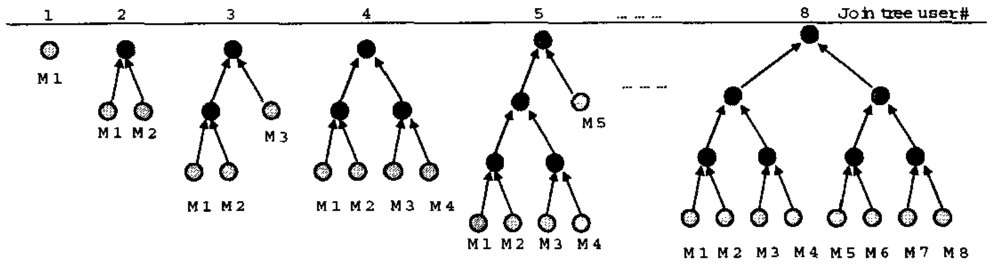


Fig. 4. Sequential User Join Strategy (only the join tree is shown)

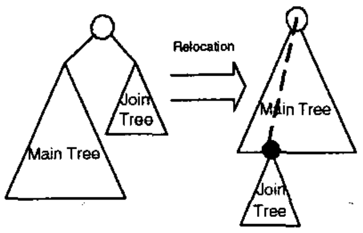


Fig. 5. Join Tree Relocation Method 1

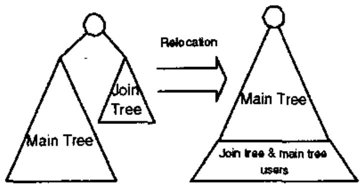


Fig. 6. Join Tree Relocation Method 2

2) *Relocation Strategy:* When the join tree is full, users in the join tree will be relocated into the main tree. Relocation can be done in two ways with different tradeoffs. The difference of these two methods is whether to preserve the sub-group keys in the original join tree.

The first method is illustrated in Fig.5. During relocation, the subgroup keys among the users in the join tree are preserved. Hence the join tree structure is also preserved. All users in the join tree are viewed as a logical user and this logical user is inserted into the main tree. An insertion node is chosen to be the leaf node on the shortest branch in the main tree, shown as the black node in Fig.5. Then all keys along the path (shown as a dashed line in Fig.5) from the insertion node to the tree root are updated.

The second relocation method is illustrated in Fig.6. This method inserts the join tree users into different nodes in the main tree. The insertion nodes are chosen to be the leaf nodes in the shortest branches. After the insertion nodes are found, a new group key is computed in a bottom-up fashion. The keys on the branches from all original join tree users to the tree root are updated.

The relocation time for the first and second method is at most $\log N_M$ and $\log N_M + 1$, respectively. The first method has a lower communication cost. Only $2 \log N_M$ messages in total are sent during relocation key update. The second method

TABLE II
SEQUENTIAL USER JOIN LATENCY

k	1	2	3	4	5	6	7	8	9	10	...
$r(k)$	1	2	2	3	2	3	3	4	2	3	...

helps to maintain the balance of the key tree, which reduces the expected cost of leave events [13]. Because the second method addresses both the join and leave time cost, we choose the second method for our analysis and simulations.

3) *Optimal Join Tree Capacity:* Using the proposed insertion strategy, the user join latency for the k-th user in the join tree after the last join tree relocation is measured as $r(k)$ rounds, which is listed in Table II. We observe that the sequence of $r(k)$ has a special property, namely,

$$r(2^p + q) = 1 + r(q), \qquad 0 < q \le 2^p, \tag{3}$$

where p is a non-negative integer, and q a positive integer.

Lemma 1. If the user join latency $r(k)$ for the k-th user on the join tree is determined by (3), then

$$\frac{1}{C_J}\sum_{k=1}^{C_J} r(k) \le \frac{1}{2}\log C_J + 1 \tag{4}$$

holds for any positive integer C_J, and equality is achieved when C_J is a power of 2.

Proof: See appendix.

Consider the average join time for C_J users joining the group after the last join tree relocation. Counting the relocation time of $\log N_M$, the average join time for these C_J users is

$$ATC_{join} = \frac{1}{C_J}\left(\sum_{k=1}^{C_J} r(k) + \log N_M\right). \tag{5}$$

Using Lemma 1, we obtain

$$ATC_{join} \le \frac{1}{2}\log C_J + 1 + \frac{1}{C_J}\log N_M. \tag{6}$$

Since it is not easy to minimize ATC_{join} directly, we minimize its upper bound over C_J. The optimal C_J value is given by

$$\begin{aligned} C_J^{opt} &= \mathbf{argmin}_{x>0}\left\{\frac{1}{2}\log x + 1 + \frac{1}{x}\log N_M\right\} \\ &= 2\ln N_M \end{aligned} \tag{7}$$

This analysis leads to the following theorem:

Theorem 1. For a given main tree user number N_M and the insertion rule specified by Algorithm 1, the optimal join tree capacity C_J is $2\ln N_M$, and the average join time during two join tree relocations is upper bounded by

$$ATC_{join} \leq \frac{1}{2}\log\log N_M + \frac{3}{2} + \frac{1}{2\ln 2} - \frac{1}{2}\log\log e. \quad (8)$$

Proof: Directly from (5)-(7).

The join tree capacity is thus determined by the number of users in the main tree. This relation gives us an upper bound on the average user join time cost. However, since users can start to communicate once they are added in the join tree, the user join latency does not include the relocation overhead of $\log N_M$ rounds. When C_J is equal to $2\ln N_M$, the maximum join latency is $\log C_J = \log\ln N_M + 1$ and the average join latency is bounded by

$$AL_{join} \leq \frac{1}{2}\log(\log N_M) + \frac{3}{2} - \frac{1}{2}\log\log e.$$

4) Activation Condition for Join Tree: We now discuss a condition under which a reduction in average join time can be achieved by using join tree. We call this condition *the activation condition* for join tree. Suppose all users joining the group will first be added to the join tree. Consider the case when users join one by one and assume that the join tree and the main tree are balanced. In the worst case, adding each user in the join tree incurs a time cost of $\log C_J$ rounds, and a batch relocation incurs an additional time cost of $\log N_M$ rounds for C_J users. So the average join time satisfies

$$ATC_{join} \leq \log C_J + (\log N_M)/C_J. \quad (9)$$

In the same situation, if a simple key tree with only a main tree is used, the average join time would be $\log N_M$. Therefore a reduction in time cost can be obtained by using a join tree if the following inequality holds,

$$\log C_J + (\log N_M)/C_J \leq \log N_M,$$

or equivalently,

$$\log N_M \geq \frac{C_J}{C_J - 1}\log C_J. \quad (10)$$

This condition tells us when the number of users in the group is large enough, a join tree should be activated to reduce the average join time. We can show that there exists a threshold group size, TH_{join}, such that all N_M values larger than TH_{join} can satisfy (10). Therefore when the group size is smaller than or equal to TH_{join}, a simple key tree is used. Otherwise, a join tree is activated.

Example

$$N_M = 9, \quad C_J = 2\ln N_M \approx 4;$$
$$\log N_M \approx 3.2, \quad \frac{C_J}{C_J - 1}\log C_J \approx 2.3.$$

This N_M value satisfies (10). Therefore TH_{join} can be set to 9.

TABLE III
NOTATIONS FOR BATCH MOVEMENT

B	batch movement size
ρ	exit tree residual rate
U_p	user number in exit tree right after the last batch movement
U_c	current number of users in exit tree

B. The Exit Tree Algorithm

The join tree algorithm employs scheduling and amortization techniques. Scheduling user departure, however, is a harder task, because there is no simple way to accurately predict user's departure time and location in the key tree. We assume that when users join the group communication, most of them can have a self-estimated departure time. In the following analysis, we show that with perfect user departure information and the use of exit tree, the average user departure time can be reduced to $\Theta(\log(\log n))$, where n is the group size. Later in the simulations, we also show that a reduction in average departure time can be obtained when the estimated departure time deviates from the actual departure time.

In this part, we first present a batch movement operation, followed by the analysis on optimal exit tree capacity. Finally we discuss the activation condition for exit tree.

1) Batch Movement: The *batch movement* refers to the operations to move the potential leaving users from the main tree to the exit tree. During the batch movement, a series of key updates are performed and a new group key is computed. The batch movement does not affect the group communications since the old group key can still be used without violating any security requirement. And the new group key becomes effective upon the completion of its computation.

When a new user joins the group, he/she will report a self-estimated departure time. The whole group maintains a *leaving queue*, which is a priority queue [32] indexed by users' estimated departure time. Before each join tree relocation, the departure information of the join tree users are added to the leaving queue.

With a user's departure, the leaving queue and a condition for batch movement (to be presented below) are checked. If the leaving user is in the leaving queue, his/her item will be removed from the leaving queue. If the batch movement condition is met, the first B users in the leaving queue will be moved to the exit tree in batch, where B is referred to as the *batch movement size*. The insertion locations for these users in the exit tree are chosen to maintain the balance of the exit tree. In Table III we introduce batch movement notations.

Our proposed batch movement condition is

$$U_c \leq \rho U_p, \quad (11)$$

where we use the exit tree residual rate (or residual rate for short), $\rho \in (0, 1)$, as well as U_p and U_c, to control the timing of batch movement. Using this condition, if we start from an empty exit tree ($U_p = 0$), the number of users in the exit tree after the k-th batch movement will be $\sum_{i=0}^{k-1}\rho^i B$, which will converge to $B/(1-\rho)$ as k goes to infinity. Therefore we set the exit tree capacity C_E as

$$C_E = B/(1-\rho). \tag{12}$$

2) Optimal Exit Tree Capacity: In deriving the optimal exit tree capacity, we minimize an upper bound of the average leaving time over the exit tree capacity. This upper bound for the average leaving time is not as tight as that for the average join time because of the randomness in users' departure.

A batch movement of B users to the exit tree will incur a time cost of $(\log N_M + 2)$. Each user leaving from the exit tree will incur at most a time cost of $(\log C_E + 2)$. Thus the average user leave time for these B users is bounded by

$$ATC_{leave} \leq \frac{1}{B}(\log N_M + 2) + (\log C_E + 2).$$

Using (12), we can rewrite it as

$$ATC_{leave} \leq \frac{1}{(1-\rho)C_E}(\log N_M + 2) + (\log C_E + 2). \tag{13}$$

Minimizing the right hand side of (13), we obtain

$$\begin{aligned} C_E^{opt} &= \mathbf{argmin}_x \left\{ \frac{1}{(1-\rho)x}(\log N_M + 2) + (\log x + 2) \right\} \\ &= \frac{\ln N_M + 2\ln 2}{(1-\rho)}. \end{aligned} \tag{14}$$

Therefore when exit tree is activated and its capacity is computed according to (14), the average leaving time is bounded by

$$ATC_{leave} \leq \log(\log N_M + 2) + \delta,$$

where $\delta = 2 - \log(1-\rho) + \log e - \log\log e$. Combining (14) and (12) leads to the optimal batch movement size

$$B^{opt} = \ln N_M + 2\ln 2. \tag{15}$$

In summary, the exit tree capacity is chosen as

$$C_E = \begin{cases} 0 & \text{if no exit tree used;} \\ (\ln N_M + 2\ln 2)/(1-\rho) & \text{otherwise.} \end{cases}$$

3) Activation Condition for Exit Tree: Recall that the average leaving time using a simple key tree with N_M users is $\log N_M$. Compared with (13), a reduction in the average leaving time can be achieved by the proposed exit tree strategy if

$$\frac{1}{(1-\rho)C_E}(\log N_M + 2) + (\log C_E + 2) \leq \log N_M,$$

or equivalently,

$$C_E \geq \frac{\log N_M + 2}{(1-\rho)(\log N_M - \log C_E - 2)}. \tag{16}$$

Combining (14) and (16), we have the activation condition as

$$\log N_M \geq \log C_E + \log e + 2. \tag{17}$$

Condition (17) indicates that, when the group size is large enough, employing an exit tree can reduce the average leaving time. Similar to the join tree case, we can show that there is a threshold group size, TH_{leave}, such that all N_M values larger than TH_{leave} can satisfy (17). Only when the group size is larger than TH_{leave}, the exit tree is activated. We also notice that the join tree is activated before the exit tree is, because satisfying (17) requires a larger N_M than satisfying (10).

Example

$$N_M = 256, \quad C_E = 7;$$
$$\log C_E \approx 2.8, \quad \log N_M - \log e - 2 \approx 4.6.$$

This N_M value satisfies (17). Therefore TH_{leave} can be set to 256.

IV. Dynamic Subtree Group Key Agreement

In this section we present a contributory group key agreement that jointly use the join and exit tree. Based on the results in Section III, the join and exit tree capacities are adjusted according to the group size. So we name it Dynamic SubTree (DST) group key agreement.

A. Group Key Establishment

In prior works, one of the assumptions in key establishment stage is that many users are available at the same time [28] [20]. Thus parallel computation can take place to establish a group key [20]. In reality, however, there are situations when users join the group sequentially, and early arrival users are not necessary to wait for all users to be present.

In DST scheme, when many users are present at the same time, subgroup keys in the key tree are computed in a bottom-up fashion in parallel to obtain the final group key. This technique is also described in [20]. Otherwise we establish and update the group key using the join protocol (discussed below) of DST agreement. The exit tree will not be activated during the key establishment stage.

B. Join Protocol

The threshold group size for join tree activation is set to $TH_{join} = 9$. Key update for a user join event follows the next four steps, as illustrated in Fig.3:

1. Choose an insertion node in the key tree;

2. Generate a new inner node to assume the position of the insertion node;

3. The insertion node and the new member become children of the new inner node;

4. Update all the keys associated with the nodes on the path from the new inner node to the root.

Before the join tree is activated, Algorithm 1 is used in the simple key tree to choose the insertion node. When the group size is larger than 9, the join tree is activated. The join tree capacity C_J is computed according to (7), and rounded to the nearest integer. If inserting the new user according to Algorithm 1 will not make the join tree height more than $\lceil \log N_M \rceil$, the insertion strategy is followed. Otherwise, the insertion node will be chosen as the minimum level leaf node in the join tree. This modification takes user departure from the join tree into consideration, and helps make the join tree balanced.

When the join tree becomes full, following the corresponding algorithms in Section III, all users in the join tree will be relocated into the main tree, and their departure information is put into the leaving queue. After the relocation, the join and exit tree capacities (if exit tree is activated) are updated according to (7) and (14), respectively.

C. Leave Protocol

The threshold group size for exit tree activation is set to $TH_{leave} = 256$. The exit tree residual rate is set to $\rho = 0.5$. Key update for a leave event follows the next four steps:

1. delete the leaving user node and its parent node,
2. promote the leaving user's sibling node to their parent node's position,
3. update all keys associated with the nodes on the path from the leaving user's grandparent node to the tree root.
4. if the leaving user's information is in the leaving queue, remove the corresponding information.

In addition to the above four steps, if a user is leaving from the main tree or the exit tree, the following extra operations are necessary.

When the user is leaving from the main tree and there are also users in the join tree, the key update for user relocation and user departure are performed together. By doing so the time cost for user relocation is further amortized. After the key update, the join tree capacity is updated according to (7). And the exit tree capacity is also updated if the value computed from (14) becomes larger than the current number of users in the exit tree.

When the user is leaving from the exit tree and the batch movement condition is satisfied, a batch movement will be performed according to the batch movement strategy in Section III. Following the batch movement, the join and exit tree capacity are updated in the same way as described in the last paragraph.

In practice, when the number of users in a group is always around TH_{leave}, using the previous activation condition will lead to repeated switching of the key tree topology, thus incurring a considerable overhead. To stabilize the key tree topology, we propose a delayed switching policy. The leave tree is activated when $N_M \geq 2TH_{leave}$ and deactivated when $N_M < TH_{leave}$. This will improve the stability of the key tree.

V. Experiments and Performance Analysis

In this section, we present three sets of simulations according to the ways user activity data are acquired. The first set of simulations focuses on group key establishment. We consider the scenario of sequential user join. The second set of simulations is based on user activity data collected from previous MBone multicast sessions [33]. The third set of simulations shows the results for a large dynamic group, whose user activity data are randomly generated according to a probabilistic model. In each simulation, the performance of our proposed scheme is compared with TGDH scheme [13], a typical of tree-based key agreement.

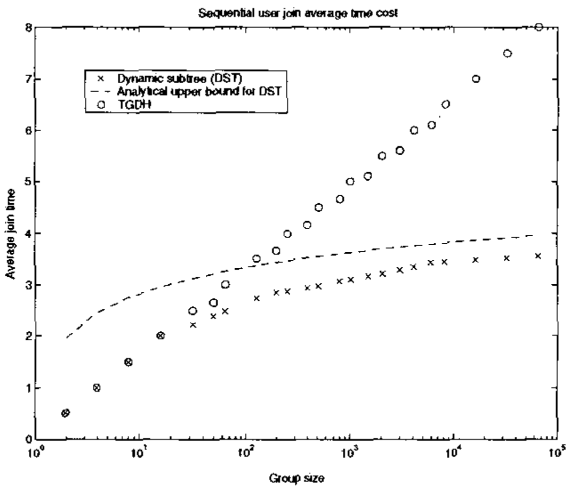


Fig. 7. Sequential User Join Average Time Cost

A. Sequential User Join Key Establishment

For sequential user join, the proposed DST protocol uses a simple key tree for small group size, and activate the join tree when the group size is larger than 9.

In Fig.7, we plot the average user join time for sequential user join using TGDH [13] and the proposed DST. The x-axis indicates the group size, and the y-axis indicates the average user join time for the corresponding group size. It can be seen that our dynamic subtree scheme achieves the same performance as TGDH when the group size is small, and outperforms TGDH scheme when the group size becomes large. From the figure we can see that, when a large group of users is joining the communication group, TGDH achieves an average time cost of $\Theta(\log N)$, and the proposed DST scheme achieves an asymptotic performance of $\Theta(\log(\log N))$. The dashed line is the theoretic upper bound for the average time cost of sequential user join from (8).

B. MBone Data Experiment

From the study of Multicast Backbone (MBone) multicast sessions, Ammeroth *et al.* observed that the MBone multicast group size is usually small (typically 100-200), and users either stay in the group for a short period of time or a very long time [33] [34].

Using our proposed DST scheme, an exit tree will not be activated for a group size smaller than TH_{leave}. However, when a user stays in the group for only a short period of time, it is highly possible that this user joins the group in the join tree and leaves from the join tree without getting to the main tree. This analysis indicates that our proposed DST scheme should outperform the existing tree-based schemes for the same user activities in MBone multicast sessions.

We choose three user activity log files from three MBone multicast sessions [35]. Two of these three sessions are NASA space shuttle coverage and the other one is CBC News World online test. The user activities can be shown using a plot along the time line (in minutes), where $N(t)$ is the current number

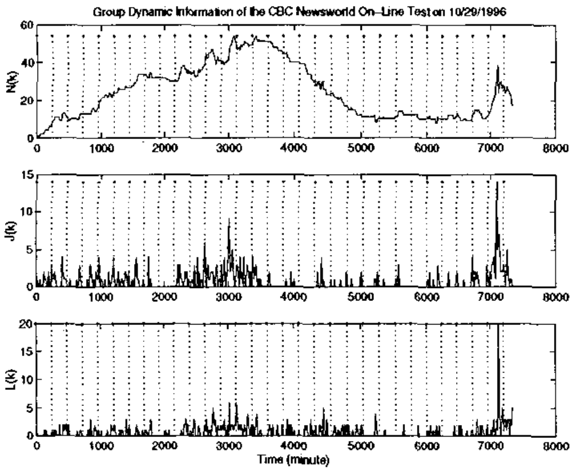


Fig. 8. MBone Session User Activity: CBC News World

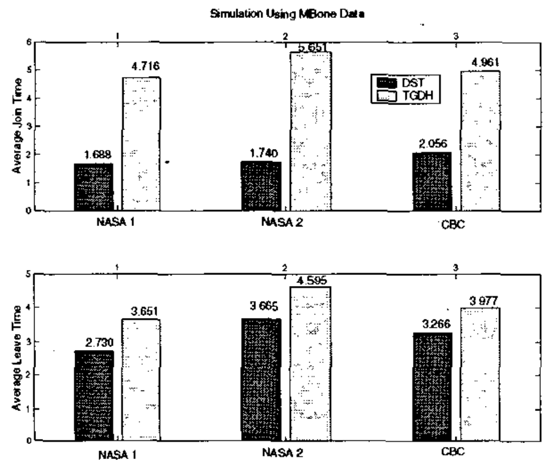


Fig. 9. Simulation Using MBone Data

of users in the multicast group, $J(t)$ is the number of users joining the group at this moment, and $L(t)$ is the number of users currently leaving the group. These log files serve as the user activity input for DST protocol simulation. Comparing the simulation results of the average time cost for our DST protocol and for TGDH in Fig.9, we can see that our proposed DST scheme has about 50% improvement in user join, and about 20% improvement in user leave.

C. Simulated Data Experiment

In the simulated data experiment, we generate user activities according to a probabilistic model. The duration of our simulation is 5000 time units and is divided into four non-overlapping segments, T_1 to T_4. In each time segment T_i, users' arrival time is a Poisson process with mean arrival rate λ_i and users' staying time follows an exponential distribution with mean value μ_i. The Poisson arrival and exponential staying time are suggested in [33]. The values of λ_i and μ_i are listed in Table IV.

The group size is initialized to be 0. In a total of 5000

TABLE IV
STATISTICAL PARAMETERS FOR USER BAHAVIOR

duration	0-199	200-499	500-4499	4500-5000
λ_i	7	5	2	1
μ_i	2500	500	500	500
	long stay	short stay		

TABLE V
SIMULATED DATA EXPERIMENT COMPARISON

	average		worst case	
	join	leave	join	leave
DST	2.766	7.083	14	14
TGDH	10.825	9.956	12	12

time units, there are 12000 user join events and 10983 user leave events. The maximum group size at any time is about 2800 and the group size at the end of simulation is about 1100. Each user's arrival time is known. And we assume the estimated staying time for each user follows Gaussian distribution $N(\mu_i, \sigma_i^2)$ with the mean value μ_i being the actual staying time for user i, and the standard deviation σ_i proportional to the mean. Let R be the ratio of the standard deviation to the mean, *i.e.*, $R = \sigma_i/\mu_i$. We fix R for all users and simulate the average join and leave time for different R values in the range of [0,1]. Because of the Gaussian distribution in the estimated staying time, a user can potentially report a negative staying time. Such a user will not be moved to the exit tree.

The simulation results in Fig.10 show that, when the standard deviation is two orders of magnitude smaller than the true staying time, the proposed DST scheme can efficiently schedule both user join and leave events. As R increases, the average processing time per event (join and leave) remains unchanged for small R and increase by a small amount for large R. When R value is small, we have more accurate estimation of users' staying time, and by our protocol, most users are moved to the exit tree and leave the group from the exit tree. When R becomes large, two situations may occur and result in an increased operation time. A leaving user may not be moved to the exit tree because of the inaccuracy in the estimated staying time, or some users may be inappropriately moved to the exit tree and saturate the exit tree capacity, preventing other users from entering the exit tree. In both situations, more users tend to leave the group from the main tree, increasing the overall average processing time. However, since our protocol tries to combine the handling of user departure from the main tree with the join tree relocation and counts the time cost of such a combined event as user leave time, the average user join time so accounted may decrease. Alternatively, the time cost of the combined event can be counted toward the join cost, in which case the average user leave time would be reduced.

In Table V we show the average join and leave time of our proposed scheme when $R = 0.01$. We also show the worst case user join and leave time. Here we count the time cost for relocation or batch movement into the time cost of the preceding join or leave event. The worst case operation time remains the same for any R value. Comparing these time cost

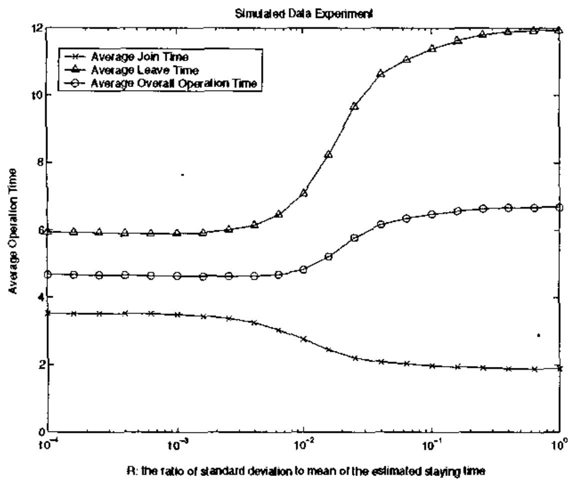


Fig. 10. Simulated Data Experiment

results with those for TGDH, we can see that the proposed scheme can improve the average user join time, and reduce the average leave time when the estimated staying time is fairly accurate.

VI. Discussions

A. Protocol Complexity

1) Time Complexity from Other Perspectives: In addition to the time cost for each join and leave event, which is $\Theta(\log(\log n))$ on average in our proposed scheme, it is interesting to examine the amount of time a user would spend on key update during his/her lifetime in the group, and the amount of time the whole group would spend on key update during the lifetime of the group communications.

Consider a sequence of n join events followed by n leave events. We assume that the first user joining the group is also the last one to leave the group. In the DST protocol, this first user will spend the majority of his/her life time in the main tree for key management purpose. On average, this first user will spend 2-round time for each user join event and 3-round time for each user leave event, assuming all users report their staying time accurately. Therefore this user has spent $\Theta(n)$ rounds in total on key update during his/her life time. Since this first user has the longest life-time among all users, $O(n)$ is the upper bound for any user's total key update time. For tree-based key agreement using a simple key tree, this first user will spend $\Theta(n \log n)$ rounds in total on key update.

From the system perspective, for the same sequence of events described above, the whole group will spend $\Theta(n \log(\log n))$ rounds in key update using the proposed DST protocol. If a key agreement using a simple key tree with only a main tree is employed, the time cost will be $\Theta(n \log n)$.

2) Communication Complexity: In this part, we discuss the average number of messages for user join and leave events under two scenarios.

In the first scenario, we assume that multicast is available for group communications. In particular, If a message needs to be sent to m users, sending one multicast message is enough. When the subgroup keys in the join tree are preserved during relocation (relocation method 1), the average number of messages for a join event is $\Theta(\log(\log n))$. Otherwise, using relocation method 2, the average number of messages needed for a join event is $\Theta(\log n)$. For a leave event, the average number of messages is always $\Theta(\log n)$.

In the second scenario, we assume that multicast is not available. If a message needs to be sent to m users, m duplicate copies of the same message must be sent. In this case the average number of messages is $O(n)$ for both user join and leave event.

3) Computation Complexity: In the proposed DST protocol, the total number of exponentiations performed by all users is $O(n)$ during the key update for a join or leave event. Such a measurement capture the overall computation load of the entire group.

For a particular user, the average number of exponentiations performed by him/her during join and leave events is less or equal to the average number of DH rounds in the same scenario. Therefore it is $O(\log(\log n))$.

B. The Group Coordinator

As suggested in [12], we prefer to have a group coordinator in the implementation of our scheme. The role of this group coordinator is to store the current key tree topology and manage future topological changes, such as determining the join location and organizing the batch movement. However, the trust in the coordinator is limited, since it is not responsible for generating and distributing keys. In implementation, the coordinator can be either a centralized or distributed third party. It can also consists of several or all members in the group.

The time complexity of the algorithms that a group coordinator needs to perform, such as a priority queue or some graph algorithms, may seemingly exceed those engaged in key updates. However, since we use DH round as the time unit for key update, the complexity of computing modular exponentiation in DH protocol is a dominating factor. Therefore the algorithmic complexity for the group coordinator would not be an important factor in the overall system time complexity.

VII. Conclusions

In this paper, we have applied dynamic amortization and scheduling techniques for time-efficient group key agreement and presented a new contributory key agreement, known as the Dynamic Subtree Group Key Agreement, for secure group communications. Built upon a tree-based key management framework, our proposed scheme employs a main tree as well as two subtrees that serve as temporary buffers for joining and leaving users. The join and exit subtrees help amortize the time cost for user join and leave events.

Focusing on time efficiency issues in contributory key management, our proposed scheme can achieve an average time cost of $\Theta(\log(\log n))$ for user join and leave events for a group of n users. In addition, our DST scheme reduces the total time cost of key update over a user's lifetime from $\Theta(n \log n)$ by the prior work to $\Theta(n)$, and over a system's life time from $\Theta(n \log n)$ to $\Theta(n \log(\log n))$. In the mean time, our proposed scheme also achieves low communication and computation overhead. These results suggest substantial savings by our proposed scheme, especially for large dynamic groups.

We have shown through analysis that the optimal subtree capacity is at the log scale of the group size. We have also designed an adaptive algorithm to activate the join/exit subtrees when the gain over using main tree only is substantial. Our experimental results on both simulated user activities and the real MBone data have shown that the proposed scheme outperforms the existing tree-based schemes in the events of group key establishment, user join and leave by a large margin for large and dynamic groups, and does not sacrifice any time efficiency for small groups.

ACKNOWLEDGEMENT

This work was supported in part by the Army Research Office under Award No. DAAD19-01-1-0494 and the National Science Foundation under Award No. CCR-0133704.

APPENDIX

In this appendix, we will show the inequality

$$\frac{1}{A}\sum_{k=1}^{A} r(k) \le \frac{1}{2}\log A + 1, \tag{18}$$

where $r(1) = 1$, $r(2^p + q) = 1 + r(q)$, p is a non-negative integer, and $q \in [1, 2^p]$ is a positive integer. The equality holds when A is a power of 2.

We first use induction to show that when $A = 2^p$, $p = 0, 1, 2, ...$, the equality holds.

When $A = 1$, LHS = RHS = 1.

Next, we assume the equality holds for $A = 2^p$, namely,

$$\frac{1}{2^p}\sum_{k=1}^{2^p} r(k) = \frac{1}{2}\log 2^p + 1. \tag{19}$$

Consider the case of $A = 2^{p+1}$.

$$\begin{aligned} LHS &= \frac{1}{2^{p+1}}\sum_{k=1}^{2^{p+1}} r(k) \\ &= \frac{1}{2^{p+1}}\left(\sum_{k=1}^{2^p} r(k) + \sum_{k=1}^{2^p}(r(k)+1)\right) \\ &= \frac{1}{2^{p+1}}\left(2\cdot(\frac{1}{2}\log 2^p + 1)2^p + 2^p\right) \quad (*) \\ &= \frac{1}{2}\log 2^{p+1} + 1 = RHS, \end{aligned}$$

where $(*)$ is obtained by using the induction assumption (19).

We now prove the inequality for any positive integer A. It is obvious to see that inequality is true for $A = 1, 2$. By induction, suppose that the inequality is true for all $1 \le A < 2^p + q$, and we consider $A = 2^p + q$, where $0 < q \le 2^p$.

$$\begin{aligned} LHS &= \frac{1}{A}\sum_{k=1}^{A} r(k) \\ &= \frac{1}{A}\left(\sum_{k=1}^{2^p} r(k) + \sum_{k=1}^{q}(r(k)+1)\right) \\ &\le \frac{1}{A}\left((\frac{1}{2}\log 2^p + 1)2^p + q(\frac{1}{2}\log q + 1) + q\right) \quad (**) \\ &= \frac{1}{2}\left\{\frac{1}{A}(2^p \log 2^p + q\log q + 2q)\right\} + 1, \end{aligned} \tag{20}$$

where $(**)$ is obtained by using the induction assumption.

To prove that (20) $\le \frac{1}{2}\log A + 1$ is equivalent to prove

$$\frac{2^p}{A}\log 2^p + \frac{q}{A}\log(4q) \le \log A. \tag{21}$$

Applying the identity $\ln k = \int_1^k \frac{1}{x}dx$, $\log k = \log e \cdot \ln k$, (21) can be written as an integration form

$$\log e\left\{\frac{2^p}{A}\int_z^{2^p}\frac{1}{x}dx + \frac{q}{A}\int_1^{4q}\frac{1}{x}dx\right\} \le \log e\int_1^A \frac{1}{x}dx$$

$$\Leftrightarrow 2^p\int_{2^p}^{A}\frac{1}{x}dx + q\left[\int_1^A \frac{1}{x}dx - \int_1^{4q}\frac{1}{x}dx\right] \ge 0 \tag{22}$$

We denote $B = 2^p$ and fix p (hence B is fixed). Thus $A = B + q$. It is straightforward to see that (22) holds when $B + q \ge 4q$, or $1 \le q \le \frac{B}{3}$.

When $B/3 \le q \le B$, (22) is equivalent to

$$\frac{2^p}{A}\int_{2^p}^{A}\frac{1}{x}dx - \frac{q}{A}\int_A^{4q}\frac{1}{x}dx \ge 0. \tag{23}$$

Since q is the only variable in (23), let $f(q)$ be the LHS of (23), and consider $f(q)$ as a continuous function of q

$$f(q) = \frac{B}{B+q}\int_B^{B+q}\frac{1}{x}dx - \frac{q}{B+q}\int_{B+q}^{4q}\frac{1}{x}dx,$$

where $q \in [B/3, B]$. Taking the derivative of $f(q)$, we get

$$\frac{d}{dq}f(q) = -\frac{B}{(B+q)^2}\int_B^{4q}\frac{1}{x}dx < 0. \tag{24}$$

In previous proof we showed that the equality of (18) holds when A is power of 2, i.e. $f(B) = 0$. We also showed that $f(q) > 0$ for $1 \le q \le \frac{B}{3}$. Since $f(B/3) > 0$, $f(B) = 0$, $f(q)$ is continuous on $[B/3, B]$ and $f'(q) < 0$, we must have $f(q) > 0$ on $[B/3, B]$. Thus (22) also holds for $B/3 \le q \le B$. This completes the proof.

REFERENCES

[1] S. Paul, *Multicast on the Internet and its applications*, Kluwer academic Publishers, 1998.
[2] L. Eschenauer and V.D. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and communications security*. 2002, pp. 41–47, ACM Press.

[3] M.J. Moyer, J.R. Rao, and P. Rohatgi, "A survey of security issues in multicast communications," *IEEE Network*, pp. 12–23, Nov./Dec. 1999.

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

[5] P. Judge and M. Ammar, "Gothic: A group access control architecture for secure multicast and anycast," in *Proceedings of the IEEE INFOCOM'02*, 2002, pp. 1547–1556.

[6] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, "Multicast security: A taxonomy and some efficient constructions," in *Proceedings of the IEEE INFOCOM'99*, 1999, pp. 708–716.

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

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

[9] H. Harney and C. Muckenhirn, "Group key management protocol (GKMP) specification," RFC 2093, July 1997.

[10] D. Wallner, E. Harder, and R. Agee, "Key management for multicast: Issues and architecture," Internet-Draft draft-wallner-key-arch-00.txt, June 1997.

[11] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha, "Key management for secure internet multicast using boolean function minimization techniques," in *Proceedings of the IEEE INFOCOM'99*, 1999, vol. 2, pp. 689–698.

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

[13] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *Proceedings of the 7th ACM Conference on Computer and Communications Security*. 2000, pp. 235–244, ACM Press.

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

[15] Y. Sun, W. Trappe, and K.J.R. Liu, "An efficient key management scheme for secure wireless multicast," in *Proceedings of the IEEE International Conference on Communications*, 2002, vol. 2, pp. 1236–1240.

[16] R. Molva and A. Pannetrat, "Scalable multicast security in dynamic groups," in *Proceedings of the 6th ACM conference on Computer and communications security*, 1999, pp. 101–112.

[17] S. Mittra, "Iolus: a framework for scalable secure multicasting," in *Proceedings of the ACM SIGCOMM'97*. 1997, pp. 277–288, ACM Press.

[18] 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.

[19] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner, "The VersayKey framework: Versatile group key management," *IEEE Journal on Selected Areas in Communications*, pp. 1614–1631, Sept. 1999.

[20] W. Trappe, Y. Wang, and K.J.R. Liu, "Establishment of conference keys in heterogeneous networks," in *Proceedings of the IEEE International Conference on Communications*, 2002, pp. 1236–1240.

[21] B. Sun, W. Trappe, Y. Sun, and K.J.R. Liu, "A time-efficient contributory key agreeement scheme for secure group communications," in *Proceedings of the IEEE International Conference on Communications*, 2002, pp. 1159–1163.

[22] S. Zhu, S. Setia, and S. Jajodia, "Performance optimizations for group key management schemes," in *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003, pp. 163–171.

[23] A. Ballardie, "Scalable multicast key distribution," RFC 1949, May 1996.

[24] K. Becker and U. Wille, "Communication complexity of group key distribution," in *Proceedings of the 5th ACM conference on Computer and communications security*. 1998, pp. 1–6, ACM Press.

[25] J. Snoeyink, S. Suri, and G. Varghese, "A lower bound for multicast key distribution," in *Proceedings of the IEEE INFOCOM'01*, 2001, vol. 1, pp. 422–431.

[26] D. Balenson, D. McGrew, and A. Sherman, "Key management for large dynamic groups: One-way function trees and amortized initialization," IETF Internet draft (work in progress), August 2000.

[27] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, November 1976.

[28] I. Ingemarsson, D.T. Tang, and C.K. Wong, "A conference key distribution system," *IEEE Transactions on Information Theory*, vol. IT-28, no. 5, pp. 714–720, September 1982.

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

[30] S.E. Eldridge and C.D. Walter, "Hardware implementation of montgomery's modular multiplication algorithm," *IEEE Transactions on Computers*, vol. 42, no. 6, pp. 693–699, June 1993.

[31] J.L. Hennessy and D.A. Patterson, *Computer architecture: a quantitative approach*, chapter 5, Morgan Kaufmann publishers, Inc., second edition, 1996.

[32] T.H. Corman, C.E. Leiserson, and R.L. Rivest, *Introduction to algorithms*, chapter 7, The MIT Press and McGraw-Hill Book Company, second edition, 2001.

[33] K.C. Almeroth and M.H. Ammar, "Multicast group behavior in the Internet's multicast backbone (MBone)," *IEEE Communications Magazine*, pp. 124–129, June 1997.

[34] K.C. Almeroth, "A long-term analysis of growth and usage patterns in the multicast backbone (MBone)," in *Proceedings of the IEEE INFOCOM'00*, March 2000, vol. 2, pp. 824–833.

[35] MBone user activity data, "ftp://ftp.cc.gatech.edu/people/kevin/release-data," March 2003.