Distributed Collaborative Key Agreement and Authentication Protocols for Dynamic Peer Groups

Patrick P. C. Lee, John C. S. Lui, and David K. Y. Yau

Abstract— **We consider several distributed collaborative key agreement and authentication protocols for dynamic peer groups. There are several important characteristics which make this problem different from traditional secure group communication. They are (1) distributed nature in which there is *no centralized key server*, (2) collaborative nature in which the group key is *contributory* (i.e., each group member will collaboratively contribute its part to the global group key), and (3) dynamic nature in which existing members may leave the group while new members may join. Instead of performing individual rekeying operations, i.e., recomputing the group key after every join or leave request, we discuss an interval-based approach of rekeying. We consider three *interval-based distributed rekeying algorithms*, or *interval-based algorithms* for short, for updating the group key: (1) the *Rebuild algorithm*, (2) the *Batch algorithm*, and (3) the *Queue-batch algorithm*. Performance of these three interval-based algorithms under different settings, such as different join and leave probabilities, is analyzed. We show that the interval-based algorithms significantly outperform the individual rekeying approach and that the Queue-batch algorithm performs the best among the three interval-based algorithms. More importantly, the Queue-batch algorithm can substantially reduce the computation and communication workload in a highly dynamic environment. We further enhance the interval-based algorithms in two aspects: authentication and implementation. Authentication focuses on the security improvement, while implementation realizes the interval-based algorithms in real network settings. Our work provides a fundamental understanding about establishing a group key via a distributed and collaborative approach for a dynamic peer group.**

I. Introduction

With the emergence of many group-oriented distributed applications such as tele/video-conferencing and multi-player games, there is a need for security services to provide group-oriented communication privacy and data integrity. To provide this form of group communication privacy, it is important that members of the group can establish a common secret key for encrypting group communication data. To illustrate the utility of this type of applications, consider a group of people in a peer-to-peer or ad-hoc network having a closed and confidential meeting. Since they do not have a previously agreed upon common secret key, communication between group members is susceptible to eavesdropping. To solve the problem, we need a *secure distributed group key agreement and authentication protocol* so that people can establish and authenticate a common group key for secure and private communication. Note that this type of key agreement protocols is both distributed and contributory in nature: each member of the group contributes its part to the overall group key.

It is important to point out that the type of distributed group key agreement protocols we study is very different from traditional centralized group key management protocols. Centralized protocols rely on a *centralized key server* to efficiently distribute the group key. An excellent body of work on centralized key distribution protocols exists in [14], [16], [21], [22]. In those approaches, group members are arranged in a logical key hierarchy known as a *key tree*. Using the tree topology, it is easy to distribute the group key to members whenever there is any change in the group membership (e.g., a new member joins or an existing member leaves the group). In the distributed key agreement protocols we consider, however, there is no centralized key server available. This arrangement is justified in many situations – e.g., in peer-to-peer or ad-hoc networks where centralized resources are not readily available. Moreover, an advantage of distributed protocols over the centralized protocols is the increase in system reliability, because the group key is generated in a shared and contributory fashion and there is no single-point-of-failure.

In the special case of a communication group having only two members, these members can create a group key using the Diffie-Hellman key exchange protocol [6]. In the protocol, members X and Y use a cyclic group of prime order p with the generator α. They can generate their secret exponents e_X and e_Y, respectively. Member X (resp., Y) can compute its public key α^{e_X} (resp., α^{e_Y}) and send it to Y (resp., X). Since both members know their own exponent, they can each raise the other party's public key to the exponent and produce a common group key $\alpha^{e_X e_Y}$. Using the common group key, X and Y can encrypt their data to prevent eavesdropping by intruders.

In this paper, we propose, based on the Tree-based Group Diffie-Hellman protocol [11], several group key agreement protocols for a dynamic communication group in which members are located in a distributed fashion and can join and leave the group at any time. The contributions of our work are:

- Instead of using individual rekeying operations, we propose three *interval-based distributed rekeying algorithms*, or *interval-based algorithms* for short, to significantly reduce the computation and communication costs of maintaining the group key. The interval-based approach provides rekeying efficiency for dynamic peer groups while preserving both distributed (i.e., no centralized key

This work is supported in part by RGC Research Grant 4186/03E and in part by the National Science Foundation under grant numbers CCR-9875742 (CAREER) and CNS-0305496.

P. Lee is with the Department of Computer Science, Columbia University, New York, NY 10027, USA (email: pclee@cs.columbia.edu).

J. Lui is with the Department of Computer Science & Engineering, The Chinese University of Hong Kong, Hong Kong (email: cslui@cse.cuhk.edu.hk).

D. Yau is with the Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, USA (email: yau@cs.purdue.edu).

An early version of this paper appeared in the IEEE ICNP'02 conference.

server is involved) and contributory (i.e., each member contributes to the resulting group key) properties.

- We evaluate the performance of our interval-based algorithms through both mathematical and simulation-based analysis. In particular, we compare their performance with that of a centralized key distribution approach.
- We propose an authenticated group key agreement protocol that can be incorporated into the interval-based algorithms. We evaluate the performance of this authenticated approach and prove its security properties.
- We implemented the *SEcure Communication Library (SEAL)* that realizes the interval-based algorithms. The library provides a programming interface for the development of secure group-based applications.

The rest of the paper proceeds as follows. In Section II, we overview the Tree-Based Group Diffie-Hellman protocol that establishes a group key with more than two members in a dynamic peer group. In Section III, we present three interval-based algorithms that establish the group key for a dynamic peer group. In Section IV, we evaluate the interval-based algorithms under dynamic joins and leaves. In Section V, we describe an authenticated group key agreement protocol and analyze its security properties. In Section VI, we provide the implementation details of the interval-based algorithms. Section VII reviews related work, and Section VIII concludes.

II. Tree-Based Group Diffie-Hellman Protocol

To efficiently maintain the group key in a dynamic peer group with more than two members, we use the Tree-Based Group Diffie-Hellman (TGDH) protocol proposed in [11]. Each member maintains a set of keys, which are arranged in a hierarchical *binary tree*. We assign a node ID v to every tree node. For a given node v, we associate a *secret* (or private) key K_v and a *blinded* (or public) key BK_v. All arithmetic operations are performed in a cyclic group of prime order p with the generator α. Therefore, the blinded key of node v can be generated by

$$BK_v = \alpha^{K_v} \bmod p. \tag{1}$$

Each leaf node in the tree corresponds to the individual secret and blinded keys of a group member M_i. Every member holds all the secret keys along its *key path* starting from its associated leaf node up to the root node. Therefore, the secret key held by the root node is shared by all the members and is regarded as the *group key*. Fig. 1 illustrates a possible key tree with six members M_1 to M_6. For example, member M_1 holds the keys at nodes 7, 3, 1, and 0. The secret key at node 0 is the group key of this peer group.

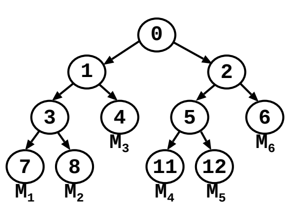


Fig. 1. A key tree used in the Tree-Based Group Diffie-Hellman protocol.

The node ID of the root node is set to 0. Each *non-leaf* node v consists of two child nodes whose node IDs are given by $2v+1$ and $2v+2$. Based on the Diffie-Hellman protocol [6], the secret key of a non-leaf node v can be generated by the secret key of one child node of v and the blinded key of another child node of v. Mathematically, we have

$$\begin{aligned} K_v &= (BK_{2v+1})^{K_{2v+2}} \bmod p \\ &= (BK_{2v+2})^{K_{2v+1}} \bmod p \\ &= \alpha^{K_{2v+1}K_{2v+2}} \bmod p. \end{aligned} \tag{2}$$

Unlike the keys at non-leaf nodes, the secret key at a leaf node is selected by its corresponding group member through a secure pseudo random number generator.

Since the blinded keys are publicly known, every member can compute the keys along its key path to the root node based on its individual secret key. To illustrate, consider the key tree in Fig. 1. Every member M_i generates its own secret key and all the secret keys along the path to the root node. For example, member M_1 generates the secret key K_7 and it can request the blinded key BK_8 from M_2, BK_4 from M_3, and BK_2 from either M_4, M_5, or M_6. Given M_1's secret key K_7 and the blinded key BK_8, M_1 can generate the secret key K_3 according to Eq. 2. Given the blinded key BK_4 and the newly generated secret key K_3, M_1 can generate the secret key K_1 based on Eq. 2. Given the secret key K_1 and the blinded key BK_2, M_1 can generate the secret key K_0 at the root. From that point on, any communication in the group can be encrypted based on the secret key (or group key) K_0.

To provide both backward confidentiality (i.e., joined members cannot access previous communication data) and forward confidentiality (i.e., left members cannot access future communication data), *rekeying*, which means renewing the keys associated with the nodes of the key tree, is performed whenever there is any group membership change including any new member joining or any existing member leaving the group. Let us first consider individual rekeying, meaning that rekeying is performed after every single join or leave event. Before the group membership is changed, a special member called the *sponsor* is elected to be responsible for updating the keys held by the new member (in the join case) or departed member (in the leave case). We use the convention that the rightmost member under the subtree rooted at the sibling of the join and leave nodes will take the sponsor role. Note that the existence of a sponsor does not violate the decentralized requirement of the group key generation since the sponsor does not add extra contribution to the group key.

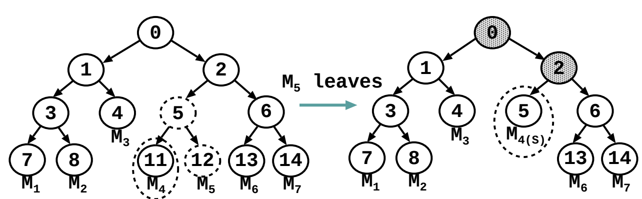


Fig. 2. Illustration of the rekeying operation after a single leave.

Fig. 2 depicts a member leave event. Suppose that member M_5 leaves the system. Node 11 is then *promoted* to node 5, and nodes 2 and 0 become *renewed nodes*, defined as the non-leaf nodes whose associated keys in the key tree are renewed. Also, member M_4 becomes the sponsor. It renews the secret keys K_2 and K_0 and broadcasts the blinded keys BK_2 and BK_5 to all the members. Members M_1, M_2, and M_3, upon receiving the blinded key BK_2, compute the new group key K_0. Similarly, members M_6 and M_7, upon receiving BK_5, can compute K_2 and then the new group key K_0.

Fig. 3 illustrates a new member M_8 that wishes to join the group. M_8 has to first determine the *insertion node* under which M_8 can be inserted. To *add* a node, say v' (or tree, say T') to the insertion node, a new node, say n', is first created. Then the subtree rooted at the insertion node becomes the left child of the node n', and the node v' (or the root node of the tree T') becomes the right child of the node n'. The node n' will replace the original location of the insertion node. The insertion node is either the rightmost shallowest position such that the join does not increase the tree height, or the root node if the tree is initially well balanced (in this case, the height of the resulting tree will be increased by 1). Fig. 3 illustrates this concept. The insertion node is node 5 and the sponsor is M_4. M_8 then broadcasts its blinded key BK_{12} upon insertion. Given BK_{12}, M_4 renews K_5, K_2, and K_0, and then broadcasts the blinded keys BK_5 and BK_2 to all members in the group. After receiving the blinded keys from M_4, all remaining members can rekey all the nodes along their key paths and compute the new group key K_0.

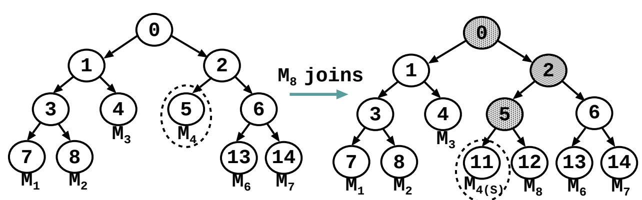


Fig. 3. Illustration of the rekeying operation after a single join.

Based on the above leave and join events in Figs. 2 and 3, we find that we can *reduce* one rekeying operation if we can simply change the association of node 12 from M_5 to M_8. *Interval-based rekeying* is thus proposed such that rekeying is performed on a batch of join and leave requests so as to reduce the number of rekeying operations. Members carry out rekeying operations at regular *rekeying intervals*. In the following section, we describe the interval-based approach to manage the rekeying operations.

III. Interval-Based Distributed Rekeying Algorithms

We develop three *interval-based distributed rekeying algorithms* (or *interval-based algorithms* for short), termed the *Rebuild algorithm*, the *Batch algorithm*, and the *Queue-batch algorithm*. Interval-based rekeying maintains the rekeying frequency regardless of the dynamics of join and leave events, with a tradeoff of weakening both backward and forward confidentialities as a result of delaying the update of the group key. The three interval-based algorithms are developed based on the following assumptions:

- All members are trusted in the key establishment process.
- The group communication satisfies *view synchrony* [7], [11] that defines reliable and ordered message delivery under the same membership view. Intuitively, when a member broadcasts a message under a membership view, the message is delivered to same set of members viewed by the sender. Note that this view-synchrony property is essential not only for group key agreement, but also for reliable multipoint-to-multipoint group communication in which every member can be a sender [11].
- Rekeying operations of all members are synchronized to be carried out at the beginning of every rekeying interval.
- When a new member sends a join request, it also includes its individual blinded key.
- All members know the existing key tree structure and all the blinded keys within the tree.
- To obtain the blinded keys of the renewed nodes, the key paths of the sponsors should contain those renewed nodes. Since the interval-based rekeying operations involve nodes lying on more than one key paths, more than one sponsors may be elected. Also, a renewed node may be rekeyed by more than one sponsor. Therefore, we assume that the sponsors can coordinate with one another such that the blinded keys of all the renewed nodes are broadcast only once.

We adopt the following notations in our description. Let T denote the existing key tree. Assume that $L \geq 0$ existing members $\boldsymbol{M}^l = \langle M_1^l, \cdots, M_L^l \rangle$ wish to leave and $J \geq 0$ new members $\boldsymbol{M}^j = \langle M_1^j, \cdots, M_J^j \rangle$ wish to join the group within a rekeying interval.

A. Rebuild Algorithm

The motivation of the Rebuild algorithm is to *minimize* the resulting tree height so that the rekeying operations for each group member can be reduced. At the beginning of every rekeying interval, we reconstruct the whole key tree with all existing members that remain in the communication group, together with the newly joining members. The resulting tree is a *left-complete* tree, in which the depths of the leaf nodes differ by at most one and those deeper leaf nodes are located at the leftmost positions. The pseudo-code of the Rebuild algorithm to be performed by every member is shown in Fig. 4.

Rebuild (T, $\boldsymbol{M}^j$, J, $\boldsymbol{M}^l$, L)
1. obtain all members from T and store them in $\boldsymbol{M}'$;
2. remove the L leaving members in $\boldsymbol{M}^l$ from $\boldsymbol{M}'$;
3. add the J new members in $\boldsymbol{M}^j$ to $\boldsymbol{M}'$;
4. create a new binary tree T' based on members in $\boldsymbol{M}'$ and set $T = T'$;
5. elect all members to be sponsors;
6. rekey renewed nodes and broadcast new blinded keys in T;

Fig. 4. Pseudo-code of the Rebuild algorithm.

Fig. 5 shows the scenario where members M_2, M_5, and M_7 wish to leave and a new member M_8 wishes to join the communication group. Based on the algorithm, the resulting

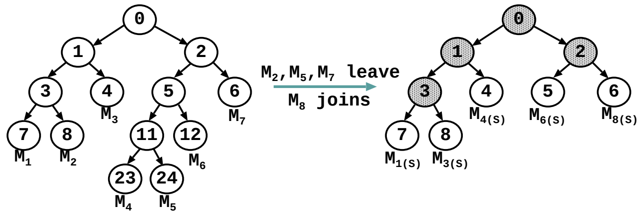


Fig. 5. Example of the Rebuild algorithm.

key tree consists of five members and has all non-leaf nodes renewed. Besides, the sponsors include all the five members.

Rebuild is suitable for some cases, such as when the membership events are so frequent that we can directly reconstruct the whole key tree for simplicity, or when some members lose the rekeying information and the simplest way of recovery is to rebuild the key tree. We can explore the situations where Rebuild is applicable.

B. Batch Algorithm

The Batch algorithm is based on the centralized approach in [14], which is now applied to a distributed system without a centralized key server. The pseudo-code of the Batch algorithm is given in Fig. 6. Given the numbers of joins and leaves within a rekeying interval, we attach new group members to different leaf positions of the key tree in order to keep the key tree as balanced as possible.

The Batch algorithm is illustrated with two examples. In Fig. 7, we show the case where $L > J > 0$. Suppose M_2, M_5, and M_7 leave and a new member M_8 wishes to join. The following steps are carried out: (i) M_8 broadcasts its join request, including its individual blinded key. (ii) The leaf node 6 associated with M_7 is replaced by the node of M_8, and the leaf nodes 8 and 24 are removed. Nodes 7 and 23 are promoted to nodes 3 and 11, respectively. (iii) M_1, M_4, M_6, and M_8 are elected to be the sponsors. M_1 renews secret keys K_1 and K_0, and M_4 renews K_5, K_2, and K_0. M_1 then broadcasts BK_1, and M_4 broadcasts BK_5 and BK_2. M_6 and M_8, though having the sponsor role, do not need to broadcast any blinded keys as M_4 has already broadcast this information. (iv) Finally, every member computes the new group key based on the received blinded keys.

Fig. 8 illustrates the case where $J > L > 0$. Suppose M_8, M_9, and M_{10} join, and M_2 and M_7 leave. The rekeying process is: (i) M_8, M_9, and M_{10} broadcast their join requests together with their own individual blinded keys. (ii) M_8 and M_9 form the subtree T_1' and M_{10} is the only member of T_2'. The root of T_1' replaces node 6 and the root of T_2' replaces node 8. (iii) The sponsors are M_1, M_6, M_8, M_9, and M_{10}. M_8 and M_9 first need to compute the secret key K_6, and either one of them computes and broadcasts the new blinded key BK_6. (iv) M_1 (or M_{10}) renews K_3 and K_1, and broadcasts BK_3 and BK_1. M_6 renews K_2 and broadcasts BK_2. (v) Finally, all the members can compute the new group key K_0.

C. Queue-batch Algorithm

We find that the previous approaches perform all rekeying steps at the beginning of every rekeying interval. This results in high processing load during the update instance and thereby delays the start of the secure group communication. Thus, we propose a more effective algorithm which we call the *Queue-batch* algorithm. Its intuition is to reduce the rekeying load by pre-processing the joining members during the idle rekeying interval.

The Queue-batch algorithm is divided into two phases,

Batch (T, $\boldsymbol{M}^j$, J, $\boldsymbol{M}^l$, L)

if ($L == 0$) { /* pure join case */
  create a new tree T' based on new members in $\boldsymbol{M}^j$;
  either add T' to the shallowest node of T (which need not be the leaf node) such that the merge will not increase the height of the result tree, or add T' to the root node of T if the merge to any node of T will increase the tree height;
} **else** { /* L > 0 */
  sort $\boldsymbol{M}^l$ in an ascending order of the associated node IDs of the members and store the results in $\boldsymbol{M}^{l,s} = \langle M_1^{l,s}, \cdots, M_L^{l,s} \rangle$;
  if ($L > J$)
    if ($J > 0$) {
      replace the departed nodes of $\langle M_1^{l,s}, \cdots, M_J^{l,s} \rangle$ with J joined nodes;
    }
    remove remaining $L - J$ leaving leaf nodes and promote their siblings;
  } **else** { /* J >= L */
    divide $\boldsymbol{M}^j$ into L subgroups $\boldsymbol{G} = \langle G_1, \cdots, G_L \rangle$ such that the first $J \bmod L$ subgroups $\langle G_1, \cdots, G_{J \bmod L} \rangle$ contain $\lfloor \frac{J}{L} \rfloor + 1$ new members and the rest contain $\lfloor \frac{J}{L} \rfloor$ new members;
    create L subtrees $\langle T_1', \cdots, T_L' \rangle$ for the subgroups $\boldsymbol{G}$;
    replace the departed nodes of $\langle M_1^{l,s}, \cdots, M_{J \bmod L}^{l,s} \rangle$ with the roots of $\langle T_1', \cdots, T_{J \bmod L}' \rangle$ and the remaining departed nodes with the roots of remaining subtrees;
  }
}
elect the members to be sponsors if they are new members, or the rightmost members of the subtrees rooted at the siblings of the departed nodes or replaced nodes in T;
if (sponsor) /* responsibility of the sponsor */
  rekey the renewed nodes and broadcast the new blinded keys;

Fig. 6. Pseudo-code of the Batch algorithm.

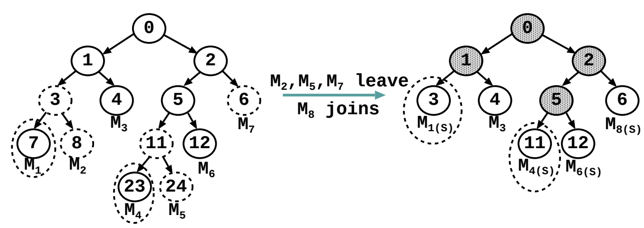


Fig. 7. Example 1 of the Batch algorithm where $L > J > 0$.

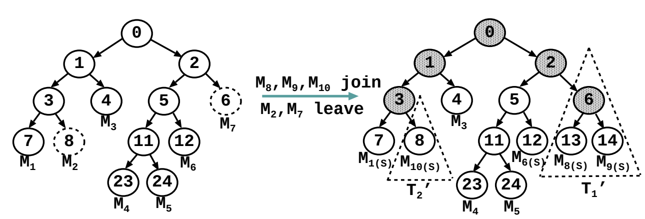


Fig. 8. Example 2 of the Batch algorithm where $J > L > 0$.

namely the *Queue-subtree* phase and the *Queue-merge* phase. The first phase occurs whenever a new member joins the communication group during the rekeying interval. In this case, we append this new member in a temporary key tree T'. The second phase occurs at the beginning of every rekeying interval and we merge the temporary tree T' (which contains all newly joining members) to the existing key tree T. The pseudo-codes of the Queue-subtree phase and the Queue-merge phase are illustrated in Figs. 9 and 10.

Queue-subtree (T')
if (a new member joins) {
    if (T' == NULL) `/* no new members in T' */`
        create a new tree T' with the only one new member;
    else { `/* there are new members in T' */`
        find the insertion node;
        add the new member to T';
        elect the rightmost member under the subtree rooted at the sibling of the joining node to be the sponsor;
        if (sponsor) `/* sponsor's responsibility */`
            rekey renewed nodes and broadcast new blinded keys;
    }
}

Fig. 9. Pseudo-code of the Queue-subtree phase.

Queue-merge (T, T', $\boldsymbol{M}^l$, L)
if ($L == 0$) { `/* there is no leave */`
    add T' to either the shallowest node (which need not be the leaf node) of T such that the merge will not increase the resulting tree height, or the root node of T if the merge to any locations will increase the resulting tree height;
} **else** { `/* there are leaves */`
    add T' to the highest leave position of the key tree T;
    remove remaining $L-1$ leaving leaf nodes and promote their siblings;
}
elect members to be sponsors if they are the rightmost members of the subtree rooted at the sibling nodes of the departed leaf nodes in T, or they are the rightmost member of T';
if (sponsor) `/* sponsor's responsibility */`
    rekey renewed nodes and broadcast new blinded keys;

Fig. 10. Pseudo-code of the Queue-merge phase.

The Queue-batch algorithm is illustrated in Fig. 11, where members M_8, M_9, and M_{10} wish to join the communication group, while M_2 and M_7 wish to leave. Then the rekeying process is as follows: (i) In the *Queue-subtree* phase, the three new members M_8, M_9, and M_{10} first form a tree T'. M_{10}, in this case, is elected to be the sponsor. (ii) In the *Queue-merge* phase, the tree T' is added at the highest departed position, which is at node 6. Also, the blinded key of the root node of T', which is BK_6, is broadcast by M_{10}. (iii) The sponsors M_1, M_6, and M_{10} are elected. M_1 renews the secret key K_1 and broadcasts the blinded key BK_1. M_6 renews the secret key K_2 and broadcasts the blinded key BK_2. (iv) Finally, all members can compute the group key.

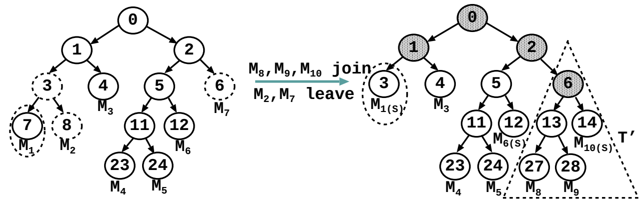


Fig. 11. Example of the Queue-merge phase.

IV. Performance Evaluation

To reflect the latency of generating the latest group key for data confidentiality, we evaluate the performance of the interval-based algorithms in two aspects: mathematical analysis and simulation-based experiments. The mathematical analysis considers the complexity of the algorithms under the assumption that the key tree is completely balanced. Using simulations, we then study their performance in a more general setting. We also compare the performance of our interval-based algorithms and a centralized key distribution approach.

A. Mathematical Analysis

We present the mathematical analysis of the three proposed algorithms based on two performance measures, namely:

1) *Number of exponentiation operations:* This metric gives a measure of the computation load of all members in the communication group.
2) *Number of renewed nodes:* As defined in Section II, a node is said to be *renewed* if it is a non-leaf node and its associated keys are renewed. This metric measures the communication cost since the new blinded keys of the renewed nodes have to be broadcast to the whole group.

For simplicity, we assume the following in the analysis:

- The existing key tree is completely balanced prior to the interval-based rekeying event.
- Every existing member has the same leave probability.
- The computation of the blinded group key of the root node is counted in the blinded key computations. With this assumption, the number of blinded key computations simply equals the number of renewed nodes, provided that the blinded key of each renewed node is broadcast only once.

Let N be the number of members originally in the communication group, L (where $0 \leq L \leq N$) be the number of members that wish to leave the group, and $J \geq 0$ be the number of new members that want to join the group. Let T denote the existing tree which contains N members. The level of a node v is $l = \lfloor \log_2(v+1) \rfloor$, where v is the node ID, and the maximum level of T is h. Based on the first assumption, i.e., the key tree is initially balanced, we know that $N = 2^h$. Also, let $\mathcal{R}_{alg}$ be the number of renewed nodes and $\mathcal{E}_{alg}$ be the number of exponentiations for the particular algorithm *alg*. The performance measure $\mathcal{E}_{alg}$ is composed of two parts: $\mathcal{E}^s_{alg}$ and $\mathcal{E}^b_{alg}$, which respectively represent the number of exponentiations of calculating the secret keys (which is done by all members) and that of calculating the blinded keys (which is done by sponsors only). We have

$$\mathcal{E}_{alg} = \mathcal{E}^s_{alg} + \mathcal{E}^b_{alg}. \tag{3}$$

Also, we know the number of blinded key computations is

$$\mathcal{E}^b_{alg} = \mathcal{R}_{alg}, \tag{4}$$

which is simply the mathematical interpretation of the last assumption.

In the following subsections, we evaluate the number of renewed nodes $\mathcal{R}_{alg}$ for the three interval-based algorithms. Readers can refer to [13] for the analysis of the number of secret key computations $\mathcal{E}^s_{alg}$.

1) Analysis of the Rebuild Algorithm: Given N, L and J, we can obtain the *exact* expressions for the two performance measures $\mathcal{R}_{Rebuild}$ and $\mathcal{E}_{Rebuild}$. It is important to note that the derived expressions below are valid even if the existing key tree T is not completely balanced originally.

The resulting number of members is $N^* = N - L + J \geq 0$. Thus, the number of renewed nodes is

$$\mathcal{R}_{Rebuild}(N^*) = \begin{cases} 0, & \text{if } N^* = 0 \\ N^* - 1, & \text{otherwise.} \end{cases} \tag{5}$$

2) Analysis of the Batch Algorithm: Since the actual performance of the Batch algorithm depends on the membership leave positions whenever $L > 0$, we consider only the *expected* performance measures rather than the exact ones, although the exact and expected results are the same when $L = 0$. Our analysis adopts the convention that the combination $\binom{n}{r}$ equals 0 if $n < 0$, $r < 0$, or $n < r$.

We first derive the expected number of renewed nodes. Consider a node v at level l. In a completely balanced tree, node v has $N/2^l$ descendants. At a rekeying instance, node v can be in one of the three distinct states: *no-change*, *pruned*, and *renewed*. Node v can be in the "no-change" state if none of its $N/2^l$ descendants wish to leave. Thus, the probability of node v being in the "no-change" state is

$$P[\text{node } v \text{ is no-change}] = \frac{\binom{N-N/2^l}{L}}{\binom{N}{L}}. \tag{6}$$

Node v is pruned if all its descendants leave, or all descendants of either its left or right subtree (but not both) leave. In the latter case, node v is pruned because it is promoted to its parent (see Section III-B). Thus, the expected number of renewed nodes is

$$E[\mathcal{R}_{Batch}] = \begin{cases} 0, & \text{if } J = 0, L = N \\ \sum_{l=0}^{h-1} 2^l \left[1 - \frac{\binom{N-N/2^l}{L}}{\binom{N}{L}}\right] + (J - L), & \text{otherwise.} \end{cases} \tag{7}$$

The equation exhibits different interpretations depending on the values of J and L. If all members leave the group but no new member joins, then the key tree disappears and hence the number of renewed nodes is zero. Otherwise, we have two possibilities. If $J \geq L$, then the first term represents the expected number of (non-leaf) nodes in the original key tree that are renewed, and the second term $(J - L)$ refers to the number of additional renewed nodes introduced to the key tree. No nodes are pruned in this case since all leaving leaf nodes are substituted by the joining ones. On the other hand, if $L > J$, then the first term corresponds to the expected number of (non-leaf) nodes that have at least one leaving descendant. The correctness of the first term relies on the assumption that the J newly joining members (if $J > 0$) randomly select L leaving leaf nodes for replacement given that those leaving leaf nodes are at the same level h in a completely balanced tree. The second term then subtracts $L - J$ for the number of non-leaf nodes that are pruned, and hence the remaining nodes are renewed nodes.

3) Analysis of the Queue-batch Algorithm: The main idea of the Queue-batch algorithm exploits the idle rekeying interval to pre-process some rekeying operations. When we compare its performance with the Rebuild or Batch algorithms, we only need to consider the rekeying operations occurring at the beginning of every rekeying interval.

When $J = 0$, Queue-batch is equivalent to Batch in the pure leave scenario. For $J > 0$, the number of renewed nodes in Queue-batch during the Queue-merge phase is equivalent to that of Batch when $J = 1$. Thus, the expected number of renewed nodes is

$$E[\mathcal{R}_{Queue-batch}] = \begin{cases} 0, & \text{if } J = 0,\ L = N \\ \sum_{l=0}^{h-1} 2^l \left[1 - \frac{\binom{N-N/2^l}{L}}{\binom{N}{L}}\right] - L, & \text{if } J=0,\ 0 \leq L < N \\ \sum_{l=0}^{h-1} 2^l \left[1 - \frac{\binom{N-N/2^l}{L}}{\binom{N}{L}}\right] - (L-1), & \text{otherwise.} \end{cases} \tag{8}$$

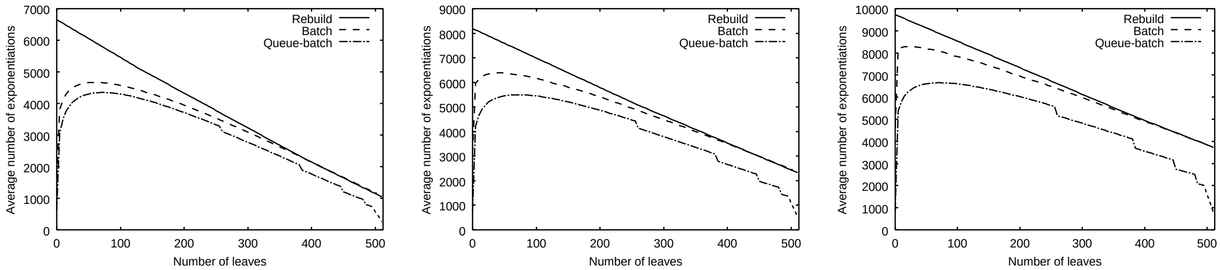
4) Evaluation: We evaluate the metrics of our three interval-based algorithms based on the mathematical models. We start with a well-balanced key tree involving 512 members and then calculate the metrics with different values of joins and leaves (i.e., J and L).

Figs. 12 and 13 illustrate the average number of exponentiations and the average number of renewed nodes under different numbers of joining and leaving members. We observe that Queue-batch outperforms the other two interval-based algorithms in all cases. Specifically, we note that there is a significant computation/communication reduction when the peer group is very dynamic (i.e., high number of members that wish to join or leave the communication group). We explain this phenomenon in Section IV-B.

B. Simulations

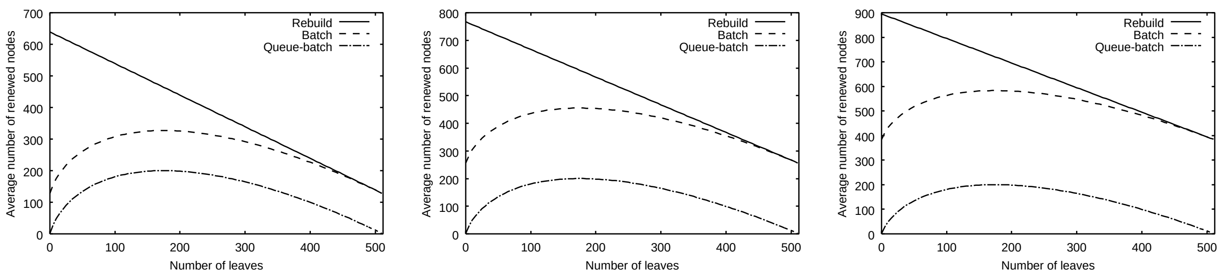
The previous subsection quantifies the performance measures by assuming that the existing key tree is completely balanced. In this subsection, we perform a more extensive performance study via simulations under different experimental settings. Our simulations focus on three performance metrics: (i) number of exponentiations, (ii) number of renewed nodes, and (iii) number of rounds required to generate the group key.

In our simulations, we consider a finite population of 1024 users. Out of these 1024 users, there are 512 members originally in a communication group at the beginning of each experiment. We assume that potential members outside the group have a tendency to join the group with a fixed join



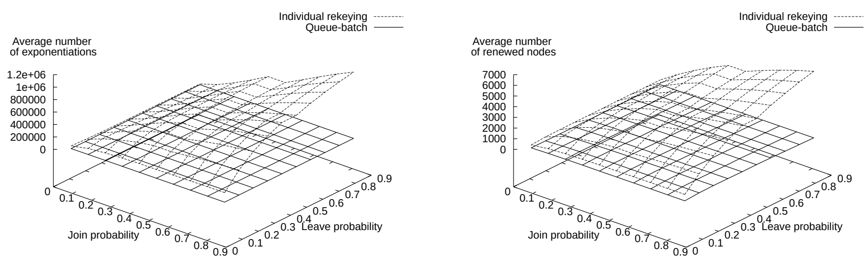
(a) J = 128 (b) J = 256 (c) J = 384

Fig. 12. Mathematical analysis: Average numbers of exponentiations at different numbers of joins based on mathematical models.



(a) J = 128 (b) J = 256 (c) J = 384

Fig. 13. Mathematical analysis: Average numbers of renewed nodes at different numbers of joins based on mathematical models.



(a) Average number of exponentiations (b) Average number of renewed nodes

Fig. 14. Experiment 1: Comparison between individual rekeying and the Queue-batch algorithm.

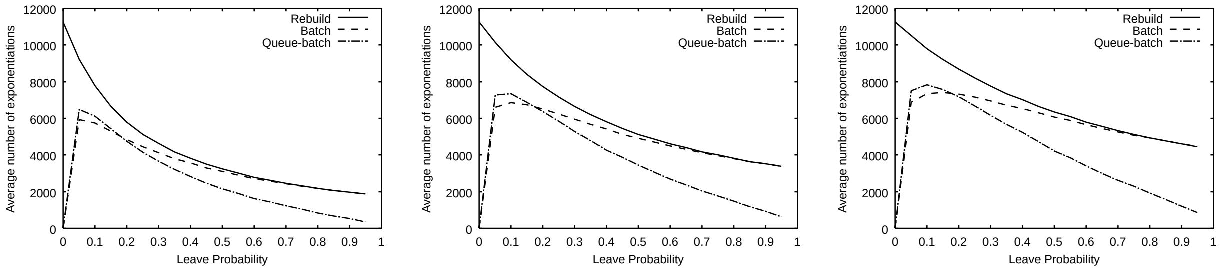
probability. Similarly, members within the group have a fixed leave probability of leaving the group. We let p_J and p_L denote the join and leave probabilities, respectively.

Experiment 1: (Comparison between individual rekeying and interval-based rekeying) We first demonstrate through simulations that interval-based rekeying outperforms individual rekeying. Given a number of join and leave requests, we consider a particular case where the individual rekeying approach first processes one by one the join requests followed by the leave requests. We then run the simulations over 300 rekeying intervals and compute the average results.

Fig. 14 illustrates the performance differences between individual rekeying and the Queue-batch algorithm. We observe that Queue-batch perform much better than the individual rekeying method, especially under high join and high leave probabilities. Similar results are observed for the comparisons between individual rekeying and the Rebuild and Batch algorithms, and readers can refer to [13] for the analysis. Therefore, the interval-based rekeying algorithms can reduce the computation and communication costs of the a group is highly dynamic.

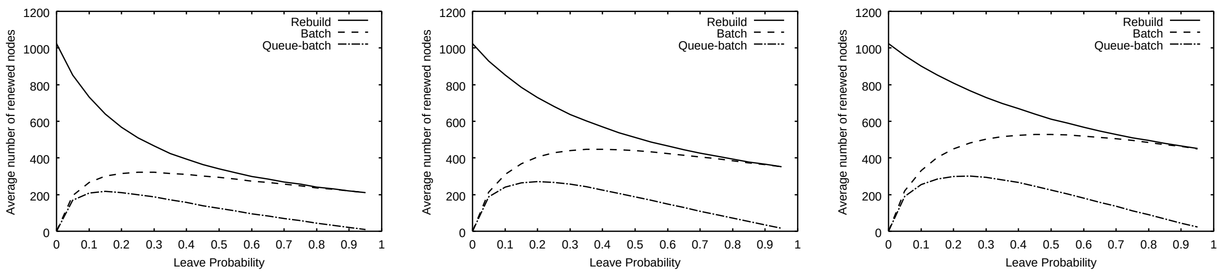
Experiment 2: (Average analysis at different fixed join probabilities) In this experiment, we examine the case where the key tree becomes unbalanced after many intervals of join and leave events. We vary the join probability p_J to be 0.25, 0.5, and 0.75. Then we evaluate the average performance measures of the three algorithms under various leave probabilities.

The results are illustrated in Figs. 15 and 16. We observe that Queue-batch outperforms the other two algorithms in terms of the costs of exponentiation and renewed nodes in most cases. The exception is that Queue-batch needs more exponentiations than Batch when the leave probability is low (smaller than 0.2). The reason is that attaching the subtree of new members to an existing tree with few leaves may make the key tree unbalanced, leading to more computations in subsequent rekeying intervals. Moreover, the performance of Rebuild is the worst when p_L is low, but approaches that of Batch when p_L is high (e.g., both algorithms have similar



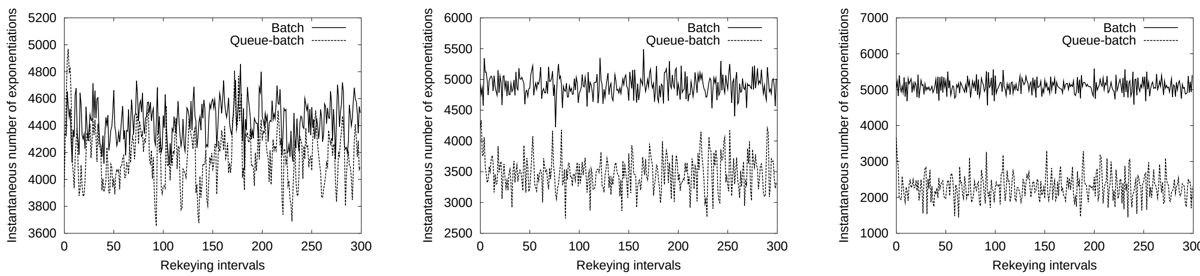
(a) $p_J = 0.25$ (b) $p_J = 0.5$ (c) $p_J = 0.75$

Fig. 15. Experiment 2: Average numbers of exponentiations of Rebuild, Batch, and Queue-batch at different fixed join probabilities.



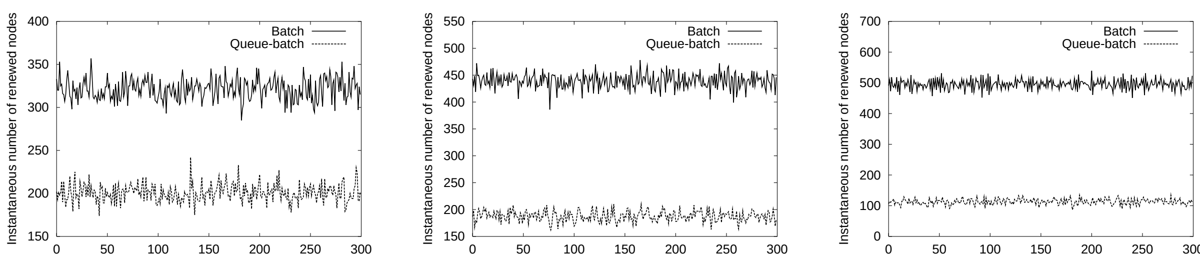
(a) $p_J = 0.25$ (b) $p_J = 0.5$ (c) $p_J = 0.75$

Fig. 16. Experiment 2: Average numbers of renewed nodes of Rebuild, Batch, and Queue-batch at different fixed join probabilities.



(a) $p_J = 0.25$, $p_L = 0.25$ (b) $p_J = 0.5$, $p_L = 0.5$ (c) $p_J = 0.75$, $p_L = 0.75$

Fig. 17. Experiment 3: Instantaneous numbers of exponentiations of Batch and Queue-batch at different pairs of join and leave probabilities.



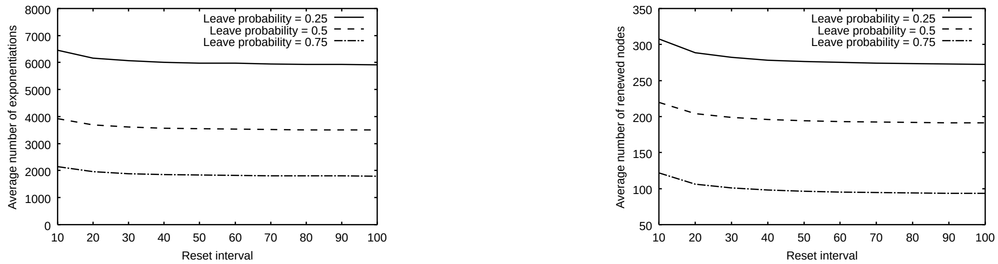
(a) $p_J = 0.25$, $p_L = 0.25$ (b) $p_J = 0.5$, $p_L = 0.5$ (c) $p_J = 0.75$, $p_L = 0.75$

Fig. 18. Experiment 3: Instantaneous numbers of renewed nodes of Batch and Queue-batch at different pairs of join and leave probabilities.

average numbers of exponentiations and renewed nodes when p_L is higher than 0.6 and 0.8, respectively). In most situations, Queue-batch outperforms the other two algorithms at different join and leave probabilities. This shows that the pre-processing of the join requests in Queue-batch can significantly reduce the computation and communication loads in rekeying.

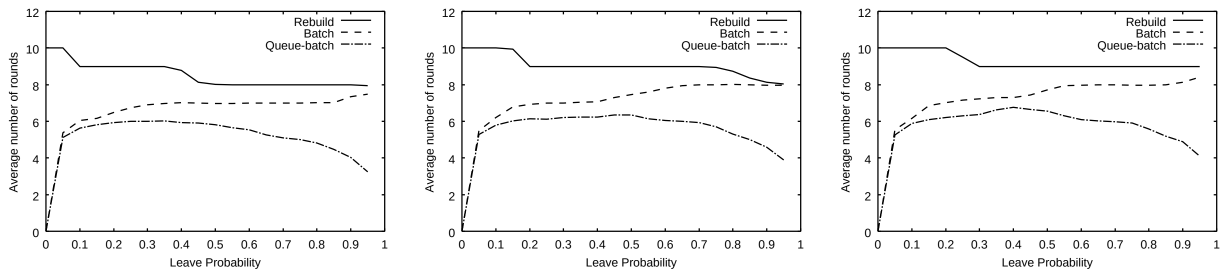
Experiment 3: (Instantaneous analysis at different pairs of join and leave probabilities) This experiment compares the instantaneous performance measures of Batch and Queue-batch over 300 rekeying intervals (we ignore Rebuild because it performs the worst among the three algorithms). We consider different pairs of p_J and p_L that represent different mobility characteristics of the peer group.

Fig. 17 illustrates the instantaneous numbers of exponentiations at different pairs of p_J and p_L. We note that when the group has high join and leave probabilities, Queue-batch sig-



(a) Average number of exponentiations at $p_J = 0.5$ (b) Average number of renewed nodes at $p_J = 0.5$

Fig. 19. Experiment 4: Average performance results of Queue-batch at different reset intervals.



(a) $p_J = 0.25$ (b) $p_J = 0.5$ (c) $p_J = 0.75$

Fig. 20. Experiment 5: Average numbers of rounds of Rebuild, Batch and Queue-batch at different join and leave probabilities.

nificantly outperforms the Batch algorithm. Fig. 18 illustrates the instantaneous numbers of renewed nodes. As compared to the Batch algorithm, Queue-batch has a much lower cost in terms of the number of renewed nodes when both join and leave probabilities are high. This implies that Queue-batch can reduce the communication cost significantly in a highly dynamic environment.

Experiment 4: (Performance analysis of Queue-batch at different reset intervals) Queue-batch does not reconstruct the whole key tree as Rebuild during rekeying. Thus the key tree may become unbalanced after some rekeying intervals. In this experiment, we consider how Queue-batch performs if we reconstruct the key tree using the Rebuild algorithm every T_R rekeying intervals, where T_R is called the *reset interval*. This approach keeps the tree balanced at the cost of executing the Rebuild algorithm. We fixed p_J = 0.5 and p_L = 0.25, 0.5, and 0.75, and ran the simulations over 1000 rekeying intervals.

Fig. 19 depicts that the performance of Queue-batch remains approximately constant even at high reset intervals, meaning that Queue-batch can still preserve its performance without reconstructing the key tree after a long period of rekeying. This shows the robustness of the Queue-batch algorithm in maintaining a relatively balanced tree. This property is important because it can reduce the average costs of exponentiations and renewed nodes in the system.

Experiment 5: (Analysis in terms of number of rounds) We define a *round* as the period during which the group members compute the secret keys as far up the key tree as they can. At the end of each round, all sponsors have to broadcast the blinded keys of the renewed nodes that have their secret keys computed so that other members can proceed with the secret key computations. In the analysis, we assume that rekeying is performed in *lock-step*, meaning that the two steps of secret key computations and blinded key broadcasts are carried out alternately.

Fig. 20 illustrates the average numbers of rounds required for Batch and Queue-batch. At high leave probabilities, Queue-batch saves three to four rounds as compared to Rebuild and Batch. The savings are due to the preprocessing of join requests at the Queue-subtree stage. A fewer number of rounds is preferred as less message overhead is involved in processing rekeying messages and storing message headers.

Discussion of the experimental results: The experiments show that the interval-based algorithms outperform the individual rekeying approach in terms of both computation and communication costs and that Queue-batch performs the best among the three interval-based algorithms. From both mathematical analysis and simulations, Queue-batch performs much better when the join and leave events occur very frequently. It also demonstrates its robustness in keeping the key tree balanced and its capability in minimizing the number of rounds required to update the group key.

To understand why Queue-batch performs much better than the other two algorithms when the group is highly dynamic, we consider two cases: frequent joins and frequent leaves. When the number of join events is high, Queue-batch benefits significantly from the pre-processing of join events in the Queue-subtree phase. In addition, when the number of leave events is high, Queue-batch reduces the heights of the existing tree nodes through node pruning. Batch, however, replaces the leaving leaf nodes with the joining ones and hence preserves the heights of tree nodes. This distinction implies that Queue-batch requires fewer rekeying operations for the members whose associated leaf nodes are promoted to shallow positions. As a result, the performance gain of Queue-batch is more significant in the presence of frequent membership events.

C. Centralized vs. Decentralized Group Key Management

In this subsection, we compare the performance of our interval-based algorithms and a centralized key management approach.

We consider the centralized logical key tree (LKT) approach [14] upon which our Batch algorithm is developed (see Section III-B). Similar to the interval-based algorithms, we assume that the key tree in the LKT approach is a binary tree and that the keys of the tree nodes are computed based on the TGDH protocol [11]. Among the group members, we select a *group controller* that centrally renews the group key at periodic rekeying intervals. We assume that the group controller knows the keys of all the nodes in the key tree and that a newly joining member sends its individual secret key to the group controller via a secure channel. At the beginning of a rekeying interval, the group controller first rekeys all (non-leaf) renewed nodes using the TGDH protocol. It then encrypts the updated secret key of each renewed node with the respective secret keys of the *two* child nodes via any symmetric encryption algorithm. Afterward, it broadcasts the encrypted keys to the group. Every member, upon receiving the encrypted keys, decrypts the keys along its key path with the secret keys it holds.

We use Fig. 7 to illustrate the LKT approach. Suppose that M_1 is the group controller. It first rekeys all renewed nodes as in the TGDH protocol. It then broadcasts to the group the following encrypted keys: $\{(K_0)_1, (K_0)_2, (K_1)_3, (K_1)_4, (K_2)_5, (K_2)_6, (K_5)_{11}, (K_5)_{12}\}$, where $(K_i)_j$ refers to the secret key K_i of parent node i encrypted by the secret key K_j of the child node j. For example, in order for member M_3 to obtain the group key K_0, it first decrypts $(K_1)_4$ with K_4, followed by $(K_0)_1$ with K_1.

We assume that the group controller broadcasts the encrypted keys using the view-synchronous communication model as in the interval-based algorithms. While the group controller can broadcast the encrypted keys via the point-to-multipoint multicast, such a communication model has two limitations. First, if the underlying group communication is multipoint-to-multipoint-based such that every member can be a sender, setting up an extra multicast channel will be an overhead. Also, it is possible for the group controller to leave the group. All other members have to detect the group controller's departure and install a new membership view in order to select another group controller. The group controller's departure, however, cannot be detected with the unilateral point-to-multipoint multicast [11]. We point out that view synchrony is essential for reliable multipoint-to-multipoint group communication regardless of which group key management approach is being used [11].

The performance of the LKT approach can be quantified as follows. Since the encryption of the updated secret keys can be achieved with any inexpensive symmetric encryption algorithm, the computation cost of the LKT approach is mainly due to the exponentiation operations of renewing the secret keys. Furthermore, since the group controller uses the view-synchronous communication model as in the interval-based algorithms, the communication costs of the LKT approach and our interval-based algorithms mainly differ by the number of keys (either encrypted or blinded) broadcast to the group and the number of rounds required to generate the group key.

	LKT	Queue-batch
Number of exponentiations	$\mathcal{R}$	$10\mathcal{R} - 30\mathcal{R}$
Number of broadcast keys	$2\mathcal{R}$	$\mathcal{R}$
Number of rounds	1	$4-6$

TABLE I

PERFORMANCE COMPARISON OF LKT AND QUEUE-BATCH.

Table I compares the performance metrics of both LKT and Queue-batch schemes, where $\mathcal{R}$ denotes the number of renewed nodes, and the numbers of exponentiations and rounds for Queue-batch are estimated based on the simulation results in Section IV-B. As compared to our interval-based algorithms, the LKT approach requires fewer exponentiations and only one round to update the group key. However, all its exponentiations are carried out within the single group controller, while our interval-based algorithms scatter the computational load among all group members. In addition, the LKT approach broadcasts two encrypted copies of the updated secret key for each renewed node, while the interval-based algorithms broadcast only the corresponding blinded key. This implies that the interval-based algorithms save half of the number of broadcast keys. More importantly, the interval-based algorithms have better fault tolerance by eliminating the single-point-of-failure problem inherent in the centralized LKT approach. Issues of how the interval-based algorithms can recover from node failures in actual implementation are discussed in Section VI.

V. AUTHENTICATED TGDH

In this section, we propose the *Authenticated Tree-Based Group Diffie-Hellman* (A-TGDH) protocol that provides key authentication for our interval-based algorithms. The idea is to couple the session-based group key with the certified permanent private components of the group members. Each member holds two types of keys: *short-term secret* and *blinded* keys as well as *long-term private* and *public* keys. Short-term keys are randomly generated when a member joins the group and become expired when the member leaves, while long-term keys remain permanent across many sessions and are certified by a trusted CA. Our protocol seeks to satisfy several requirements that are crucial for key establishment [3]: (i) *perfect forward secrecy* (i.e., the compromise of long-term keys does not degrade the secrecy of past short-term keys), (ii) *known-key security* (i.e., the compromise of past short-term keys does not degrade the secrecy of future short-term keys), and (iii) *key authentication* (i.e., all group members are assured that no outsiders can identify the group key). Also, our protocol can be implemented in a way that satisfies *key confirmation* (i.e., all group members are assured that every other member holds the same group key).

A. Description of A-TGDH

We first define the notation. As stated in Section II, every node v in the key tree is associated with a secret key K_v and a blinded key BK_v. We then construct the *blinded key set* BK'_v, which, in general, refers to a number of copies BK_v's respectively encrypted by the long-term private component of every descendant member of the sibling of node v (the mathematical formulation of BK'_v is presented below). The set of the descendant members of node v is given by $\boldsymbol{M}_v$. The ith member, M_i, holds a short-term secret key r_{M_i} and the corresponding blinded key $\alpha^{r_{M_i}} \bmod p$, as well as a long-term private key x_{M_i} and the corresponding public key $\alpha^{x_{M_i}} \bmod p$, where all arithmetic operations are to be performed on the cyclic group of prime order p with generator α. For brevity, we omit the term "mod p" in the following description.

We assume that each member has acquired the certificates of all other members and hence their long-term public keys from a trusted CA *prior to* the key agreement process.

The A-TGDH protocol is based on the two-party, two-pass authenticated key-agreement (AK) protocol in [18]. Given two parties, say M_1 and M_2, the two-pass AK protocol works as follows: M_1 sends $\alpha^{r_{M_1}}$ to M_2 and M_2 sends $\alpha^{r_{M_2}}$ to M_1. M_1 computes $(\alpha^{x_{M_2}})^{r_{M_1}} \cdot (\alpha^{r_{M_2}})^{r_{M_1}+x_{M_1}}$ = $\alpha^{r_{M_1}r_{M_2}+r_{M_1}x_{M_2}+r_{M_2}x_{M_1}}$. Analogous operations are performed by M_2. The agreed session key is then given by $K = \alpha^{r_{M_1}r_{M_2}+r_{M_1}x_{M_2}+r_{M_2}x_{M_1}}$.

The two-pass AK protocol offers a number of advantages. It involves only two passes and thus saves communication cost. It achieves key authentication and known-key security [18]. If it is incorporated with key confirmation, then it gives perfect forward secrecy as well [4].

We next extend the two-party, two-pass AK protocol to our proposed A-TGDH protocol. In A-TGDH, we associate a node v with K_v and BK'_v as follows:
case 1) If node v is a non-leaf node with child nodes $2v+1$ and $2v+2$ (assume $v \neq 0$ since we need not broadcast the blinded group key):

$$K_v = \alpha^k \bmod p, \text{ where}$$
$$k = K_{2v+1}K_{2v+2} + K_{2v+1}\sum_{M_i \in \boldsymbol{M}_{2v+2}} x_{M_i} + K_{2v+2}\sum_{M_i \in \boldsymbol{M}_{2v+1}} x_{M_i} \quad (9)$$

$$BK'_v = \begin{cases} \{\alpha^{K_v x_{M_i}} \bmod p : M_i \in \boldsymbol{M}_{v+1}\} \\ \quad \text{if node } v \text{ is the left child of its parent} \\ \{\alpha^{K_v x_{M_i}} \bmod p : M_i \in \boldsymbol{M}_{v-1}\} \\ \quad \text{if node } v \text{ is the right child of its parent.} \end{cases} \quad (10)$$

case 2) If node v is a leaf node associated with member M_i:

$$K_v = r_{M_i} \quad (11)$$
$$BK'_v = \{\alpha^{r_{M_i}} \bmod p\}. \quad (12)$$

Thus, if a given node v needs to be renewed, a sponsor can simply broadcast BK'_v according to one of our interval-based algorithms. Also, any member can still include its short-term blinded key (i.e., the blinded key of its corresponding leaf node) in its join request.

To illustrate how A-TGDH works, we consider a possible key tree formed after the rekeying process as shown in Fig. 21. Nodes 0, 1, and 2 are renewed nodes. Also, M_1 and M_3 are chosen to be the sponsors. Hence, the members perform the following steps:

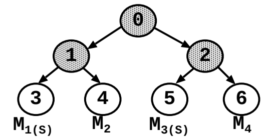
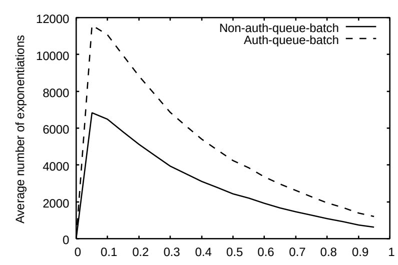


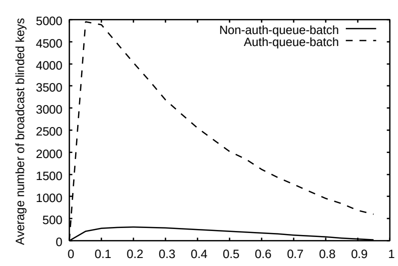
Fig. 21. Example of authenticated key agreement involving 4 members.

- Since the blinded keys of leaf nodes are $\alpha^{r_{M_i}}$, for $i = 1, 2, 3$ and 4, the secret keys of nodes 1 and 2 are computed as $K_1 = \alpha^{r_{M_1}r_{M_2}+r_{M_1}x_{M_2}+r_{M_2}x_{M_1}}$ and $K_2 = \alpha^{r_{M_3}r_{M_4}+r_{M_3}x_{M_4}+r_{M_4}x_{M_3}}$.
- The sponsor M_1 broadcasts $\alpha^{K_1 x_{M_3}}$ and $\alpha^{K_1 x_{M_4}}$, and the sponsor M_3 broadcasts $\alpha^{K_2 x_{M_1}}$ and $\alpha^{K_2 x_{M_2}}$.
- M_1 and M_2 can retrieve α^{K_2} from $\alpha^{K_2 x_{M_1}}$ and $\alpha^{K_2 x_{M_2}}$, respectively. Similarly, M_3 and M_4 can retrieve α^{K_1}. Therefore, the members can compute the resulting group key as $K_0 = \alpha^{K_1K_2+K_1(x_{M_3}+x_{M_4})+K_2(x_{M_1}+x_{M_2})}$.

Using the same experimental setting as in Section IV-B, we compare the non-authenticated and authenticated Queue-batch algorithms for a population of size 1024 with a fixed join probability $p_J = 0.25$. Fig. 22 plots the average number of exponentiations of computing K_v and BK_v as well as the average number of blinded key copies BK_v broadcast to the group for all renewed nodes v. It shows that the authenticated version requires about twice the exponentiations and more than 10 times the blinded key copies as compared to the non-authenticated one. Thus, the use of authentication is subject to the trade-off between security and performance.



(a) Average number of exponentiations



(b) Average number of broadcast blinded keys

Fig. 22. Comparison between the non-authenticated and authenticated Queue-batch algorithms at $p_J = 0.25$.

B. Key Confirmation

Key confirmation should be enforced to guarantee all group members are actually holding the same group key. Providing complete key confirmation requires every member to demonstrate its knowledge of the group key to all other members.

One way to achieve this is to ask every member to broadcast the one-way function result of every newly generated group key. However, this involves many broadcasts and is not scalable. In another approach proposed in [9], each member only needs to prove its knowledge of the group key to its neighbors, provided that all members are arranged in a ring. However, such an approach is vulnerable to the collusion attack [9].

We suggest two feasible implementation approaches that support a moderate level of key confirmation. In the first approach, we divide a group into several subgroups such that members only confirm the group key with others *within the same subgroup* via broadcast. The subgroup size and the number of subgroups are chosen depending on the desired level of security. In an alternative approach, we appoint a sponsor to broadcast the blinded group key so that every other member can verify if its computed blinded group key is identical to the one it receives. If a member finds that the keys are different, then it will report the error. This approach is similar to that in [3] except that it is applied to our tree-based setting. Our implementation chooses the latter approach. Section VI discusses the implementation details.

C. Security Analysis

A-TGDH satisfies our stated security goals with the following assumptions. Since key confirmation is essential for achieving perfect forward secrecy [4], we assume that it has been implemented as described in Section V-B. Also, we assume that there exists only a passive adversary E that monitors the flow of blinded key messages. We further assume that E cannot solve the Diffie-Hellman problem [6] (i.e., given only α, p, $\alpha^x \bmod p$, and $\alpha^y \bmod p$, it is infeasible for E to compute $\alpha^{xy} \bmod p$) and the discrete logarithm problem (i.e., given only α, p, and $\alpha^x \bmod p$, it is infeasible for E to compute x). The following proof is based on [3], [15].

Theorem: *A-TGDH satisfies perfect forward secrecy, known-key security, and key authentication.*

Proof Sketch:

1) Perfect forward secrecy. We want to prove that the authenticated short-term keys of all non-leaf nodes remain secret even the long-term keys are compromised. We prove this property by induction on the levels of the tree which has the lowest level h.

- **Basis**. Consider a non-leaf node v_o at level $h-1$ whose children are both leaf nodes associated with members M_{i1} and M_{i2}. Given the long-term private keys $x_{M_{i1}}$ and $x_{M_{i2}}$, the adversary E cannot compute $K_{v_o} = \alpha^{r_{M_{i1}} r_{M_{i2}} + r_{M_{i1}} x_{M_{i2}} + r_{M_{i2}} x_{M_{i1}}}$, since computing $\alpha^{r_{M_{i1}} r_{M_{i2}}}$ without the knowledge of $r_{M_{i1}}$ or $r_{M_{i2}}$ is infeasible.
- **Induction hypothesis**. Suppose the keys of nodes $2v+1$ and $2v+2$ at some level l, where $0 < l \leq h-1$, remain secret despite the compromise of long-term keys.
- **Induction step**. Consider the node v at level $l-1$. Given only the long-term private keys, we cannot deduce K_{2v+1} and K_{2v+2} (by hypothesis). This implies K_v cannot be computed as it contains the component $\alpha^{K_{2v+1} K_{2v+2}}$.

Thus, by induction, E cannot compute the secret keys (including the group key) of any one of the non-leaf nodes given only the long-term private keys. Perfect forward secrecy is achieved.

The remaining properties can also be proved by induction, although we omit the inductive proofs for brevity.

2) Known-key security. It should be noted that the authenticated group key K_0 consists of a secret random component equivalent to the group key of the non-authenticated TGDH. If E compromises this authenticated group key K_0, then it cannot compute the past group keys whose corresponding secret random components are composed of the short-term secrets r_{M_i}'s offered by different combinations of members, and doing so will require E to solve the Diffie-Hellman problem. If any two past group keys refer to the same set of members, then they are still different since each member M_i renews r_{M_i} when it re-joins the group.

3) Key authentication. To determine K_{v_o} for a non-leaf node v_o whose children are both leaf nodes corresponding to members M_{i_1} and M_{i_2}, the adversary E has to know $\alpha^{r_{M_{i_1}} r_{M_{i_2}}}$. However, E only observes $\alpha^{r_{M_{i_1}}}$ and $\alpha^{r_{M_{i_2}}}$. Thus, it is infeasible for E to solve the Diffie-Hellman problem for $\alpha^{r_{M_{i_1}} r_{M_{i_2}}}$. On the other hand, to determine K_v for a non-leaf node v which contains at least one non-leaf child node, say node $2v+1$, E has to know $\alpha^{K_{2v+1} K_{2v+2}}$. However, E cannot identify K_{2v+1} from the blinded key messages due to the intractability of the discrete logarithm problem (i.e., given only $\alpha^{K_{2v+1} x_{M_i}}$ and $\alpha^{x_{M_i}}$, it is infeasible to compute K_{2v+1}). Therefore, A-TGDH provides key authentication. ∎

VI. Implementation

We implemented the *SEcure Communication Library (SEAL)* to realize the interval-based algorithms described in Section III and to offer a programming interface for the development of a secure group-based application. In the application, a group member first invokes `SEAL_init` to initialize a *SEAL session object* that holds all necessary components of the library. Using `SEAL_join`, it joins a group and presents its certificate obtained from a certificate authority to the entire group for identification. It can then send and receive encrypted data messages with `SEAL_send` and `SEAL_recv`, respectively, or read the membership status with `SEAL_read_membership`. It leaves the group with `SEAL_leave`. It can later either re-join another or the same group, or terminate by destroying the SEAL session object with `SEAL_destroy`. Fig. 23 depicts the flowchart of using the library.

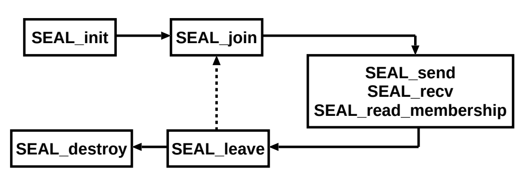


Fig. 23. Flowchart of using SEAL in secure group-based applications.

SEAL is built upon the *Spread* toolkit [2], which provides view-synchronous message delivery. Every member connects to a Spread daemon, which maintains an active TCP connection to all other Spread daemons and keeps track of the current membership status of the communication group. When a member joins or leaves the group, the associated Spread daemon notifies other daemons to flush the remaining messages to the original membership view and to block the transmission of new messages until all Spread daemons and existing group members install the updated membership view. Similarly, if a Spread daemon fails, the associated members are removed from the membership view by the remaining Spread daemons. Therefore, every existing group member always holds the latest membership view. Also, all messages are originated from the sender and delivered to all members under the same membership view, or equivalently between two consecutive membership events. To ensure the ordered delivery, the Spread daemons append a timestamp to every transmitted message.

SEAL addresses the assumptions made in Section III. Given the agreed-upon membership view provided by Spread daemons, all group members select the one that stays the longest in the group as the *leader*, which is responsible for synchronizing the rekeying operations. The leader periodically broadcasts a *rekeying message* to notify other members to start the rekeying operation. To enable new members to construct the current key tree, each rekeying message includes the existing key tree as well as the join and leave requests in the last rekeying interval. Note that the leader is not a centralized key server that generates the group key, so the contributory requirement of our proposed algorithms still holds. Upon receiving the rekeying message, each member updates its own key tree and checks whether it is a sponsor. Any member that becomes the sponsor will broadcast the updated blinded keys based on the *sponsor-coordination algorithm*, which ensures that each updated blinded key is broadcast only once and that no extra communication is involved in the coordination among the sponsors. Each member then caches any received blinded keys and computes the new secret keys along its key path. Finally, one of the sponsors will broadcast the blinded group key. Every other member then verifies the blinded group key with the one it has computed (see Section V-B). If this key confirmation process succeeds, then the rekeying operation is finished.

We point out that the leader and sponsors can leave the group during a rekeying operation, and their departures can make the rekeying operation fail to complete. In SEAL, if the communication group detects a change of the membership view before the completion of the rekeying operation (i.e., the group key is not yet confirmed), it first elects a new leader (if necessary), and the leader will broadcast another rekeying message immediately to restart the rekeying operation. Any renewed nodes whose blinded keys have not yet been broadcast remain renewed in the new rekeying operation. Such a *self-stabilizing* property [11] is realized in SEAL.

Here, we implicitly assume that the Spread daemons always provide trusted membership views. Maintaining an authenticated membership view involves the change of implementation in Spread and is not the focus of this paper. We pose this problem as future work.

We experimented the performance of the interval-based algorithms based on SEAL under various join and leave dynamics. When there are 40 group members connected via eight Spread daemons in a local area network, the rekeying time generally takes less than one second. We refer readers to reference [12] for further discussion.

VII. Related Work

To achieve secure group communication, Wallner *et al.* [21] and Wong *et al.* [22] propose the key tree approach that associates keys in a hierarchical tree and rekeys at each join or leave event. Li *et al.* [14] and Yang *et al.* [23] then apply the periodic rekeying concept in Kronos [16] to the key tree setting. All the key-tree-based approaches [14], [21], [22], [23] require a centralized key server for key generation.

References [5], [10], [11], [19] extend the Diffie-Hellman protocol [6] to group key agreement schemes for secure communications in a decentralized network. Burmester and Desmedt [5] propose a computation-efficient protocol at the expense of high communication overhead. Steiner *et al.* [19] propose *Cliques*, in which every member introduces its key component into the result generated by its preceding member and passes the new result to its following member. Cliques is efficient in rekeying for leave or partition events, but imposes a high workload on the last member in the chain. Kim *et al.* [11] propose TGDH, which arranges keys in a tree structure (see Section II for details). The setting of TGDH is similar to that of the One-Way Function Tree (OFT) scheme [17] except that TGDH uses Diffie-Hellman instead of one-way functions for the group key generation. Kim *et al.* [10] also suggest a variant of TGDH called STR which minimizes the communication overhead by trading off the computation complexity. All the above schemes are decentralized and hence avoid the single-point-of-failure problem in the centralized case, though they introduce high message traffic due to distributed communication. References [10], [11], [19] consider rekeying at single join, single leave, merge, or partition events. Our work considers a more general case that consists of a batch of join and leave events.

Comparison between the centralized and decentralized rekeying is studied by Amir *et al.* [1] and Waldvogel *et al.* [20]. In particular, Amir *et al.* [1] suggest a centralized key distribution scheme based on Cliques [19] and compare the performance of both schemes. In contrast, our work compares the centralized and decentralized key management schemes adapted from a key tree setting.

Rather than emphasize the rekeying efficiency, references [3], [9], [15] focus on the security issues and develop authenticated group key agreement schemes based on the Burmester-Desmedt model, Cliques, and TGDH, respectively. For instance, the AGKA-G protocol [15] is an extension of the two-party Günther scheme [8] to the TGDH protocol. Our A-TGDH protocol is an authenticated version of our interval-based algorithms.

VIII. Conclusion

We consider several distributed collaborative key agreement protocols for dynamic peer groups. The key agreement setting is performed in which there is no centralized key server to maintain or distribute the group key. We show that one can use the TGDH protocol to achieve such distributive and collaborative key agreement. To reduce the rekeying complexity, we propose to use an interval-based approach to carry out rekeying for multiple join and leave requests at the same time, with a trade-off between security and performance. In particular, we show that the Queue-batch algorithm can significantly reduce both computation and communication costs when there exist highly frequent membership events. We also address both authentication and implementation for the interval-based key agreement algorithms.

REFERENCES

[1] Y. Amir, Y. Kim, C. Nita-Rotaru, J. L. Schultz, J. Stanton, and G. Tsudik. Secure Group Communication Using Robust Contributory Key Agreement. *IEEE Transactions on Parallel and Distributed Systems*, 15(5):468–480, May 2004.

[2] Y. Amir and J. Stanton. The Spread Wide Area Group Communication System. CNDS-98-4, Johns Hopkins University, 1998.

[3] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated Group Key Agreement and Friends. In *Proc. of 5th ACM Conference on Computer and Communications*, Nov 1998.

[4] S. Blake-Wilson and A. Menezes. Authenticated Diffie-Hellman Key Agreement Protocols. In *Proc. of the 5th Annual Workshop on Selected Areas in Cryptography*, 1998.

[5] M. Burmester and Y. Desmedt. A Secure and Efficient Conference Key Distribution System. *Advances in Cryptology – EUROCRYPT '94*, 1995.

[6] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[7] A. Fekete, N. Lynch, and A. Shvartsman. Specifying and Using a Partionable Group Communication Service. In *ACM Symposium on Principles of Distributed Computing (PODC)*, Aug 1997.

[8] C. G. Günther. An Identity-Based Key Exchange Protocol. In *EUROCRYPT*, 1989.

[9] M. Just and S. Vaudenay. Authenticated Multi-Party Key Agreement. In *Advances in Cryptology ASIACRYPT '96*, pages 36–49. LNCS 1163, Springer-Verlag, 1996.

[10] Y. Kim, A. Perrig, and G. Tsudik. Communication-Efficient Group Key Agreement. In *Proceedings of the 17th International Information Security Conference IFIP SEC'01*, Nov 2001.

[11] Y. Kim, A. Perrig, and G. Tsudik. Tree-Based Group Key Agreement. *ACM Trans. on Information and System Security*, 7(1):60–96, Feb 2004.

[12] P. Lee. Distributed and Collaborative Key Agreement Protocols with Authentication and Implementation for Dynamic Peer Groups. MPhil Thesis, The Chinese University of Hong Kong, June 2003.

[13] P. P. C. Lee, J. C. S. Lui, and D. K. Y. Yau. Distributed Collaborative Key Agreement and Authentication Protocols for Dynamic Peer Groups. CS&E Technical Report, The Chinese University of Hong Kong, Jul 2005.

[14] X. S. Li, Y. R. Yang, M. G. Gouda, and S. S. Lam. Batch Rekeying for Secure Group Communications. In *Proc. of Tenth International World Wide Web Conference (WWW10)*, May 2001.

[15] A. Perrig. Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication. In *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99)*, July 1999.

[16] S. Setia, S. Koussih, and S. Jajodia. Kronos: A Scalable Group Re-Keying Approach for Secure Multicast. In *Proc. of IEEE Symposium on Security and Privacy*, May 2000.

[17] A. T. Sherman and D. A. McGrew. Key Establishment in Large Dynamic Groups Using One-Way Function Trees. *IEEE Trans. Software Eng.*, 29(5):444–458, May 2003.

[18] B. Song and K. Kim. Two-Pass Authenticated Key Agreement Protocol with Key Confirmation. In *Proc. of IndoCrypt*, volume LNCS vol. 1977, pages 237–249, Dec 2000.

[19] M. Steiner, G. Tsudik, and M. Waidner. Key Agreement in Dynamic Peer Groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, Aug 2000.

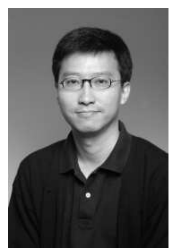
[20] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The VersaKey Framework: Versatile Group Key Management. *IEEE Journal on Selected Areas in Communications*, 17(9):1614–1631, Sep 1999.

[21] D. M. Wallner, E. J. Harder, and R. C. Agee. Key Management for Multicast: Issues and Architectures. Internet draft draft-wallner-key-arch-00.txt, Internet Engineering Task Force, Jul 1997.

[22] C. K. Wong, M. Gouda, and S. S. Lam. Secure Group Communications Using Key Graphs. *IEEE/ACM Trans. on Networking*, 8(1):16–30, Feb 2000.

[23] Y. R. Yang, X. S. Li, X. B. Zhang, and S. S. Lam. Reliable Group Rekeying: A Performance Analysis. *Proc. of ACM SIGCOMM*, August 2001.

Patrick P. C. Lee received his B.Eng. (first-class honors) in Information Engineering in July 2001 and M.Phil. in Computer Science and Engineering in July 2003, both from the Chinese University of Hong Kong. He is now a Ph.D. student at Columbia University, New York, NY. His research interests are in distributed network algorithms and network security.



John C. S. Lui was born in Hong Kong. He received his Ph.D. in Computer Science from UCLA. He worked at the IBM T.J. Watson Research Laboratory and the IBM Almaden Research Laboratory/San Jose Laboratory after his graduation. He participated in various research and development projects on file systems and parallel I/O architectures. John later joined the Department of Computer Science and Engineering at the Chinese University of Hong Kong. For the past several summers, he has been a visiting professor in computer science departments at UCLA, Columbia University, University of Maryland at College Park, Purdue University, University of Massachusetts at Amherst and Universit degli Studi di Torino in Italy. His research interests span both in system and in theory/mathematics. His current research interests are in theoretic/applied topics in data networks, distributed multimedia systems, network security, OS design issues and mathematical optimization and performance evaluation theory. John received various departmental teaching awards and the CUHK Vice-Chancellor's Exemplary Teaching Award in 2001. He is an associate editor in the Performance Evaluation Journal, member of ACM, a senior member of IEEE and an elected member in the IFIP WG 7.3. John was the TPC co-chair of ACM Sigmetrics 2005. His personal interests include films and general reading.

David K. Y. Yau received the B.Sc. (first class honors) degree from the Chinese University of Hong Kong, and the M.S. and Ph.D. degrees from the University of Texas at Austin, all in computer sciences. From 1989 to 1990, he was with the Systems and Technology group of Citibank, NA. He was the recipient of an IBM graduate fellowship, and is currently an Associate Professor of Computer Sciences at Purdue University, West Lafayette, IN. He received an NSF CAREER award in 1999, for research on network and operating system architectures and algorithms for quality of service provisioning. His other research interests are in network security, value-added services routers, and mobile wireless networking. David is a member of the ACM and IEEE, and serves on the editorial board of the IEEE/ACM Transactions on Networking.