Distributed Collaborative Key Agreement Protocols for Dynamic Peer Groups*

Patrick P. C. Lee
Department of Computer Science & Engineering
The Chinese University of Hong Kong
pclee@cse.cuhk.edu.hk

John C. S. Lui
Department of Computer Science & Engineering
The Chinese University of Hong Kong
cslui@cse.cuhk.edu.hk

David K. Y. Yau
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
yau@cs.purdue.edu

Abstract

We consider several distributed collaborative key agreement protocols for dynamic peer groups. This problem has several important characteristics which make it 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 can leave the group while new members may join. Instead of performing individual rekey operations, i.e., recomputing the group key after every join or leave request, we consider an interval-based approach of rekeying. In particular, we consider three distributed algorithms for updating the group key: (1) the* Rebuild algorithm*, (2) the* Batch algorithm*, and (3) the* Queue-batch algorithm*. Performance of these distributed algorithms under different settings, such as different join and leave probabilities, is analyzed. We show that these three distributed algorithms significantly outperform the individual rekey algorithm, and that the Queue-batch algorithm performs the best among the three distributed algorithms. Moreover, the Queue-batch algorithm has the intrinsic property of balancing the computation/communication workload such that the dynamic peer group can quickly begin secure group communication. This provides a fundamental understanding about establishing a collaborative group key for a distributed dynamic peer group.*

1. Introduction

With the emergence of many group-oriented distributed applications such as multi-player games and tele/video-conferencing, 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. For example, consider a group of people in a peer-to-peer ad hoc network having a closed and confidential business meeting. Since they have not previously agreed upon a common secret key, communication between group members is susceptible to eavesdropping. To solve the problem, we need a *secure distributed group key agreement protocol* such that the group of people can establish the 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 more traditional centralized group key distribution 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 [10, 11, 7, 6]. 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). For distributed key agreement protocols, however, no centralized key server is available. This arrangement is justified in many situations – e.g., in a peer-to-peer or ad hoc network where centralized resources are not readily available. Moreover, an advantage of distributed protocols over the centralized protocols is the increase in system reliability, since 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 [2]. In the protocol, members X and Y use a cyclic group $\mathcal{G}$ of prime

* J. Lui was supported in part by the Mainline and RGC Research Grant; D. Yau was supported in part by the National Science Foundation under grant numbers CCR-9875742 (CAREER) and EIA-9806741.

order p and a generator α. They can generate their secret components 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 consider a dynamic communication group in which members are located in a distributed fashion. We extend the Diffie-Hellman key exchange protocol to more than two members in the communication group. The membership of the communication group is dynamic so that members can leave and new members can join the group at any time. The contributions of our work are:

- The key agreement protocol is distributed in nature and does not require a centralized key server.
- The key agreement protocol is contributory – each member contributes its part to the overall group key.
- We illustrate that instead of performing individual rekeying operations, one can use an interval-based approach to significantly reduce the computation and communication costs of maintaining the group key.
- We propose three distributed interval-based rekey protocols, and carry out quantitative and simulation-based analysis to illustrate their performance merits.

The balance of the paper is organized as follows. In Section 2, we provide the background of the Diffie-Hellman protocol. We explain how it can be extended to a Tree-Based Group Diffie-Hellman protocol so that it can accommodate more than two members in a dynamic peer group. In Section 3, we present three interval-based distributed algorithms to reduce the computation and communication costs for maintaining the group key in a dynamic peer group. We also carry out mathematical analysis to quantify system performance according to given performance metrics when the original Diffie-Hellman tree is a completely balanced tree. In Section 4, we report several experiments that illustrate the system cost under dynamic joins and leaves, for various system parameters (e.g., join/leave probabilities). We discuss related work in Section 5. Section 6 concludes.

2. 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 [3]. Each member maintains a set of keys. The keys are arranged in a hierarchical *binary tree structure*. 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 arithmetics are performed in a group of order p with generator α. Thus, 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 represents the individual secret key of a member and is uniquely defined by a group member M_i. Every member holds all the secret keys along the 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*. Figure 1 illustrates a possible key tree with six members M_1 to M_6. For example, member M_1 holds the keys in nodes 7, 3, 1 and 0. The secret key at node 0 is the group key of the peer group.

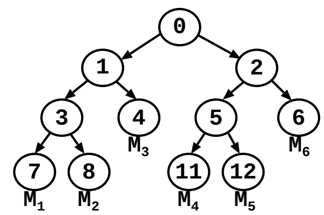


Figure 1. Tree-Based Group Diffie-Hellman

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 $2v+1$ and $2v+2$. Based on the Diffie-Hellman protocol [2], 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 keys at the non-leaf nodes, the secret key at a leaf node is selected by its associated member in the communication group. The key selection can be achieved through a secure pseudo random number generator [8]. 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 group membership in Figure 1. Every member M_i will generate 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 Equation (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 Equation (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 between any members in the group can be encrypted based on the group key (or secret key) K_0.

To ensure backward and forward confidentiality, *rekeying* is performed whenever there is any change in the group membership. This includes any new member joining or any existing member leaving the group. Let us first consider individual rekeying, meaning that rekeying is carried out for each single join or leave event. Before the group membership is changed, a special member called the *sponsor* is elected, and the sponsor is responsible for updating the keys held by the new members (in the join case) or departed members (in the leave case). We use the convention that the rightmost member under the subtree rooted at the sibling of the join/leave nodes takes the sponsor role. Note that the existence of the sponsor does not violate the requirement of the decentralized management as the group key generation still requires the contribution of all members in the group.

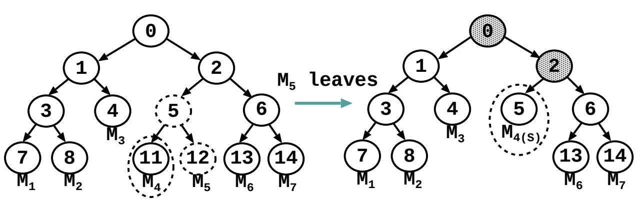


Figure 2. Rekeying at a single leave

Figure 2 illustrates a member leave event. Suppose that the member M_5 leaves the system, node 11 will be *promoted* to node 5, and member M_4 will be the sponsor. M_4 needs to rekey the secret keys K_2 and K_0, and broadcasts the blinded keys BK_2 and BK_5 to all the members. Upon receiving the blinded key BK_2, M_1, M_2 and M_3 can compute the group key K_0. Members M_6 and M_7, upon receiving BK_5, can compute K_2 and then the group key K_0.

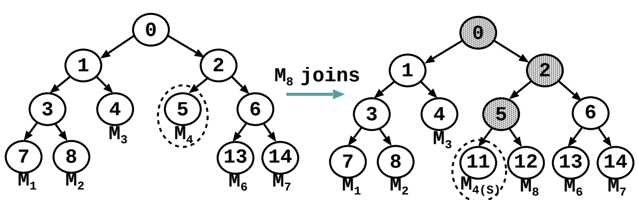


Figure 3. Rekeying at a single join

Figure 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 a 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). Figure 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 rekeys K_5, K_2 and K_0 in its local memory, 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 keys along their key paths and obtain the new group key.

Based on the above leave/join events in Figures 2 and 3, we find that we can *reduce* one rekeying operation if we could 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. Members carry out rekeying operations at regular *rekey intervals*. The motivation is to improve system performance. Security becomes weaker since, for instance, a departed user could still access data until the next rekeying interval. However, the tradeoff may be acceptable to practical applications, since we can adjust the rekeying interval according to application requirements. In the following section, we describe three interval-based distributed algorithms.

3. Interval-Based Distributed Algorithms

3.1. Description of Algorithms

In this subsection, we present three interval-based distributed rekeying algorithms. They are the *Rebuild algorithm*, the *Batch algorithm* and the *Queue-batch algorithm*. The use of interval-based rekeying aims to maintain good rekeying performance independent of the dynamics of joins and leaves. The three distributed algorithms are developed based on the following assumptions:

- The key tree of TGDH is used as a foundation of all the algorithms.
- The rekeying operations are carried out at the beginning of every rekey interval. There exists a *virtual queue* holding all join and leave requests till the beginning of the next rekey interval.
- When a new member sends a join request, it should also include its individual blinded key.
- For simplicity, all clients know the existing key tree structure and they also know all the blinded keys within the tree.
- The group members would elect sponsors to be responsible for computing and broadcasting blinded keys. To

obtain the blinded keys of the renewed nodes (a node is said to be *renewed* if it is a non-leaf node and its associated keys are updated), 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. In this case, we assume that the sponsors can coordinate with one another such that the blinded keys of all the renewed nodes are broadcast once only.

We adopt the following notations for the three distributed algorithms. 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 communication group within a rekey interval.

3.1.1. Rebuild Algorithm

The motivation for the Rebuild algorithm is to *minimize* the final tree height so that the rekeying operations for each group member can be reduced. At the beginning of every rekey interval, we reconstruct the whole key tree with all existing members who remain in the group, together with the newly joining members. The resulting tree would be a *complete* tree. The pseudo-code of the Rebuild algorithm to be performed by every member is shown below:

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. rekey the key nodes and broadcast the new blinded keys in T;

Figure 4 illustrates the scenario that members M_2, M_5 and M_7 wish to leave the communication group and a new member M_8 wishes to join the group. The resulting key tree has five members and all the nodes need to be renewed. The sponsors will include all the five members.

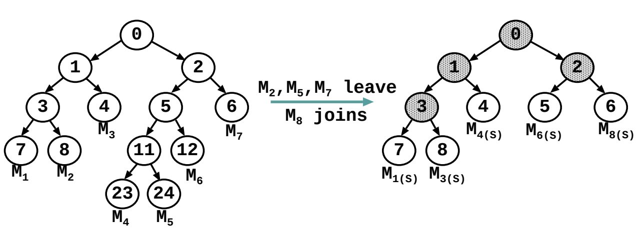


Figure 4. Example of the Rebuild algorithm

3.1.2. Batch Algorithm

The Batch algorithm is based on the centralized approach in [6], except that we are now applying it to a distributed system without a centralized key server and all clients contribute to the composition of the group key. The pseudo code of the Batch algorithm is given as:

Batch (T, $\boldsymbol{M}^j$, J, $\boldsymbol{M}^l$, L)
1. **if** ($L == 0$) { `/* pure join case */`
2. create a new tree T' based on new members in $\boldsymbol{M}^j$;
3. either (a) add T' to the shallowest node of T (which need not be the leaf node) such that the merge would not increase the height of the result tree, or (b) add T' to the root node of T if the merge to any node of T would increase the tree height;
4. } **else** { `/* L > 0 */`
5. sort $\boldsymbol{M}^l$ in an ascending order of the associated node IDs of the members and store the results in $\boldsymbol{M}^{l,s} = \langle M_1^{l,s}, \cdots, M_L^{l,s} \rangle$;
6. **if** ($L \geq J$) {
7. `/* more members want to leave than join */`
8. **if** ($J > 0$)
9. replace the departed nodes of $\langle M_1^{l,s}, \cdots, M_J^{l,s} \rangle$ with J joined nodes;
10. **if** ($L - J > 0$) {
11. remove remaining $L - J$ leaving leaf nodes from the parent node;
12. promote the siblings of the leaving leaf nodes;
13 }
14 } **else** {
15. `/* more newly joining members than leaving members */`
16. 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;
17. create L subtrees $\langle T_1', \cdots, T_L' \rangle$ for the subgroups $\boldsymbol{G}$;
18. 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;
19. }
20. }
21. elect the members to be sponsors if (1) they are new members, or (2) the rightmost members of the subtrees rooted at the siblings of the departed nodes or replaced nodes in T;
22. **if** (sponsor) `/* sponsor's responsibility */`
23. rekey the key nodes and broadcast the new blinded keys;

Notice that the sponsors may have to wait for the blinded keys on another key path in order to proceed upwards to rekey the nodes. Finally, all the members obtain the necessary blinded keys to compute the new group key K_0.

The Batch algorithm is illustrated with two examples. In Figure 5, we illustrate 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 will be 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 selected to be the sponsors. M_1 rekeys secret keys K_1 and K_0 and M_4 rekeys 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 can compute the group key based on the received blinded keys.

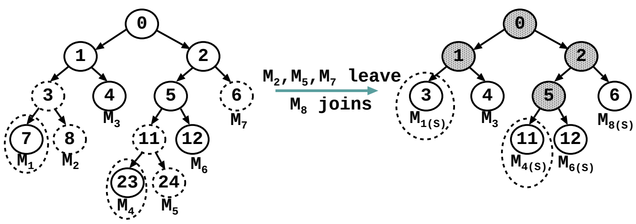


Figure 5. Example 1 of the Batch algorithm where $L > J > 0$

Figure 6 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 key. (ii) M_8 and M_9 form the subtree T'_1 and M_{10} is the only member of the subtree T'_2. The root of T'_1 replaces node 6 and the root of T'_2 replaces node 8. (iii) The sponsors will be 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 can compute and broadcast the new blinded key BK_6. (iv) M_1 (or M_{10}) rekeys K_3 and K_1 and broadcasts BK_3 and BK_1. M_6 rekeys K_2 and broadcasts BK_2. (v) Finally, all the members can compute the group key K_0.

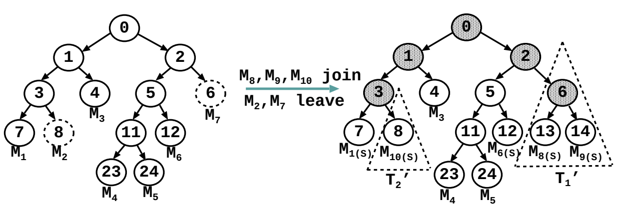


Figure 6. Example 2 of the Batch algorithm where $J > L > 0$

3.1.3. Queue-batch Algorithm

The previous approaches perform rekeying at the beginning of every rekey interval, which can result in a high processing load during the update instance and thereby delay the start of the secure group communication. The processing load includes the computation cost of the exponentiation operations in generating the keys, as well as the communication cost of broadcasting all the blinded keys to all members in the communication group. We propose a more effective algorithm which we call the *Queue-batch* algorithm. The intuition of this algorithm is to reduce the rekeying load by pre-processing the joining members in the virtual queue during the idle rekey interval.

The Queue-batch algorithm is divided into two phases, namely the *Queue-subtree formation* phase and the *Queue-merge* phase. The first phase occurs whenever a new member joins the communication group during the rekey interval. In this case, we append this new member in a temporary key tree T'. The second phase occurs at the beginning of every rekey interval and we merge the temporary tree T' (which contains all newly joining members) to the existing key tree T. Specifically:

Queue-subtree formation (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 the key nodes and broadcast the new blinded keys to the communication group;
    }
}

Queue-merge (T, T', $\boldsymbol{M}^l$, L)

if ($L == 0$) { `/* there are no leave */`
    add T' to either (a) the shallowest node (which need not be the leaf node) of T such that the merge would not increase the resulting tree height, or (b) the root node of T if the merge to any locations would increase the resulting tree height;
} **else** `/* there are leaves */`
    add T' to the highest leave position of the key tree T;
elect members to be sponsors if they are (a) the rightmost member of the subtree rooted at the sibling nodes of the departed leaf nodes in T, or (b) the rightmost member of T';
if (sponsor) `/* sponsor's responsibility */`
    rekey the key nodes and broadcast the new blinded keys to the communication group;

The Queue-batch algorithm is illustrated in Figure 7, 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) At the *Queue-subtree formation* phase, the three new members M_8, M_9, and M_{10} would first form a tree T'. M_{10}, in this case, will be elected as the sponsor. (ii) At the *Queue-merge* phase, the tree T'

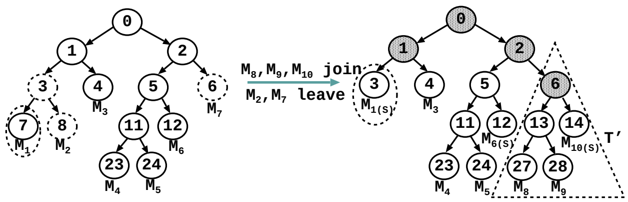


Figure 7. Queue-batch: the Queue-merge phase

will be 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 rekeys the secret key K_1 and broadcasts the blinded key BK_1, M_6 rekeys the secret key K_2 and broadcasts the blinded key BK_2. (iv) Finally, all members can compute the group key.

3.2. Performance Evaluation

In this section, we present the mathematical analysis of the three proposed algorithms. We consider two performance measures, namely:

1. *Number of renewed nodes:* a node is said to be *renewed* if it is a non-leaf node and its associated keys are renewed. This metric provides a measure of the communication cost since new blinded keys of the renewed nodes have to be broadcast to the whole group.

2. *Number of exponentiation operations:* this metric provides a measure of the computation load for all members in the communication group.

For simplicity, we assume the following in the analysis:

- The existing key tree T is a completely balanced tree before the interval-based rekeying event.
- Each member has a homogeneous leave probability.
- The number of blinded key computations simply equals that of renewed nodes, provided that the blinded key of each renewed node is broadcast only once.

For the mathematical analysis, let N be the number of members originally in the system, L (where $0 \leq L \leq N$) be the number of members which wish to leave the system, and $J \geq 0$ be the number of new members which wish to join the communication 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, 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 the number of exponentiations of calculating the blinded keys (which is done by sponsors only). We have

$$\mathcal{E}_{alg} = \mathcal{E}^s_{alg} + \mathcal{E}^b_{alg}. \quad (3)$$

Based on the last assumption, we know the number of blinded key computations is

$$\mathcal{E}^b_{alg} = \mathcal{R}_{alg}. \quad (4)$$

In the following analysis, we only consider the number of secret key computations $\mathcal{E}^s_{alg}$.

3.2.1. 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 $\mathcal{E}_{Rebuild}$, 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 (i.e., the number of non-leaf nodes) is

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

For the performance measure $\mathcal{E}_{Rebuild}(N^*)$, we find that when $N^* \leq 1$, $\mathcal{E}_{Rebuild}(N^*) = 0$. If $N^* \in (2^{h'-1}, 2^{h'}]$ for $h' \geq 1$ where $h' = \lfloor \log_2(N^* - 1) \rfloor + 1$, we have

$$\begin{aligned} \mathcal{E}^s_{Rebuild}(N^*) &= (\text{number of members at level } h') \times h' \\ &\quad + (\text{number of members at level } h'-1) \times (h'-1) \\ &= 2\,(N^* - 2^{\lfloor \log_2(N^*-1) \rfloor})(\lfloor \log_2(N^*-1) \rfloor + 1) \\ &\quad + (N^* - 2(N^* - 2^{\lfloor \log_2(N^*-1) \rfloor}))\lfloor \log_2(N^*-1) \rfloor \\ &= N^* \lfloor \log_2(N^*-1) \rfloor + 2N^* - 2^{(\lfloor \log_2(N^*-1) \rfloor + 1)}. \quad (6) \end{aligned}$$

3.2.2. Analysis of the Batch Algorithm

When $L > 0$, the performance metrics will depend on the membership leave positions and exact metrics cannot be obtained. Therefore, whenever $L > 0$, we derive the *expected* performance measures. Due to limited space, readers can refer to [5] for detailed mathematical derivation and results.

3.2.3. Analysis of the Queue-batch Algorithm

The main idea of the Queue-batch algorithm exploits the idle rekey interval to pre-process certain rekeying operations. When we compare its performance with the Rebuild or Batch algorithms, we only need to consider the rekey operations occurring at the beginning of each rekey 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

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

Also, the expected number of exponentiations when $J > 0$ for Queue-batch is given by

$$E[\mathcal{E}_{Queue-batch}] = \begin{cases} N + J, & \text{if } J > 0 \text{ and } L = 0 \\ E[\mathcal{E}_{Batch, L>J=0}], & \text{if } J = 0 \text{ and } L > 0 \\ E[\mathcal{E}_{Batch, J=1 \text{ and } L>0}] - d + dJ, & \\ & \text{if } J > 0 \text{ and } L > 0. \end{cases} \tag{8}$$

For $J > 0$ and $L > 0$, assume the new subtree is attached to a node at some level d. We first decrement d from $E[\mathcal{E}_{Batch,J=1 \text{ and } L>0}]$ to exclude the secret key computations of the leaf node which is now replaced by the root node of the new subtree. We then add dJ to account for the secret key computations done by these new J members.

The value d is the level of the highest node that has all its descendants departed. Instead of computing the expected value of d, we can find the upper bound value of d, which occurs when the leaving leaf nodes are evenly distributed in the key tree. Thus, d is given by

$$d = \begin{cases} \lfloor \log_2(N-L) \rfloor + 1 & \text{if } N > L \\ 0 & \text{if } N = L. \end{cases} \tag{9}$$

4. Experiments

In the previous section, we quantify the performance measures by assuming that the existing tree is completely balanced. In this section, we perform a more elaborate performance study by investigating the costs of exponentiations and renewed nodes of the three proposed algorithms under different experimental settings. The experiments assume a finite population of 1024 users. Out of these 1024 users, there are 512 members originally in the communication group at the beginning of each experiment. We assume that potential members outside the group have a tendency to join the group with the same join 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: (Evaluation based on mathematical models). This experiment evaluates the metrics of the three interval-based algorithms based on the mathematical models presented in Section 3.2. We start with a well-balanced key tree involving 512 members and then obtain the metrics under different values of joins and leaves (i.e. J and L).

Figures 8 and 9 illustrate the average number of exponentiations and average number of renewed nodes under different numbers of joining and leaving members. From these figures, we observe that the Queue-batch algorithm outperforms the other two interval-based algorithm in all cases and there is a significant computation/communication reduction when the peer group is very dynamic (i.e., high number of members who wish to join or leave the group).

Experiment 2: (Average analysis of a finite population with varying join probabilities). The previous experiment studies the case where the original tree is a balanced key tree. In this experiment, we further examine the case when 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 and evaluate the average performance measures of the three algorithms under various leave probabilities. The results are illustrated in Figures 10 and 11. From these figures, we observe that Queue-batch outperforms the other two algorithms in terms of the costs of exponentiations 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 rekey 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., they have similar average numbers of exponentiations and average numbers of renewed nodes when p_L is higher than 0.6 and 0.8, respectively). Nevertheless, Queue-batch outperforms the other two algorithms at different join/leave probabilities. This shows that the pre-processing of the join requests in Queue-batch can significantly reduce the computation and communication loads at the rekey intervals.

Experiment 3: (Instantaneous performance measures of a finite population). This experiment compares the instantaneous performance measures of the Batch and Queue-batch algorithms over 300 rekey intervals (we ignore the rebuild algorithm because it performs the worst among the three algorithms). We consider the cases with different values of p_J and p_L to represent different mobility characteristics of the peer group.

Figure 12 illustrates the instantaneous number of exponentiations at different values of p_J and p_L. We note that when the communication group has a high leave probability, Queue-batch significantly outperforms Batch. Figure 13 illustrates the instantaneous number of renewed nodes. It shows that Queue-batch has a much lower cost in renewing nodes, as compared to Batch. This implies that Queue-batch can reduce the communication cost significantly.

Summary of experimental results. The above experiments show that Queue-batch offers the best performance in terms of computation and communication costs among the three interval-based algorithms. The superior performance of Queue-batch is more obvious when the occurences of join and leave events are highly frequent.

5. Related Work

Wong *et al.* [11] and Wallner *et al.* [10] independently proposed the key tree approach to secure group communications. They suggested to associate keys in a hierarchical tree and rekey at every join or leave event. Later, the authors in [6, 7, 12] introduced the concept of batch rekeying to enhance system efficiency since the rekeying workload is independent of membership dynamics. All the above approaches rely on a centralized key server, which is responsible for generating and distributing new keys.

The authors in [1, 9, 3, 4] extended the Diffie-Hellman protocol [2] to group key agreement schemes for secure communications in a peer-to-peer network. Burmester *et al.* [1] proposed a computation-efficient protocol at the expense of high communication overhead. Steiner *et al.* [9] developed *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.* [3] proposed the Tree-Based Group Diffie-Hellman (TGDH) to arrange keys in a tree structure. Every member only needs to hold the keys along its key path, implying that the rekeying workload is distributed to all members. The authors also suggested a variant of TGDH called STR which minimizes the communication overhead by trading off the computational complexity [4]. All the above schemes are *contributory*, meaning that key generation is performed by all members and hence avoids the single-point-of-failure problem in the centralized approach. While the scheme in [1] is independent of membership change, the rest of the schemes [9, 3, 4] suggest to perform rekeying at single join, leave, merge or partition events. Our paper enhances the scheme in [3] to support rekeying involving a batch of join and leave events.

6. Conclusion

We have considered several distributed collaborative key agreement protocols for dynamic peer groups. The key agreement setting is performed wherein there is *no centralized key server* to maintain or distribute the group key. We show that one can use the Tree-Based Group Diffie-Hellman protocol to achieve such distributive and collaborative key agreement. To reduce the rekey complexity, we propose to use an interval-based rekey approach so that we can group multiple join/leave requests and process them at the same time. In particular, we show that the Queue-batch algorithm can significantly reduce both computational and communication costs. This reduction enables a more efficient way to manage secure group communication.

References

[1] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology – EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 275–286. Springer-Verlag, 1995.

[2] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[3] Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. *Proc. of 7th ACM Conference on Computer and Communications Security*, pages 235–244, November 2000.

[4] Y. Kim, A. Perrig, and G. Tsudik. Communication-efficient group key agreement. *Information Systems Security, Proceedings of the 17th International Information Security Conference IFIP SEC'01*, November 2001.

[5] P. P. C. Lee, J. C. S. Lui, and D. K. Y. Yau. Distributed collaborative key agreement protocols for dynamic peer groups. Technical report cs-tr-2002-04, Dept of Computer Science and Engineering, Chinese University of Hong Kong, May 2002. Also as CS TR-02-013, Purdue University, West Lafayette, IN.

[6] X. S. Li, Y. R. Yang, M. G. Gouda, and S. S. Lam. Batch rekeying for secure group communications. *Proceedings of Tenth International World Wide Web Conference (WWW10)*, May 2001.

[7] S. Setia, S. Koussih, and S. Jajodia. Kronos: A scalable group re-keying approach for secure multicast. *Proc. of IEEE Symposium on Security and Privacy 2000*, May 2000.

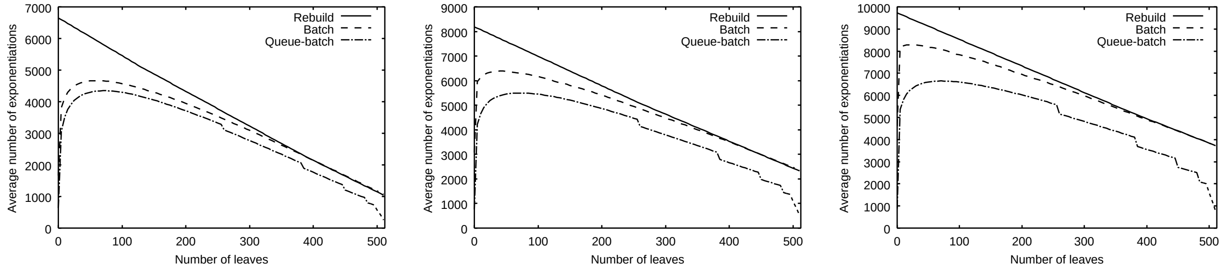
[8] W. Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall, 2nd edition, 1999.

[9] M. Steiner, G. Tsudik, and M. Waidner. CLIQUES: A new approach to group key agreement. *IEEE International Conference on Distributed Computing Systems*, pages 380–387, May 1998.

[10] 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, July 1999. Expires in six months.

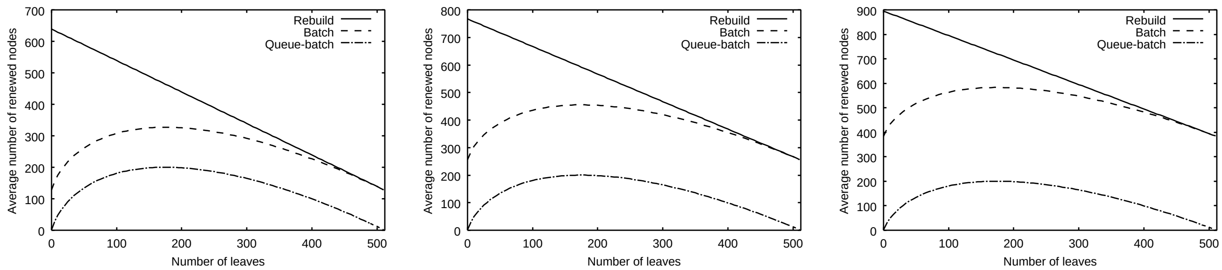
[11] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. *Proc. of ACM SIGCOMM'98*, September 1998.

[12] Y. R. Yang, X. S. Li, X. B. Zhang, and S. S. Lam. Reliable group rekeying: A performance analysis. *Proc. of ACM SIGCOMM'01*, August 2001.



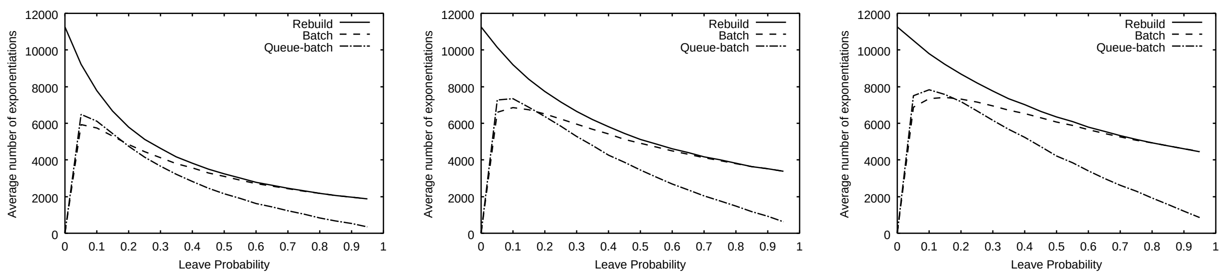
(a) $J = 128$ (b) $J = 256$ (c) $J = 384$

Figure 8. Average number of exponentiations at different numbers of joins when the original tree is completely balanced



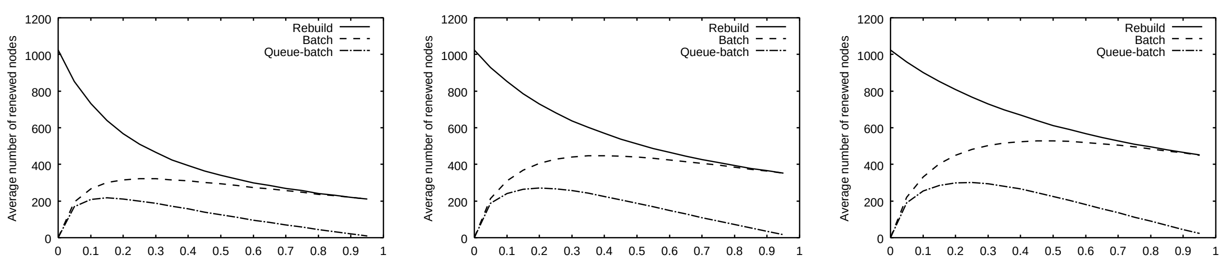
(a) $J = 128$ (b) $J = 256$ (c) $J = 384$

Figure 9. Average number of renewed nodes at different numbers of joins when the original tree is completely balanced



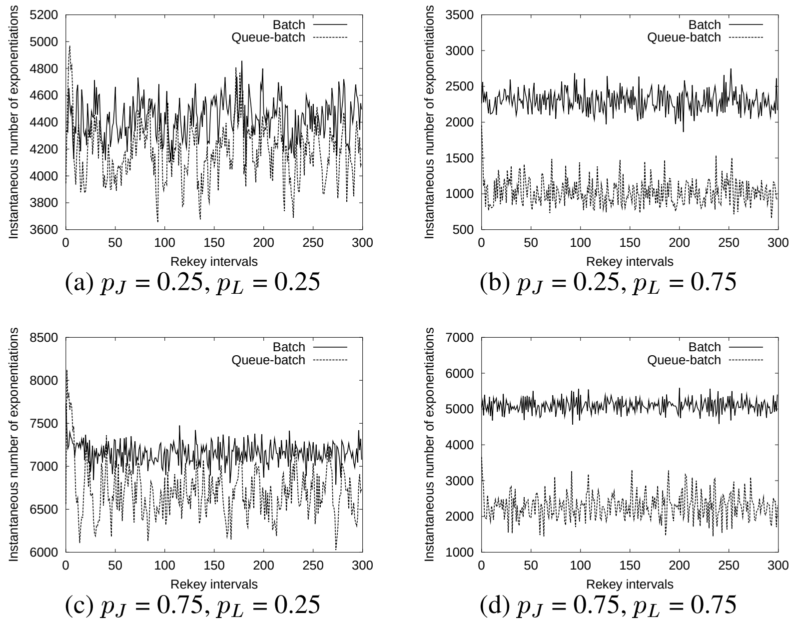
(a) $p_J = 0.25$ (b) $p_J = 0.5$ (c) $p_J = 0.75$

Figure 10. Average number of exponentiations at different join probabilities



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

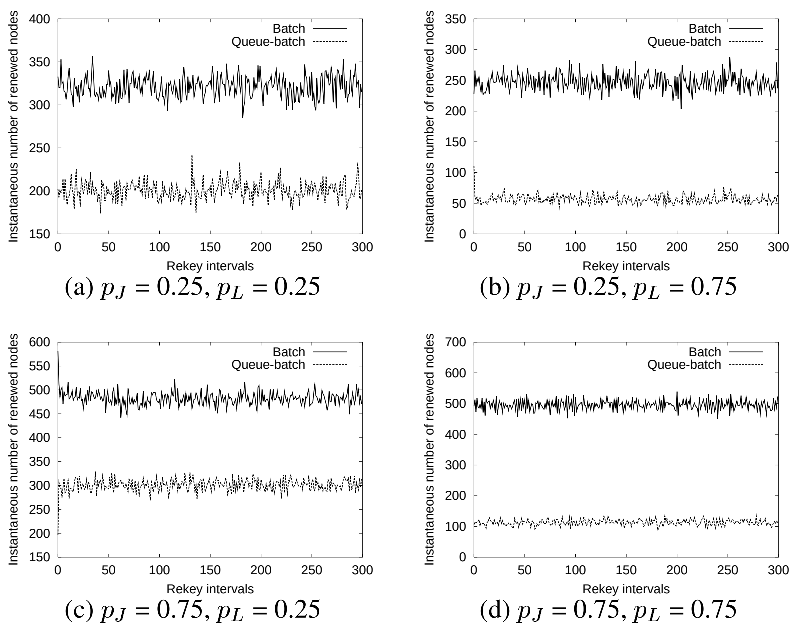
Figure 11. Average number of renewed nodes at different join probabilities



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

(c) $p_J = 0.75, p_L = 0.25$ (d) $p_J = 0.75, p_L = 0.75$

Figure 12. Instantaneous number of exponentiations at different join and leave probabilities for Batch and Queue-batch



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

(c) $p_J = 0.75, p_L = 0.25$ (d) $p_J = 0.75, p_L = 0.75$

Figure 13. Instantaneous number of renewed nodes at different join and leave probabilities for Batch and Queue-batch