International Journal of Innovative Computing, Information and Control
Volume **8**, Number **8**, August **2012**

ICIC International ©2012 ISSN 1349-4198
pp. 5523–5542

RESIDENCY-BASED DISTRIBUTED COLLABORATIVE KEY AGREEMENT FOR DYNAMIC PEER GROUPS

Chii-Jyh Guo and Yuh-Ming Huang

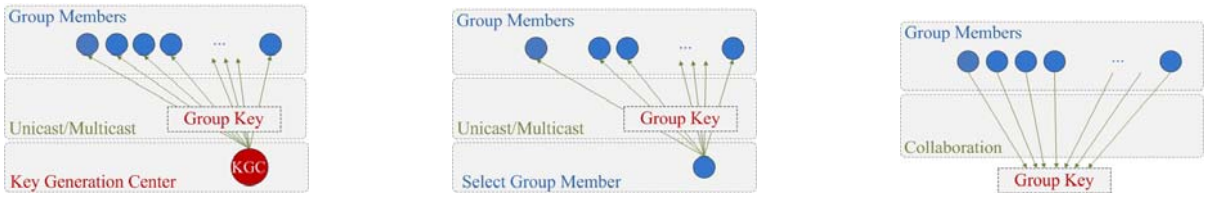
Department of Computer Science and Information Engineering
National Chi-Nan University
No. 1, University Road, Puli, Nantou 54561, Taiwan
huby@mail2000.com.tw; ymhuang@csie.ncnu.edu.tw

Received May 2011; revised September 2011

Abstract. *The contributory group key agreement is suitable for real group-oriented network environments because it avoids the drawbacks of the centralized key generation center and the problem with the single point of failure in three different types of group-oriented communication schemes. Previous studies have pointed out that the group member topology and the group key re-establishment strategy are two important aspects of scheme performance, especially with frequent member changes and large group sizes. Hence, based on the binary tree topology, we proposed the Residency-based Batch Rekeying Algorithm (RBR) to deal with membership dynamic events and group key re-establishment to maintain the forward/backward secrecy. The Residency-based Batch Rekeying Algorithm is divided into four phases, namely the Queue-Tree phase, the QT-Merge phase, the ST-Merge phase, and the Relocation phase. Using the concepts of subtree division and residency time classification, we design a new key tree topology to ensure that the join/leave member's location is as near as possible to the key tree root node to reduce the rekeying cost. Finally, we show that the rekeying cost of RBR is lower than that of any other proposed scheme via both mathematical analysis and simulation experiment.*
Keywords: Residency-based rekeying, Interval-based rekeying, Contributory group key agreement, Dynamic peer groups

1. **Introduction.** With the development of networking technologies, many group-oriented applications, such as bank transactions, teleconferencing, and multi-player games, conveniently assist humans with information dissemination. Unequivocally, information security is an important issue in communication privacy and data integrity. For example, in the ad hoc network, in order to prevent unauthorized access, encryption algorithms are commonly used for secure data delivery. The encryption keys must be negotiated by all group members so that they can receive and decrypt the information [1]. Instead of repeatedly processing a pair-wise Diffie-Hellman group key agreement [2] between each pair of group members, group-oriented communication uses a group key [3-5] held by all group members to encrypt the communication content [6,7] and simultaneously achieve both confidentiality and efficiency.

The group key management approaches of past studies can be classified into three different types. The approach shown in Figure 1(a), called *Centralized Group Key Distribution*, relies on a single *Key Generation Center* (*KGC*) to generate and distribute all of the keys to group members. Despite the advantage of a lower computation cost for each member, this approach has two problems [8]: 1) KGC must be constantly available and 2) KGC must exist in every possible subset of a group in order to support continued operation in the event of network partitions. Another approach, called *Decentralized Group Key Distribution* and shown in Figure 1(b), involves dynamically selecting a group member to



(a) (b) (c)

FIGURE 1. Three types of the group key management approaches. (a) The centralized group key distribution. (b) The decentralized group key distribution. (c) The contributory group key agreement.

generate and distribute all of the keys to the other group members. Because any partition can elect a member to be the key server, this approach is more robust and applicable to group key management. However, the key server must consume significant costs to establish long-term pairwise secure channels with all current members to distribute the group key, and some problems still arise, as in the KGC case. In contrast to the above approaches, the *Contributory Group Key Agreement* (or *Group Key Agreement*), requires all group members to contribute their shares and compute the group key collaboratively [8-12], as shown in Figure 1(c). This avoids the problems with the centralized trust and the single point of failure. Moreover, some contributory methods do not require the establishment of pair-wise secret channels among group members. Because the contributory group key agreement does not need to depend on the KGC, this type of group key management becomes necessary in the situation [10]: 1) a central key server KGC cannot be established; 2) the group members are equal and do not trust another entity to manage their private key; or 3) the members and server do not share any common knowledge about each other's secret keys beforehand. Over the past decades, several contributory group key agreement schemes have been proposed [8-16], and all of these schemes have pointed out that there are two important aspects of scheme performance that should be noticed. 1) The complexity for generating the group key will closely depend on the group member (key tree) topology. For example, using the ring-based member topology, Burmester and Desmedt (BD) [12] proposed an efficient protocol that takes only two rounds per member to generate a group key. However, it requires $2n$ (n is the group size) broadcast messages, which are expensive on a wide area network [8]. Under the same conditions, using the balanced binary tree-based topology, Kim et al.'s Tree-based Group Diffie-Hellman protocol (TGDH) [8] requires $\log n$ rounds and broadcasts $2 \log n$ messages per member to generate a group key. Moreover, when group member dynamic events occur, the group key needs to be reconstructed to maintain the backward and forward secrecy. Therefore, 2) a group key re-establishment strategy (called the *Rekeying* algorithm) is another aspect that should be considered. However, this operation often consumes a large amount of communication and computation overhead, especially for a large dynamic group, which is an important issue for some lower-capability devices such as mobile ad hoc or sensor networks. In 1994, Burmester and Desmedt (BD) [12] proposed an efficient protocol in the initialization phase of group key establishment. However, the rekeying cost upon group membership change events is still high. To improve that, Steiner et al. extended the two-party Diffie-Hellman protocol [2] to the group Diffie-Hellman protocol (GDH) [17,18]. While it is more efficient, it still requires a high cost when members leave. Recently, in order to achieve scalability for contributory group key agreements, based on the use of the binary balanced tree topology to maintain the key material, Kim et al. proposed TGDH [8], where the average rekeying rounds when a single member joins/leaves is $\log n$. In

TGDG, the rekeying cost for the remaining members is in direct proportion to the depth of the join/leave member's location (node depth). In order to reduce the join/leave node depth, Mao et al. proposed the Dynamic SubTree (DST) [10] group key agreement using a special join-exit-tree topology and exploiting cost amortization in 2006. It reduces the average rekeying rounds when a single member joins/leaves to $\Theta(\log \log n)$. However, Yu et al. [11] indicated that DST has an unrealistic requirement that the member's leave time is known by others in advance. If this condition does not hold, the time cost is higher than that in TGDH. In addition, Yu et al. used a novel logical key tree structure (called PFMH) to divide the group into the join tree and main tree. In addition, they used the concept of phantom member position to design the PFMH tree-based contributory group key agreement (PACK) to achieve the performance lower bound proposed by Snoeyink et al. [19]. Unfortunately, the height of the main tree (leave node position) is nearly $\log n$, and simulation results also showed that the computation and communication costs in PACK are slightly higher than those in TGDH upon a single member leave event.

According to the rekeying timing, the rekeying strategies can be classified into two categories. 1) The *individual-based distributed rekeying algorithms*, including [2,8,10,11,17,18]. A rekeying is performed immediately upon the occurrence of a member join/leave event, even if the member only stayed for a very short time. Intuitively, the computation/communication cost can be reduced by 2) the *interval-based distributed rekeying algorithm*. Each rekeying and its adjacent rekeying are separated by a fixed time interval. After the previous rekeying, the interval-based distributed rekeying algorithm will temporarily disregard all of the incoming membership join/leave events for a rest period of one time interval. In 2006, Lee et al. [9] first proposed the interval-based distributed rekeying algorithm, called the Queue-Batch algorithm (QB), and showed that the rekeying cost is lower than that of an individual-based distributed rekeying algorithm. Recently, several interval-based distributed rekeying algorithms [20-25] have been proposed, but these algorithms are still similar to QB.

In order to evaluate the performance of the rekeying process, both a mathematical analysis and simulation experiment should be used. Conventionally, in previous studies [8-11], the performance evaluation in a mathematical analysis was made under the assumption that the key tree is completely balanced. However, this assumption may be inconsistent with reality. Through the queuing theory [26], we used a simulation experiment that provides all user dynamic events to random demands to evaluate the overhead of all of the proposed algorithms and found that the membership dynamic events in the simulation experiment could lead to an unbalanced key tree and high rekeying cost. For example, QB can be regarded as a special case of our proposed scheme where the join tree capacity is unlimited. In fact, in the simulation, the rekeying costs of QB were higher than those in the mathematical analysis. Therefore, to meet reality, we gave more attention to the simulation results.

In this paper, based on the tree-based group key agreement and interval-based distributed rekeying algorithm, we propose the Residency-based Batch Rekeying (RBR) algorithm, and the contributions are as follows. 1) Using a lower capacity join tree and the concept of QB, we propose using *Queue-Tree* and *QT-Merge* phases to deal with the member join events, which allows the computation and communication costs of member join events to be reduced and more efficient than those of PACK. 2) Dividing the main tree into several subtrees and disposing those subtrees' root nodes depth are ascending from the root node. Based on the members' residence times, we propose using *ST-Merge* and *Relocation* phases to reset the location of each subtree so that the longest-staying members (having higher leaving probability) are closest to the root node. We not only solve the unrealistic requirement of DST that the member's leave time is known by others

in advance, but achieve higher rekeying performance than PACK during the member leave events. 3) We evaluate the performance of our RBR algorithm using both a mathematical analysis and simulation experiment.

The rest of this paper is organized as follows. Section 2 briefly introduces the tree-based group Diffie-Hellman protocol. Section 3 presents the proposed RBR algorithm with four phases. In Section 4, we evaluate the overhead of our proposed scheme using a mathematical analysis and make a comparison with other schemes. In Section 5, the simulation results are presented and discussed. In Section 6, we briefly describe the security analysis, and finally the conclusions are presented in Section 7.

2. Background of Tree-Based Group Diffie-Hellman Protocol (TGDH).

Over the past few decades, tree-based contributory group key agreements have been the most promising because of their scalability. In this section, we briefly introduce the tree-based group Diffie-Hellman (TGDH) [8] protocol for reconstructing the group key for security requirements, such as group key security, forward secrecy, backward secrecy, and key independence in user dynamic events. In TGDH, the key set that was maintained by each user is arranged in a hierarchical binary tree, as illustrated in Figure 2. To facilitate the reading, we assign a node ID, v, to every tree node. Each node v holds a secret (or private) key, K_v, and computes a blinked (or public) key, $BK_v = g^{K_v} \bmod p$. All of the arithmetic operations are based on the Diffie-Hellman protocol and performed in a cyclic group of prime order p with generator g. The i-th group member is denoted by M_i, and arranged on the leaf nodes in a key tree. Each M_i selects an individual secret key through a secure pseudo random number generator and publishes its corresponding blinked key to the sibling node. Moreover, all members hold all secret keys from their associated leaf node up to the root node, called the *key-path*, and the secret key of the root node held by each member simultaneously is the group key. Moreover, the *co-path* denotes the set of the sibling nodes of each node in the key-path. Based on the Diffie-Hellman protocol [2], each non-leaf v consists of two child nodes (i.e., $2v + 1$ and $2v + 2$); the secret key of non-leaf node v can be generated by one child node's secret key and another child node's blinked key. The secret key of non-leaf node v is as follows:

$$K_v = (BK_{2v+1})^{K_{2v+2}} \bmod p = (BK_{2v+2})^{K_{2v+1}} \bmod p = g^{K_{2v+1} \cdot K_{2v+2}} \bmod p \tag{1}$$

For example, in Figure 2, member M_2 first generates a random number to be an individual secret key, K_2. After receiving all of the blinked keys along the co-path (BK_7, BK_4, BK_2), M_2 generates the secret key (K_3, K_1, K_0) using the above two-party Diffie-Hellman agreement, and K_0 is the group key. Upon occurrence of a member join/leave event, the group key needs to be rebuilt (rekeying process) to maintain the backward secrecy (joined members cannot access previous information) and forward secrecy (former members cannot access future information). Therefore, a special member called the *sponsor* is elected

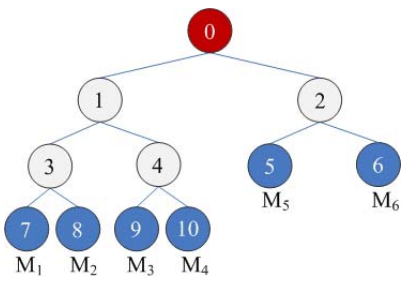


FIGURE 2. Key tree in the tree-based group Diffie-Hellman (TGDH) protocol

to deal with the key updating (reselect his/her secret key and communicate with the join member) and adjust the node position in a member leave case.

3. **Residency-Based Batch Rekeying Algorithm.** In this section, we first present our logical key tree topology and the associated algorithms in Sections 3.1 and 3.2, respectively. Furthermore, we describe each phase in our RBR algorithm in Section 3.3.

3.1. **Key tree topology.** As shown in Figure 3, our proposed logical key tree consists of three parts: 1) the *queue tree* of dynamic size, which is used to deals with the member join/leave events during one idle rekeying time interval, 2) the *join tree* with a lower capacity, which is primarily used to deal with the member join events, and 3) the *main tree*, which is organized into several subtrees using the concept of STR [14] (short for Skinny tree) to primarily deal with the member leave events. Let N be the average group size, and the height of the key tree is at least $\log N$ (tree is balanced). We divide the key tree (including the joint tree and the main tree) into $g = \lceil \log N \rceil$ subtrees, denoted by ST_i, where $1 \leq i \leq g$. During the period of every rekeying time interval, all join members first pre-build a temporary key tree, called the queue tree, and the queue tree is merged into the join tree at the beginning of every rekeying time interval. Independently maintaining a join tree with a lower height makes the queue tree be inserted into a shallower location, which can reduce the rekeying cost for member join events. Later, the join tree will be relocated into be the sibling tree of the deepest and leftmost subtree in the current main tree if the number of members in the join tree reaches the maximum capacity. Besides, before performing the relocation process, the two shallowest subtrees in the main tree (ST_2, ST_3) will be merged to keep the number of subtrees less than g. In this way, the members in ST_2 (the shallowest subtree in the main tree) stay much longer, so those members have relatively higher leaving probabilities. For the same reason, the rekeying cost for member leave events can be reduced.

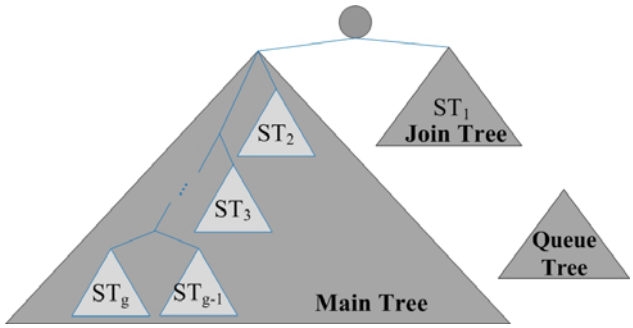


FIGURE 3. Topology for the proposed RBR algorithm

3.2. **Algorithm.** As shown in Figure 4, RBR consists of four phases: the *Queue-Tree* phase, *Queue-Tree-Merge* (*QT-Merge*) phase, *SubTree-Merge* (*ST-Merge*) phase, and *Relocation* phase. Referring to [9], within the batch rekeying time interval, all join members are added into a temporary queue tree in the *Queue-Tree* phase. At the beginning of every batch rekeying time interval, the *QT-Merge* phase first completely immigrates the queue tree into the shallowest leave node location in the join tree and then deals with the member leave events (promotes the sibling nodes and elects the sponsors). After the *QT-Merge* phase, the *Relocation* phase is activated to move the whole join tree into the main tree as a sibling of the deepest and leftmost subtree if the number of members in the join tree is greater than its maximum capacity. Notice that, once the total number of

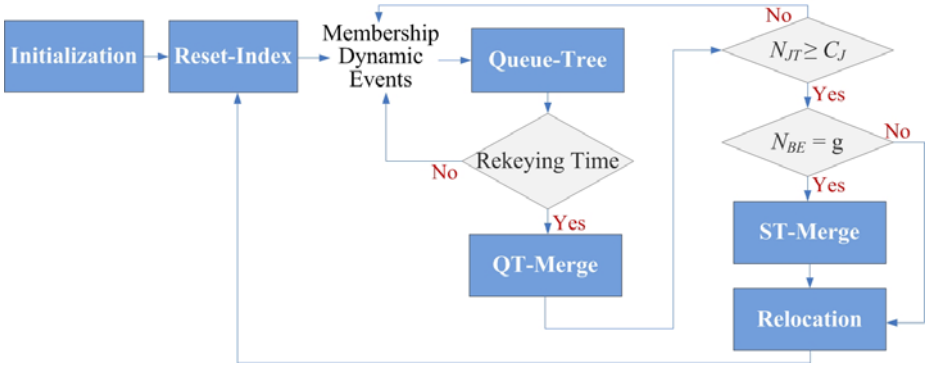


FIGURE 4. Structure of the proposed RBR algorithm

subtrees is greater than g, the *ST-Merge* phase will be activated before activating the *Relocation* phase to keep the number of subtrees equal to g. The interval-based distributed rekeying algorithm is more efficient in terms of less communication and computations costs because it performs the rekeying just in the fixed time slots. After completing one operation of batch rekeying, the interval-based rekeying algorithm temporarily disregards the following membership dynamic events until the beginning of the next rekeying time interval. At this time, another batch rekeying is started. However, the length of the rekeying time interval is a tradeoff with the forward/backward confidentiality. The batch rekeying algorithms are developed based on the following assumptions [9].

1. All members are trusted in the key establishment process.
2. The group communication satisfies view synchrony [8,27] that defines reliable and ordered message delivery under the same membership view. It means that when a member broadcast a message under the membership view, the message is delivered to the same set of members viewed by the sender.
3. Rekeying operations of all members are synchronized to be carried out at the beginning of every rekeying time interval.
4. When a new member sends a join request, it also includes its individual blinded key.
5. All members in the group know the existing key tree structure and all the blinded keys within the tree.
6. To retain the forward/backward confidentiality, sponsors are needed to broadcast the blinded key along its key-path. In the batch rekeying algorithm, more than one sponsor may be elected, and 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.

3.3. **Pseudo-code of each phase.** In this section, some notations are presented as follows. Let N be the average group member size. N_{QT}, N_{JT} and N_{MT} respectively denote the number of members in the queue tree, the join tree, and the main tree. ST_j denote the j^{th} subtree. Note that ST_1 is the join tree, and the number of members in the i^{th} group is denoted by N_{ST_i}. N_{BE} denotes the number of backbone's external node. The join tree capacity is denoted by C_J, and the number of members leaving from the join tree and main tree are denoted by L_{JT} and L_{MT}, respectively.

3.3.1. *Initialization phase and index-reset phase.* The *Initialization* phase is used to set the genesis of group. The system first builds the virtual backbone using the MH key tree topology [11], where the number of external nodes in the backbone is equal to $g = \lceil \log N \rceil$. In the following operation, each external node of the virtual backbone is used to hang a

subtree, i.e., it is the root node of the subtree. To facilitate the reading, the backbones internal/external node indices are denoted by I_i / E_j, where $1 \le i \le g-1$ and $1 \le j \le g$. The Pseudo-code of the *Index-Reset* phase is shown in Figure 5.

Index-Reset

1. Set the backbone internal nodes indexed by I_i from top to bottom, where $1 \le i \le g-1$ and the root node indexed by I_1.
2. Set the backbone external nodes indexed by E_j from top to bottom, where $1 \le j \le g$.

FIGURE 5. Pseudo-code of the index-reset phase

Queue-Tree [9]

```
1.  if (a new member joins {
2.     if (Queue Tree is empty) /* no new member in Queue Tree */
3.        Create a new Queue Tree with the only one member;
4.     else { /* Some new members in Queue Tree */
5.        Find the insertion node; Add the new member to Queue Tree;
6.        Elect the rightmost under the subtree rooted at the sibling of the join node to be
          the sponsor;
7.        if (sponsor)    /* sponsor's responsibility */
8.           Rekey renewed nodes and broadcast new blinded keys;}
9.  }
```

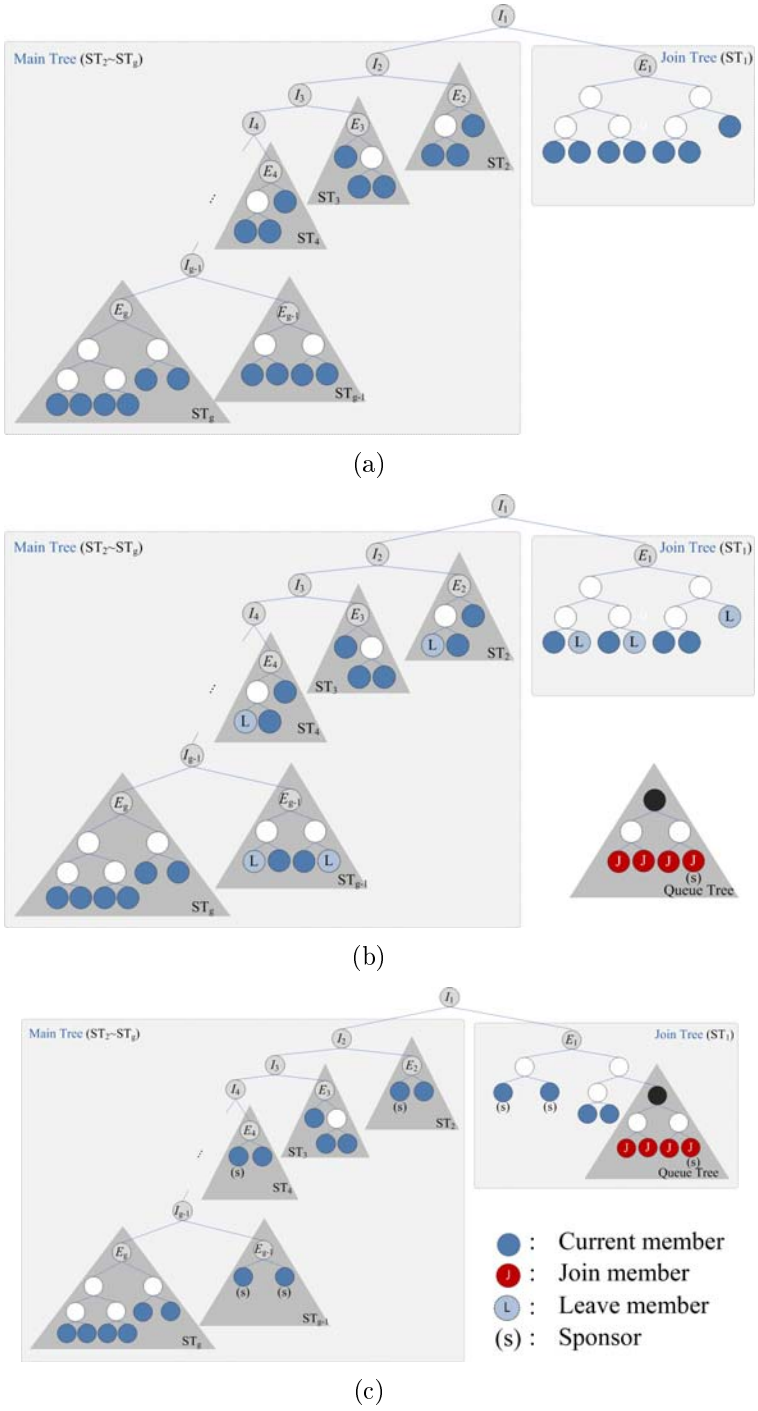
FIGURE 6. Pseudo-code of the queue-tree phase

QT-Merge (*JoinTree*, *Queue-tree*, L_{Main}, L_{Join})

```
1.  if (L_MT == 0 ) { do nothing; } /* no leave member in the Main Tree */
2.  if (L_JT == 0 ) { /* no leave member in the Join Tree */
3.     Add the Queue Tree to either the shallowest node (which need not to be the leaf
       node) of the Join Tree such that the merge will not increase the resulting tree height,
       or the root node of the Join Tree if the merge to any locations will increase the
       resulting tree height;
4.     if ( N_JT ≤ C_J ){ /* Join Tree is not full */
5.     Elect the member who is the rightmost leaf node of the subtree rooted at the sibling
       node of the Queue Tree root in the Join Tree;}
6.     else{/* Join Tree is full */
7.        Relocation;}
8.     }
9.  else{ /* there are leaves */
10.    Add the Queue Tree to the highest leave position of the key tree Join Tree;
11.    Remove the remaining L_Join-1 and L_Main leaving leaf nodes and promote their
       siblings;
12.    if ( N_JT ≤ C_J ){ /* Join Tree is not full */
13.     Elect one member from the Join Tree and Main Tree, respectively, to be the
        sponsors if they are the rightmost members of the subtree rooted at the sibling
        nodes of the deepest departed leaf nodes in the Join Tree and Main Tree;}
14.    else{ Relocation;}
15.    }
16. if (sponsor)    /* sponsor's responsibility */
17.    Rekey renewed nodes and broadcast new blinded keys;
```

FIGURE 7. Pseudo-code of the QT-merge phase

3.3.2. *Queue-tree phase and QT-merge phase.* In the batch rekeying algorithm, at the beginning of every rekeying time interval, the key tree needs to process both member join and leave events. However, higher computation/communication overheads result from processing two different member dynamic events simultaneously, which may cause an ineffective rekeying operation. Therefore, we first preprocess the *Queue-Tree* phase for join members to pre-build a temporary queue tree during the idle rekeying interval. Next,



(a)

(b)

(c)

FIGURE 8. Example of the queue-tree and QT-merge phases. (a) The key tree structure after batch rekeying. (b) The key tree structure before next batch rekeying. (c) The key tree structure after next batch rekeying.

we merge the temporary queue tree into the join tree using the *QT-Merge* phase in the beginning of the every rekeying interval. The pseudo-codes of the *Queue-Tree* phase and *QT-Merge* phase are illustrated in Figure 6 and Figure 7, respectively. For example, the key tree structure after a particular batch rekeying is shown in Figure 8(a). During the rekeying interval before the next batch rekeying, there are some new members that want to join and some current members that want to leave. As shown in Figure 8(b), the join members first process the *Queue-Tree* phase to form a queue tree and elect the rightmost member under the subtree rooted at the sibling of the join node to be the sponsor. The leave nodes in the whole key tree are first removed, and the subtrees rooted at the leave nodes' siblings are not promoted immediately until the beginning of the next rekeying time interval. As shown in Figure 8(c), the *QT-Merge* phase merges the temporary queue tree into the shallowest leave node location in the join tree and elects the sponsors for other leave nodes.

3.3.3. *ST-merge phase and relocation phase.* Past studies have shown that the cost for remaining members to process the batch rekeying is in direct proportion to the depth of the join or leave node level. Hence, based on the average number of group members, we divide the whole key tree into some subtrees to reduce the height of the join tree. As our proposed key tree topology, ST_1 is the join tree, and the shallower queue tree insert location will cut down the rekeying cost. For the same reason, members in ST_2 (the shallowest subtree in the main tree) are the longest-staying and have the highest relative leaving probability. In order to maintain the topology of the key tree, the *Relocation* and *ST-Merge* phases will be operated if $N_{JT} \leq C_J$. The join tree will be relocated into the sibling with the deepest subtree in the main tree if the number of join tree members has reached the maximum capacity. To keep the number of subtrees less than g, the *ST-Merge* phase first determines the lowest subtree rooted at E_2 (root node of ST_2), and then merges ST_3 to be a sibling with this subtree. Figure 9 and Figure 10 illustrate the pseudo-codes of the *Relocation* and *ST-Merge* phases, respectively.

Relocation

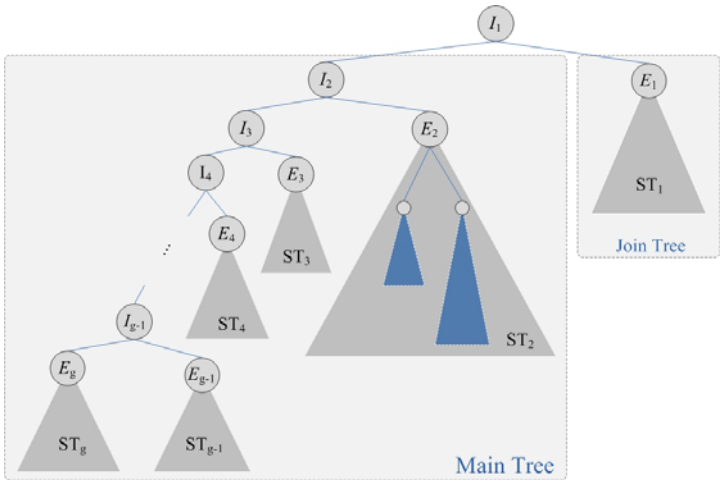
1. /* Relocate Join tree*/
2. Find the deepest internal node in virtual MH Tree, denoted by $I_{deepest}$;
3. Create $E'_{deepest+1}$;
4. $E'_{deepest+1}.RLink \rightarrow E_{deepest+1}$;
5. $E'_{deepest+1}.LLink \rightarrow E_1$;
6. $I_{deepest}.LLink \rightarrow E'_{deepest+1}$
7. /* Rebuild Join tree root node */
8. Create E'_1
9. $I_1.RLink \rightarrow E'_1$;
10. Index-Reset;
11. Elect one member to be the sponsor if he/she is the rightmost member of the subtree rooted at the sibling nodes of the deepest departed leaf nodes in the Main Tree, or they are the rightmost member rooted at E_g ;
12. **if** (sponsor) /* sponsor's responsibility */
13. rekey renewed nodes and broadcast new blinded keys;

FIGURE 9. Pseudo-code of the queue-tree phase

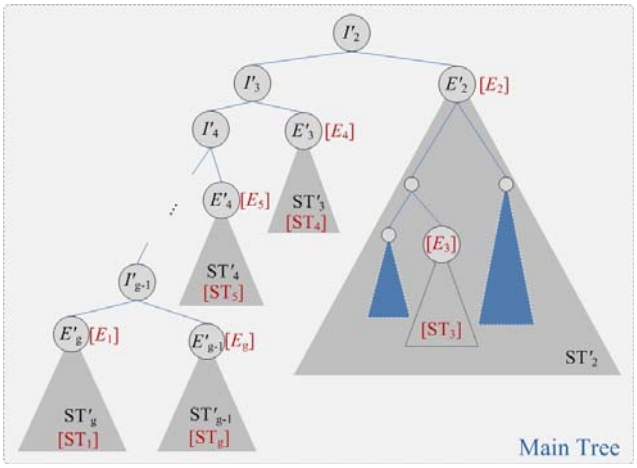
ST-Merge

1. **if** ($H_{E_2\,LeftChild} > H_{E_2\,RightChild}$) {
2. Create $E'_{2\,RightChild}$ and drop I_3;
3. $E'_{2\,RightChild}.RLink \rightarrow E_3$, $E'_{2\,RightChild}.LLink \rightarrow E_{2RightChild}$ and $I_2.LLink \rightarrow I_4$;
4. } **else if** ($H_{E_2\,LeftChild} < H_{E_2\,RightChild}$) {
5. Create $E'_{2\,LeftChild}$ and drop I_3;
6. $E'_{2\,LeftChild}.RLink \rightarrow E_3$, $E'_{2\,LeftChild}.LLink \rightarrow E_{2LeftChild}$ and $I_2.LLink \rightarrow I_4$;
7. } **else{**
8. Create E'_2 and drop I_3;
9. $E'_2.RLink \rightarrow E_3$ and $E'_2.LLink \rightarrow E_2$;
10. $I_2.RLink \rightarrow E'_2$ and $I_2.LLink \rightarrow I_4$;
11. }
12. Index-Reset;

FIGURE 10. Pseudo-code of the QT-merge phase



(a)



(b)

FIGURE 11. Example of the ST-merge and relocation phases. (a) BEFORE the ST-merge and Relocation. (b) AFTER the ST-merge and relocation.

For example, the key tree structure before the *ST-Merge* and *Relocation* is shown in Figure 11(a). Before the ST_1 *Relocation* phase, the two shallowest subtrees in the main tree (ST_2, ST_3) will be merged to keep the number of subtrees less than g. As shown in Figure 11(b), because the height of E_2's left subtree is shorter than E_2's right subtree, in the *ST-Merge* phase, ST_3 is merged to be the sibling of E_2's left subtree. Next, ST_1 is relocated to be the sibling with the deepest subtree, ST_g, in the main tree in the *Relocation* phase. Finally, the *Index-Reset* phase resets the backbone's internal/external nodes index. Note that, to facilitate the reading, $[E_i]$ and $[ST_i]$ denote the external node index and subtree index before the *ST-Merge* and *Relocation* phases, respectively.

4. **Mathematical Analysis.** In this paper, we evaluate the performance of the RBR Algorithm using two methods: mathematical analysis and simulation. Before these evaluations, we generate the membership dynamic events according to the following probabilistic models: the member join events according to a Poisson process with an average arrival rate of λ. The member leave events are defined by the members staying time in the group, which follows an exponential distribution with a mean of α [13,26]. Therefore, the average group member size is $N = \alpha\lambda$. The mathematical analysis considers the complexity of the algorithms under the assumption that the key tree is completely balanced [9]. For ease of evaluation, through the queuing theory [26], we define the group member residual rate $\beta_1 = e^{-\frac{R_I}{\alpha}}$ for the time between R_I, and $\beta_2 = e^{-\frac{kR_I}{\alpha}}$ after the time kR_I, where R_I is the rekeying time interval.

In previous studies, the number of rounds was usually used to evaluate the efficiency of the rekeying process in the Diffie-Hellman contributory group key agreement [9-11,17,18]. However, a difference in the mathematical computations in distinct rounds may cause an inequality comparison [17,18]. To address this problem, we apply the simple round concept [28] in our RBR algorithm. In each round, each member not only can perform at most one two-party Diffie-Hellman operation, but can also send and receive at most one message. According to [10], we define the average join/leave rounds as the number of rounds to perform the rekeying process for a member join/leave event. The average join/leave rounds, denoted by $AverageRounds_{Join/leave}$, is defined as

$$AverageRounds_{Join/Leave} = \frac{TotalRound_{Join/Leave}}{N_{Join/Leave}} \tag{2}$$

where $TotalRound_{Join/Leave}$ is the total number of Diffie-Hellman rounds performed for $N_{Join/Leave}$ join/leave events. In addition, the communication cost of a contributory key agreement refers to the number of messages sent for a rekeying process during a join or leave event. Under the fundamental assumptions that each sending message incurs the same communication cost [10], we evaluate the communication performance using the average number of messages per member join or leave event. For a member join/leave event, because the changed node and its sponsor are needed to update the blind keys along their key-paths, the average join/leave communication cost can intuitively be regarded as twice the average join/leave rounds. Furthermore, we evaluate the computation efficiency by the average number of total exponentiations per member join or leave event.

4.1. **Member join.** Time 0 is set as the time point after the previous *Relocation* phases, and whole join tree members have moved to the main tree (i.e., the join tree is now empty). Consider the circumstances that during k rekeying time intervals, there are a total of x members in the join tree. At the rekeying time point, the queue tree is combined with the join tree. At the beginning of every batch rekeying time interval, the whole queue tree is immigrated into the shallowest leave node location in the join tree directly. Hence, the

join rounds for each batch rekeying are the sum of the backbone's depth and the height of the join tree (i.e., rounds $= 1 + H_{JoinTree}$).

For example, as shown in Table 1, R_I members join the queue tree between adjacent rekeying time intervals. At time 0, it only needs one round to immigrate the queue tree into the empty join tree. At time $2R_I$, the number of join tree residual members is $\lambda R_I \beta_1$. Hence, it needs $1 + \log \lambda R_I \beta_1$ rounds for the queue tree to be immigrated to the join tree.

After kR_I, the total members in the join tree is equal to x, and

$$x = \lambda R_I (1 + \beta_1 + \cdots + \beta_1^{k-2})\beta_1 + \lambda R_I = \lambda R_I \left(\frac{1 - \beta_1^k}{1 - \beta_1} \right) \tag{3}$$

We can evaluate the total join rounds of x join members as follows:

$$\begin{aligned} &\left(\begin{array}{l} 1 + (1 + \log \lambda R_I \beta_1 + \log \lambda R_I (1 + \beta_1)\beta_1) + (1 + \log \lambda R_I (1 + \beta_1 + \beta_1^2)\beta_1) \\ + \cdots + \left(1 + \log \lambda R_I (1 + \beta_1 + \beta_1^2 \cdots \beta_1^{k-2})\beta_1\right) \end{array} \right) \\ &\leq k + (k-1) \log \lambda R_I (1 + \beta_1 + \beta_1^2 \cdots \beta_1^{k-2})\beta_1 \\ &\quad k + (k-1) \log \lambda R_I \left(\tfrac{1 - \beta_1^{k-1}}{1 - \beta_1} - 1 \right) = k + (k-1) \log (x - \lambda R_i) \\ &\leq k + (k-1) \log x \end{aligned} \tag{4}$$

As a result, the average join rounds and average communication costs for one member to join the group are as follows:

$$\begin{aligned} AverageRounds_{Join} &\leq \frac{1}{x} \left(k + (k-1) \log x + h_{main} \right) \\ &= \frac{1}{x} \left(k + (k-1) \log x + g - 2 + \log x \right) = \frac{k}{x} (\log x + 1) + \frac{g-2}{x} \\ &\leq \frac{k}{x} (\log x + 1) + 1 \end{aligned} \tag{5}$$

$$AverageCommunicationCost_{Join} = 2 \left(\frac{k}{x} (\log x + 1) + 1 \right) \tag{6}$$

Denoting N_{OJT} as the old join tree size, the computation cost of each rekeying time interval is equal to $N_{MT} + 2(N_{OJT} + 1)$. The term N_{MT} in the computation cost comes from performing the Diffie-Hellman operation to generate the group by the members in the main tree. In the *QT-Merge* phase, the queue tree merge can be regarded as a one member join. Therefore, according to [11], the term $2(N_{OJT} + 1)$ in the computation cost comes from one member joining the old join tree. Because the average number of group members $N = N_{MT} + N_{OJT} + N_{QT}$, the computation cost of each rekeying time interval is bounded as follows:

$$N_{MT} + 2(N_{OJT} + 1) = N_{MT} + 2N_{OJT} + 2 \leq N + N_{OJT} + 1 \tag{7}$$

Between, the total computation cost in term $2(N_{OJT} + 1)$ is

$$\begin{aligned} &(\lambda R_I \beta_1 + 1) + (\lambda R_I (\beta_1 + \beta_1^2) + 1) + \cdots + \left(\lambda R_I (\beta_1 + \beta_1^2 \cdots + \beta_1^{k-1}) + 1 \right) \\ &\leq (k-1) \left(\lambda R_I (\beta_1 + \beta_1^2 \cdots + \beta_1^{k-1}) + 1 \right) = (k-1)(x - \lambda R_I + 1) \leq k(x+1) \end{aligned} \tag{8}$$

TABLE 1. Performance analyses for member join events

$Time$	0	R_I	$2R_I$	...	kR_I
N_{QT}	0	λR_I	λR_I	...	λR_I
N_{JT}	0	0	$\lambda R_I \beta_1$	...	$\lambda R_I \left(1 + \beta_1 + \beta_1^2 + \cdots + \beta_1^{k-2}\right) \beta_1$
Rounds	0	1	$1 + \log \lambda R_I \beta_1$	...	$1 + \log \lambda R_I \left(1 + \beta_1 + \beta_1^2 + \cdots + \beta_1^{k-2}\right) \beta_1$

TABLE 2. Performance analysis for member leave events in average case

ST_i	N_{ST_i}	$BackboneLevel$	$Rounds(Backbone)$	$Round(\mathrm{ST_i})$
ST_g	$x\beta_2^2$	$g-1$	$x\beta_2^2\left(\frac{g}{2}\right)$	0
ST_{g-1}	$x\beta_2^2$	$g-1$	$x\beta_2^2\left(\frac{g}{2}\right)$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathrm{ST}_{\frac{g}{2}}$	$x\beta_2^{\frac{g}{2}}$	$\frac{g}{2}$	$x\beta_2^{\frac{g}{2}}\left(\frac{g}{2}\right)$	$x\beta_2^{\frac{g}{2}}\left(\log x\beta_2\right)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
ST_3	$x\beta_2^{g-2}$	3	$x\beta_2^{g-2}\cdot 3$	$x\beta_2^{g-2}\left(\log x\beta_2^{g-2}\right)$
ST_2	N_{ST_2}	2	$N_{\mathrm{ST}_2}\cdot 2$	$N_{\mathrm{ST}_2}\left(\log N_{\mathrm{ST}_2}\right)$
ST_1	x	1	$x\cdot 1$	$x\log x$

Hence, the computation cost for one member to join the group is presented as follows:

$$ComputationCost_{Join} \le \frac{1}{x}\left(kN + k(x+1)\right) = \frac{k}{x}(N+x+1) \tag{9}$$

4.2. **Member leave.** In our proposed key tree topology, each subtree attached to the main tree's backbone is a variation of the structure of the STR protocol. Hence, for a member leave event, the rekeying overheads (such as rounds, communication, and computation cost) depend on the location of the deepest leave node [14]. According to [29], we evaluate the average case when the member leaves from the $\frac{g}{2}$-th subtree, $\mathrm{ST}_{\frac{g}{2}}$. Consider the circumstances that during k rekeying time intervals, the total join member quantity, x, is equal to the total leave member quantity.

As an example of the $\frac{g}{2}$-th subtree, $\mathrm{ST}_{\frac{g}{2}}$, in Table 2, the residual member quantity, $N_{\mathrm{ST}_{\frac{g}{2}}} = x\beta_2^{\frac{g}{2}}$, and the average rounds for each $\mathrm{ST}_{\frac{g}{2}}$'s remaining member is the sum of the $\mathrm{ST}_{\frac{g}{2}}$'s backbone depth (i.e., $\frac{g}{2}$) and the height of the ST_g (i.e., $x\beta_2^{\frac{g}{2}}$). Hence, the average rounds for all of $\mathrm{ST}_{\frac{g}{2}}$'s remaining members are $x\beta_2^{\frac{g}{2}}\left(\frac{g}{2}\right) + x\beta_2^{\frac{g}{2}}\left(\log x\beta_2^{\frac{g}{2}}\right) = x\beta_2^{\frac{g}{2}}\left(\frac{g}{2} + \log x\beta_2^{\frac{g}{2}}\right)$. Furthermore, all of the members in $\mathrm{ST}_{\frac{g}{2}} \sim \mathrm{ST}_g$ are unaltered, and only need $\frac{g}{2}$ rounds ($\mathrm{ST}_{\frac{g}{2}}$'s backbone depth).

For x members that leave from the main tree during time $0 \sim kR_I$ (only k batch rekeying intervals), the total rounds (for rekeying the N remaining members) are as follows:

$$\begin{aligned}
&TotalRounds_{Leave} = \tfrac{k}{x}\cdot\left(TotalRounds_{Leave}(Backbone) + TotalRounds_{Leave}(\mathrm{ST}_i)\right)\\
&= \tfrac{k}{x}\cdot\left(\begin{array}{l}\left(x\beta_2\left(\frac{g}{2}\right) + x\beta_2^2\left(\frac{g}{2}\right) + \cdots + x\beta_2^{\frac{g}{2}}\left(\frac{g}{2}\right) + \cdots + x\beta_2^{g-2}\cdot 3 + N_{\mathrm{ST}_2}\cdot 2 + x\cdot 1\right)\\ +\left(\begin{array}{l}\left(x\beta_2^{\frac{g}{2}}\log x\beta_2^{\frac{g}{2}} + \cdots + x\beta_2^{g-2}\log x\beta_2^{g-2}\right)\\ +\left(N_{\mathrm{ST}_2}\log N_{\mathrm{ST}_2}\right) + (x\log x)\end{array}\right)\end{array}\right)\\
&\le \tfrac{k}{x}\cdot\left(\begin{array}{l}\left(N\left(\frac{(3g-4)}{8}\beta_2+2\right)-x\right)_{(Appendix1)}\\ +\left(x\beta_2\left(\left(\frac{g}{2}-1\right)\log x + \left(\frac{(3g-4)(g-2)}{8}\right)\log\beta_2\right)\right)_{(Appendix2)}\\ +\left(N_{\mathrm{ST}_2}\log N_{\mathrm{ST}_2}\right)_{(Appendix3)} + (x\log x)\end{array}\right)
\end{aligned} \tag{10}$$

Therefore, the average leave rounds and average communication costs for one member to leave the group are as follows:

$$AverageRoungs_{Leave} \le \frac{1}{x}\left(\frac{1}{N}\left(TotalRounds\right)\right) \tag{11}$$

$$AverageCommunicationCost_{Leave} = 2\,(AverageRounds) \tag{12}$$

Because each subtree, ST_i, in the main tree is balanced, those subtrees can be regarded as a TGDH case. According to [11], the computation cost of each subtree, i, is equal to $2N_{\mathrm{ST}_i}$. In addition, the extra computation cost of the backbone $(\frac{g}{2})$ should be included. For $0 \sim kR_I$, the computation cost for one member to leave the group is presented as follows:

$$\begin{aligned}
&ComputationCost_{Leave}\\
&= \frac{k}{x}\left(2\left(N_{\mathrm{ST}_{\frac{g}{2}}} + \cdots + N_{\mathrm{ST}_3} + N_{\mathrm{ST}_2} + N_{\mathrm{ST}_1}\right) + \frac{g}{2}\right)\\
&= \frac{k}{x}\left(2\left(x\beta_2^{\frac{g}{2}} + \cdots + x\beta_2^{g-2} + N_{\mathrm{ST}_2} + x\right) + \frac{g}{2}\right)\\
&\le \frac{k}{x}\left(2\left(x\beta + \cdots + x\beta + N_{\mathrm{ST}_2} + x\right) + \frac{g}{2}\right)\\
&= \frac{k}{x}\left(2\left(\left(\frac{g}{2} - 2\right)x\beta_2 + N_{\mathrm{ST}_2} + x\right) + \frac{g}{2}\right)
\end{aligned} \tag{13}$$

4.3. **Comparison and discussion.** From the above mathematical analysis, the special batch rekeying factor $(\frac{k}{x})$ that appears in those equations is explained as follows.

1. $k = x$: This means that exactly one membership dynamic event occurs during each batch rekeying time interval. Roughly, it can be implied to match the individual-based rekeying case.
2. $k > x$: Because only x membership dynamic events (rekeying process) occur during k batch rekeying time intervals, the remaining k-x times cause no rekeying cost and can be disregarded. Just as in case 1, it can be implied to match the individual-based rekeying case roughly.
3. $k < x$: In the interval-based rekeying case, the total cost of x membership dynamic events is equally divided into k batch rekeying time intervals. Therefore, the average rekeying cost of an interval-based rekeying case is $(\frac{k}{x})$ times that of the individual-based rekeying case.

The average rekeying time costs for a single member join/leave event for the different schemes are presented in Table 3. Under the assumption that the key tree is completely

TABLE 3. Comparison of the average time cost, communication cost, and computation cost among PACK, TGDH, DST, and QB upon single member join/leave event

Single Member Join			
	Time Cost	Communication Cost	Computation Cost
PACK [11]	$1 \sim 2$	$2 \sim 4$	N
TGDH [8]	$\log N$	$2\log N$	$2N$
DST [10]	$1 + \log\log N$	$1 + \log\log N$	$N + \log N$
QB [9]	$\frac{k}{x}\log N$	$2\frac{k}{x}\log N$	$2\frac{k}{x}N$
Single Member Leave			
	Time Cost	Communication Cost	Computation Cost
PACK	$\log N$	$2\log N$	$(1 \sim 2.5)N$
TGDH	$\log N$	$2\log N$	$2N$
DST	$1 + \log N + \log\log N$	$2(1 + \log N + \log\log N)$	$3N$
QB	$\frac{k}{x}\log N$	$2\frac{k}{x}\log N$	$2\frac{k}{x}N$

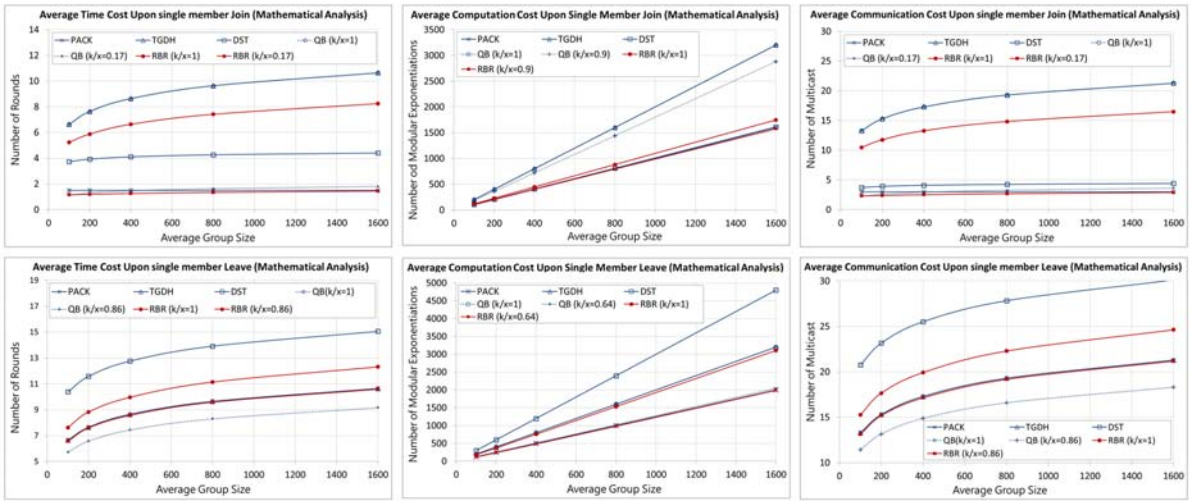


FIGURE 12. Comparison of the rekeying costs among RBR, PACK, TGDH, DST, and QB in mathematical analysis

balanced, when $\frac{k}{x} = 1$ (implying individual-based rekeying), the average rekeying rounds of QB are equal to TGDH, but higher than RBR. As shown in Figure 12, the average time cost and communication cost of RBR($\frac{k}{x} = 0.17$)/RBR($\frac{k}{x} = 0.86$) upon a single member join/leave event are respectively lower than those of the other schemes. Under our key tree topology, because the join tree capacity is lower than the group size, the computation cost of RBR ($\frac{k}{x} = 0.9$) upon a single member join event is lower than the other schemes, even the interval-based rekeying algorithm, QB. Moreover, upon a single member leave event, RBR ($\frac{k}{x} = 0.64$) is more efficient than the other schemes in relation to its computation overhead. Note that the mathematical analysis of Lee et al.'s QB [9] protocol is described in detail in Appendix 4. In addition, the computation cost of PACK is $(1 \sim 2.5)N$; we take the average cost $(1.25)N$ for the convenience of analysis.

5. **Simulation Results.** As mentioned above, in our simulations, we generate the member activities according to the following probabilistic models: the member join events according to a Poisson process with the average arrival rate λ. The leave events are defined by the members' staying time in the group, which follows an exponential distribution with mean α [13,26]. Therefore, the average group member size is $N = \alpha\lambda$. For each simulation, the initial group size is set to empty; we fix $\lambda = 1$, and vary α to get different average group member sizes. For each different average group member size case, there is a sequence of $100\alpha\lambda$ member join/leave events in this simulation. We set the rekeying time interval RI to 1, 2 and 4, and it can roughly take $\frac{\lambda}{R_I} = \frac{k}{x}$. We evaluate three cases as follows: 1) $R_I = 1 \rightarrow \frac{k}{x} = 1$: Implying that the interval-based rekeying case matches the individual-based rekeying case, then comparing that with three individual-based rekeying algorithms: PACK [11], TGDH [8] and DST [10]. 2) $R_I = 2 \rightarrow \frac{k}{x} = 0.5$: Evaluating the performance of the interval-based rekeying case using QB [9] and our proposed RBR, in a shorter rekeying time-interval case. 3) $R_I = 4 \rightarrow \frac{k}{x} = 0.25$: Evaluating the performance of the interval-based rekeying case in a longer rekeying time-interval case.

As shown in Figure 13, RBR ($R_I = 4$) has the lowest time cost among all of the schemes. Compared with PACK (the lowest one of the three individual-based rekeying algorithms), RBR ($R_I = 4$) has a more than 45% reduction upon a single member join and a more than 20% reduction upon a single member leave. It should be noted that the queue tree in the QB scheme will be merged to one of the shallowest leave node positions,

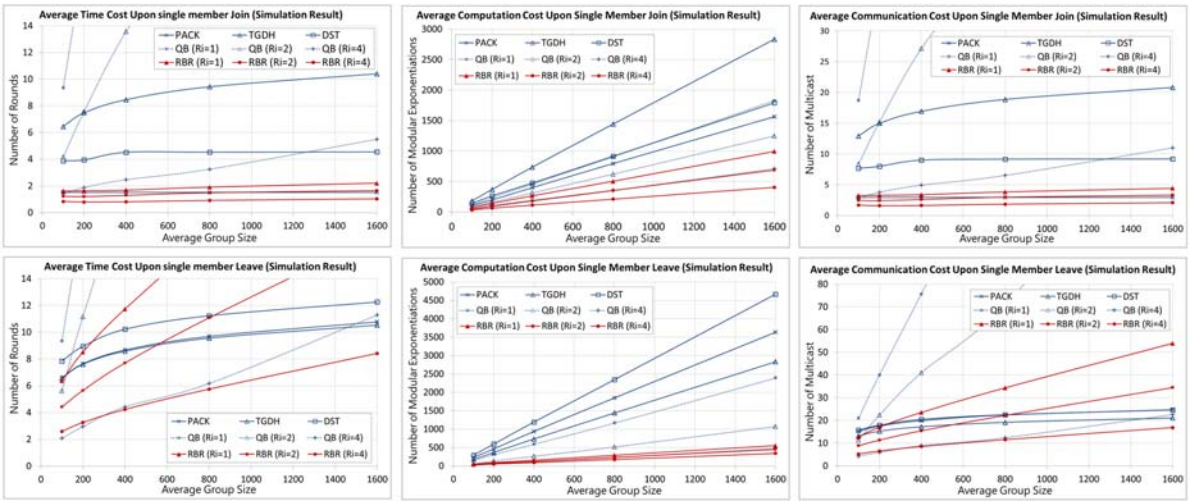


FIGURE 13. Comparison of the rekeying costs among RBR, PACK, TGDH, DST and QB in simulations

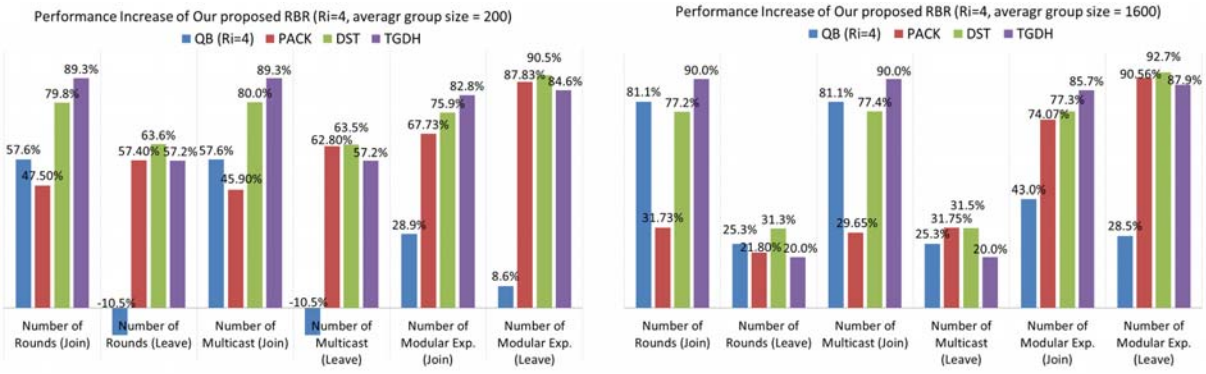


FIGURE 14. Performance increase rates among PACK, TGDH, DST and QB in simulations

but this position may not be the shallowest position in the whole key tree. This will lead to an unbalanced key tree and increase the rekeying time cost. Hence, the average time cost of QB ($R_I = 1$) is the highest of all the schemes and needs to increase R_I to reduce the overhead. In contrast to our proposed RBR, the join tree capacity and *Relocation* phase can prevent the spread of an unbalanced tree and increase the rekeying efficiency. Note that the overhead upon a single member join includes the *Relocation* and *ST-Merge* phases. In addition, because the longest-staying members (having higher leaving probabilities) will be moved to the subtree closest to the root node automatically, the computation and communication costs will be reduced significantly. At the same time, because the join tree has a lower capacity and is the closest to the root node, the computation and communication costs are lower than other schemes. As shown in Figure 14, we illustrate the performance increase rate for PACK, TGDH, DST, and QB in simulations. It should be noted that in the case of a smaller average group size (less than 300 approximately), the overhead of RBR ($R_I = 4$) is slightly higher than QB ($R_I = 4$) in the average time/communication cost upon a single member leave. However, the average cost computed as the average for the member join and leave cases is still the lowest one, and is more efficient, especially for a large average group size.

6. **Security Analysis.** In this section, we describe two security requirements as follows. 1) Forward/Backward secrecy: **forward/backward secrecy** implies that a compromise of the current key should not compromise any future/earlier key. In our proposed scheme, as the proofs of TGDH [8], when a member dynamic event occurs, the sponsor who is elected to deal with the rekeying process will first reselect its secret key. After receiving all of the blinded keys along his/her co-path, the sponsor computes a corresponding blinded key with its new secret key, and broadcasts this blinded key along its key-path. Finally, the new group key can be generated by all of the members and the forward and backward secrecy can be satisfied. 2) **Key confirmation**: key confirmation implies that all of the group members are actually holding the same group key. To achieve that confirmation, each member needs to reveal some information about its group key to all of the other members. In previous studies, several methods have been proposed but have suffered from high broadcast overhead and collusion attack [9,30]. Based on our proposed key tree topology, we can embed the concept of [9] to confirm the group key within the same subtree members to improve the above problems, which achieves key confirmation.

7. **Conclusions.** In this paper, based on the concepts of subtree division and residency time classification, we proposed four phases, including the *Queue-Tree* phase, *QT-Merge* phase, *ST-Merge* phase and *Relocation* phase, to deal with membership dynamic events. Without predicting the departure time of the leave member, the higher leaving probability members will automatically be moved to the subtree closest to the root node. Using the key tree topology of RBR ensures that the join/leave members' locations are as close as possible to the key tree root node, which significantly reduces the rekeying cost. Using both a mathematical analysis and simulation experiment, the performance of RBR was evaluated and found to achieve lower rekeying costs in terms of time, computation, and communication than the existing schemes.

REFERENCES

[1] N. Asokan and P. Ginzboorg, Key agreement in ad hoc networks, *Computer Communications*, vol.23, no.17, pp.1627-1637, 2000.

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

[3] C. K. Wong, M. Gouda and S. S. Lam, Secure group communications using key graphs, *IEEE/ACM Transactions on Networking*, vol.8, pp.1, pp.16-30, 2000.

[4] A. Perrig, D. Song and J. D. Tygar, ELK, a new protocol for efficient large-group key distribution, *Proc. of IEEE Symp. Security Privacy*, pp.247-262, 2001.

[5] H. Harney and C. Muckenhirn, Group key management protocol (GKMP) specification, *RFC*, vol.2093, 1997.

[6] P. Judge and M. Ammar, Gothic: A group access control architecture for secure multicast and anycast, *Proc. of IEEE INFOCOM*, pp.1547-1556, 2002.

[7] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor and B. Pinkas, Multicast security: A taxonomy and some efficient constructions, *Proc. of IEEE INFOCOM*, pp.708-716, 1999.

[8] Y. Kim, A. Perrig and G. Tsudik, Tree-based group key agreement, *ACM Transactions on Information and System Security*, vol.7, no.1, pp.60-96, 2004.

[9] P. C. Lee, C. S. Lui and K. Y. Yan, Distributed collaborative key agreement and authentication protocols for dynamic peer groups, *IEEE Transactions on Networking*, vol.14, no.2, pp.263-276, 2006.

[10] Y. Mao, Y. Sun, M. Wu and K. J. Liu, JET: Dynamic join-exit-tree amortization and scheduling for contributory key management, *IEEE Transactions on Networking*, vol.14, no.5, pp.1128-1140, 2006.

[11] W. Yu, Y. Sun and K. J. Liu, Optimizing the rekeying cost for contributory group key agreement schemes, *IEEE Transactions on Dependable and Secure Computing*, vol.4, no.3, pp.228-242, 2007.

[12] M. Burmester and Y. Desmedt, A secure and efficient conference key distribution scheme, *Proc. of Workshop Theory and Application of Cryptographic Techniques (EUROCRYPT'94)*, pp.275-286, 1994.
[13] K. C. Almeroth and M. H. Ammar, Multicast group behavior in the Internet's multicast backbone (mbone), *IEEE Comm. Magazine*, pp.124-129, 1977.
[14] Y. Kim, A. Perrig and G. Tsudik, Group key agreement efficient in communication, *IEEE Transactions on Computers*, vol.53, no.7, pp.905-921, 2004.
[15] I. Ingemarsson, D. T. Tang and C. K. Wong, A conference on key distribution system, *IEEE Transactions on Information Theory*, vol.28, no.5, pp.714-720, 1982.
[16] D. G. Steer, L. Strawczynski, W. Diffie and M. Wiener, A secure audio teleconference system, *Proc. of Advances in Cryptology*, pp.520-528, 1990.
[17] M. Steiner, G. Tsudik and M. Waidner, Diffie-Hellman key distribution extended to group communication, *Proc. of the 3rd ACM Conf. Computer and Comm. Security*, pp.31-37, 1996.
[18] M. Steiner, G. Tsudik and M. Waidner, Key agreement in dynamic peer groups, *IEEE Transactions on Parallel and Distributed Systems*, vol.11, no.8, pp.769-780, 2000.
[19] J. Snoeyink, S. Suri and G. Varghese, A lower bound for multicast key distribution, *Proc. of INFOCOM'01*, 2001.
[20] M. Rajaram and D. Thilagavathy, An interval based contributory key agreement, *Proc. of International Conference on Wireless Communication and Sensor Computing*, pp.1-6, 2010.
[21] M. Rajaram and D. Thilagavathy, Authenticated collaborative key agreement for dynamic peer groups, *Proc. of International Conference on Man-Machine Systems*, pp.11-13, 2009.
[22] M. Rajaram and D. Thilagavathy, An interval-based contributory key agreement, *International Journal of Network Security*, vol.13, no.2, pp.92-97, 2011.
[23] M. Rajaram, D. Thilagavathy and N. K. Devi, Resilient group key agreement protocol with authentication security, *Proc. of International Conference on Innovative Computing Technologies*, pp.1-6, 2010.
[24] J. Zhang, J. G. Luo, B. Li and S. G. Yang, SIKAS: A scalable distributed key management scheme for dynamic collaborative groups, *Proc. of IEEE International Conference on Multimedia and Expo*, Canada, pp.1205-1208, 2006.
[25] J. Zhang, B. Li, C. X. Chen and P. Tao, EDKAS: A efficient distributed key agreement scheme using one way function trees for dynamic collaborative groups, *Proc. of IMACS Multiconference on Computational Engineering in Systems Applications*, Beijing, pp.1215-1222, 2006.
[26] K. C. Almeroth, A long-term analysis of growth and usage patterns in the multicast backbone (mbone), *Proc. of INFOCOM'00*, vol.2, pp.824-833, 2000.
[27] A. Fekete, N. Lynch and A. Shvartsman, Specifying and using a partionable group communication service, *Proc. of the 16th ACM Symp. Principles of Distributed Computing*, pp.53-62, 1997.
[28] K. Becker and U. Wille, Communication complexity of group key distribution, *Proc. of the 5th ACM Conf. Computer and Communications Security*, pp.1-6, 1998.
[29] Y. Amir, Y. Kim, C. Rotaru, J. Schultz, J. Stanton and G. Tsudik, On the performance of group key agreement protocols, *ACM Transactions on Information and Systems Security (TISSEC)*, vol.7, no.3, pp.1-32, 2004.
[30] M. Just and S. Vaudenay, Authenticated multi-party key agreement, *Proc. of Advances in Cryptology (ASIACRYPT'96), LNCS*, vol.1163, pp.36-49, 1996.

Appendix 1. The upper bound of $x\beta_2\left(\frac{g}{2}\right)+x\beta_2^2\left(\frac{g}{2}\right)+\cdots+x\beta_2^{g-2}\cdot 3+N_{\mathrm{ST}_2}\cdot 2+x\cdot 1$ shown in Equation (11).

$$\begin{aligned}
&\left(x\beta_2\left(\tfrac{g}{2}\right)+x\beta_2^2\left(\tfrac{g}{2}\right)+\cdots+x\beta_2^{\frac{g}{2}}\left(\tfrac{g}{2}\right)+\cdots+x\beta_2^{g-2}\cdot 3+N_{\mathrm{ST}_2}\cdot 2+x\cdot 1\right)\\
&=\begin{pmatrix}(x\beta_2+x\beta_2^2+\cdots+x\beta_2^{g-2}+N_{\mathrm{ST}_2}+x)\\ +(x\beta_2+x\beta_2^2+\cdots+x\beta_2^{g-2}+N_{\mathrm{ST}_2})\\ +(x\beta_2+x\beta_2^2+\cdots+x\beta_2^{g-2})\\ +\cdots\\ +(x\beta_2+x\beta_2^2+x\beta_2^3+\cdots+x\beta_2^{\frac{g}{2}})\end{pmatrix}\leq\begin{pmatrix}N\\ +(N-x)\\ +(x\beta_2+x\beta_2^2+\cdots+x\beta_2^{g-2})\\ +\cdots\\ +(x\beta_2+x\beta_2^2+\cdots+x\beta_2^{\frac{g}{2}})\end{pmatrix}\\
&\leq\begin{pmatrix}N\\ +(N-x)\\ +(x\beta_2+x\beta_2+\cdots+x\beta_2)\\ +\cdots\\ +(x\beta_2+x\beta_2+\cdots+x\beta_2)\end{pmatrix}=x\beta_2\tfrac{\left(\frac{3}{2}g-2\right)\left(\frac{1}{2}g-2\right)}{2}+2N-x\\
&=x\beta_2\tfrac{(3g-4)(g-4)}{8}+2N-x\leq x\beta_2\tfrac{(3g-4)g}{8}+2N-x\\
&=N\left(\tfrac{(3g-4)}{8}\beta_2+2\right)-x\leq N\left(\tfrac{(g-2)}{2}+2\right)-x=N\tfrac{(g+2)}{2}-x
\end{aligned}\tag{14}$$

Appendix 2. The upper bound of $x\beta_2(\log x\beta_2^{\frac{g}{2}})+\cdots+x\beta_2^{g-2}(\log x\beta_2^{g-2})$ shown in Equation (11).

$$\begin{aligned}
&x\beta_2(\log x\beta_2^{\frac{g}{2}})+\cdots+x\beta_2^{g-2}(\log x\beta_2^{g-2})\leq\left(x\beta_2(\log x\beta_2^{\frac{g}{2}})+\cdots+x\beta_2(\log x\beta_2^{g-2})\right)\\
&=\left(x\beta_2\left(\log x+\log\beta_2^{\frac{g}{2}}\right)+\cdots+x\beta_2\left(\log x+\log\beta_2^{g-2}\right)\right)\\
&=\left(x\beta_2\left(\left(\tfrac{g}{2}-1\right)\log x+\left(\tfrac{(3g-4)(g-2)}{8}\right)\log\beta_2\right)\right)
\end{aligned}\tag{15}$$

Appendix 3. The number of members in ST_2. If the average group member size is N, and $N_{ST_1}=x$.

$$\begin{aligned}
N_{ST_2}&=N-N_{ST_1}-(N_{ST_3}+N_{ST_4}+\cdots+N_{ST_g})\\
&=N-x-(x\beta_2+x\beta_2^2+\cdots+x\beta_2^{g-3})\\
&=N-x(1+\beta_2+\beta_2^2+\cdots+\beta_2^{g-3})\\
&=N-x\left(1+\frac{\beta_2(1-\beta_2^{g-3})}{1-\beta_2}\right)
\end{aligned}\tag{16}$$

Appendix 4. Performance evaluation of the Lee et al.'s QB algorithm [9]. To facilitate the performance analysis, QB can be regarded as a special case of our proposed scheme where the join tree capacity is unlimited is larger than the average group size N. Because all member dynamic events are occurred in the join tree (i.e., do not require the main tree and subtrees), the *Relocation* and *ST-Merge* phase are not required. As shown in Table 4, RI new members join into the queue tree between each adjacent rekeying time interval (i.e., RI members leave from the join tree simultaneously). Hence, the number of join tree residual members is N-RI, which needs log(N-RI) rounds for the queue tree to be immigrated to the join tree and deal with the rekeying process. Consider the circumstances that during k times rekeying time intervals, the total members in the join tree is equal to x. We evaluate the average rounds and average communication costs for one join/leave events are as follows:

$$AverageRounds_{join/leave}=\frac{1}{x}\left((k-1)\log(N-\lambda R_I)\right)\leq\frac{k}{x}\left(\log N\right)\tag{17}$$

$$AverageCommunicationCost_{Join/Leave} = 2\left(AverageRounds_{join/leave}\right) \tag{18}$$

During k times rekeying time intervals, the total computation costs for N group members are $2k$(N-RI). Therefore, the computation costs for one join/leave events are as follows:

$$TotalComputationCost_{join/leave} = \frac{1}{x}\left(2k \cdot (N - \lambda R_I)\right) \leq 2\frac{k}{x}N \tag{19}$$

TABLE 4. Cost analysis for member join/leave events in Lee et al.'s QB algorithm

$Time$	0	R_I	$2R_I$	...	kR_I
N_{QT}	0	λR_I	λR_I	...	λR_I
N_{JT}	0	0	$\lambda R_I \beta_1$	...	$\lambda R_I \left(1 + \beta_1 + \beta_1^2 + \cdots + \beta_1^{k-2}\right) \beta_1$
Rounds	0	1	$1 + \log \lambda R_I \beta_1$	...	$1 + \log \lambda R_I \left(1 + \beta_1 + \beta_1^2 + \cdots + \beta_1^{k-2}\right) \beta_1$