# Optimizing Centralized Secure Group Communications with Binary Key Tree Recomposition

Jen-Chiun Lin* Chien-Hua Tzeng† Feipei Lai‡ Hung-Chang Lee§

## Abstract

*The growth of the Internet inspires lots of new network applications, and many of them are based on group communication models, In this paper, we propose a novel approach to reduce the path length of a binary key tree managed by a centralized group key server by recomposing it, such that subsequent group operations can benefit from fewer auxiliary key updates, fewer encryptions, and less multicast bandwidth. The server can recompose the key tree in a join or a leave operation without the use of additional auxiliary keys or encryptions. An optimal composition algorithm is presented. It is shown in our analysis that the algorithm is effective to reduce the path length of a binary key tree in join operations and leave operations.*

## 1. Introduction

The growth of the Internet inspires lots of new network applications, and many of them are based on group communication models [11, 5, 10], where a message originated from a user has to be sent to other users in the group. Typical group applications include distance education, video conferences, collaborative work, and distributed interactive simulation. Group communications can take advantages of a more efficient multicast service [4, 1], which is capable of sending a message to multiple destinations. Communication confidentiality is crucial for applications communicate via an open network environment, such as the Internet. A unique characteristic of group communications is that the parties participating in the communication can change with time, which complicates key management. While there is an abundance of solutions for traditional, point-to-point secure communications, they are usually not directly applicable to a secure group communication system due to poor performance and scalability.

A popular approach to achieve group communication confidentiality is to have a group key shared by all users, and the key can be used to encrypt messages sent to other users to prevent eavesdropping of outsiders. Secure group communication protocols must ensure that, for a user that does not belong to the group, it is computationally infeasible to derive any group key, and for a current group user, it is computationally infeasible to derive group keys used before its participation and after its departure. So each time a new user joins or a current user leaves the group, the group key of every group user has to be securely updated to preserve key secrecy [6].

Binary key trees are widely used by secure group communication systems with centralized group key servers. Since it is desirable to keep the binary key tree balanced to reduce the cost of key updates caused by group operations, we propose a novel idea to recompose a binary key tree to reduce its path length. Subsequent group operations can benefit from the reduction of path length that the required key updates are fewer. It can be shown that, recomposition can be done without additional key or encryption overhead. An optimal composition algorithm is presented to maximize the reduction of path length of a binary tree.

The rest of the paper is organized as follows. Section 2 gives an overview of binary key tree and tree recomposition. An optimal composition algorithm is presented. Section 3 demonstrates how recomposition works with join and leave operations. The effectiveness of the algorithm and its complexity are analyzed in section 4. Finally, we give the related work of this field and the conclusion of our work.

## 2. Binary Key Tree Recomposition

Binary key trees are special cases of key graphs [12], and they are favored for their simplicity and balance of cost between join operations and leave operations. A binary key tree is composed of a set of k-nodes, and each k-node holds

*Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan

†Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan

‡Department of Electrical Engineering & Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan

§Department of Information Management, Tamkang University, Taipei 251, Taiwan

Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA'04)
0-7695-2051-0/04 $ 20.00 © 2004 IEEE

a secret key. Every user of the group is uniquely associated with a leaf k-node of the tree. There are two kinds of keys: an individual key is placed at a leaf k-node and is only shared by the server and the corresponding user; an auxiliary key is placed at an internal k-node and is shared by the server and the users whose associated leaf k-nodes are descendants of that k-node. An individual key is established at the time a user joins the group, and it is usually not changed until the user leaves. On the other hand, auxiliary keys are frequently updated due to group operations. The root k-node stores the group key, which is a special auxiliary key shared by all users.

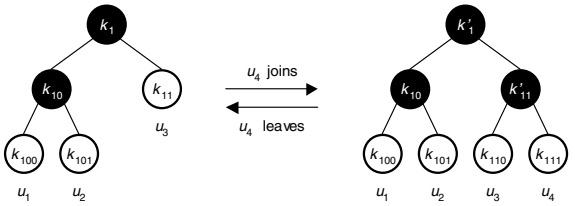


**Figure 1. Join operation and leave operation.**

Figure 1 gives a typical binary key tree supporting join operations and leave operations. Assume that user $u_4$ joins the group. The server adds a new internal k-node and a leaf k-node, and updates the auxiliary keys of all the internal k-nodes which are ascendants of $k_{111}$. Then, it sends messages to all users to update their keys:

$$\begin{aligned} s \rightarrow u_3, u_4 &: [k'_{11}]_{k_{110}} || [k'_{11}]_{k_{111}}, \\ s \rightarrow u_1, u_2, u_3, u_4 &: [k'_1]_{k_{10}} || [k'_1]_{k'_{11}}. \end{aligned}$$

If now that user $u_4$ wishes to leave the group, the server removes k-node $k_{111}$, its parent k-node $k'_{11}$, updates the auxiliary keys of all the internal k-nodes that are ascendants of $k'_{11}$, and sends remaining users:

$$s \rightarrow u_1, u_2, u_3 : [k_1]_{k_{10}} || [k_1]_{k_{11}}$$

Since a join operation or a leave operation causes the keys of the k-nodes along a particular path to be updated, the cost is proportional to the length of the path. It can be easily concluded that while the tree is balanced enough, both a join operation and a leave operation cost $O(\log N)$ key updates for a group with $N$ users. For a user in the group, it only stores one individual key and $O(\log N)$ additional auxiliary keys. Therefore, it is crucial to reduce the path length of a binary key tree to keep it balanced.

## 2.1. Binary Key Tree Recomposition

To address the binary key tree balancing problem, we take a totally different point of view that a group operation will decompose the key tree into several subtrees, which are going to be composed to a new key tree by the server. Hence, the server can try to reduce the path length of the key tree via a suitable composition algorithm.

**Definition 1** *For a binary tree $T$, an ordered set $D_k(T) = \{T_1, T_2, ..., T_k\}$, $k \geq 1$, is said to be a decomposition of $T$ of order $k$, if the set of all leaf nodes of $T$ can be partitioned into $k$ nonempty subsets, $S_1$, $S_2$,..., $S_k$, and all the nodes in every $S_i$ are the leaf nodes of some subtree $T_i$ of $T$, for all $1 \leq i \leq k$.*

**Definition 2** *A composition of an order set of binary trees $D = \{T_1, T_2, ..., T_k\}$, $k \geq 1$, is a binary tree $C(D)$, where there is an decomposition $D_k$ such that $D_k(C(D)) = D$.*

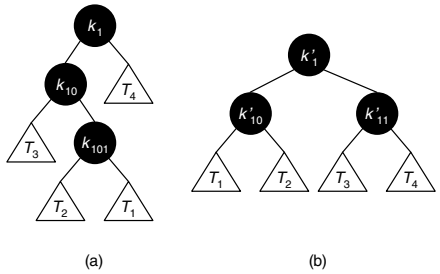


**Figure 2. (a) $T$ (b) $T'$, a recomposition of $T$.**

Figure 2 illustrates the concept of decomposition, composition and recomposition. For the key tree in figure 2a, $D = \{T_1, T_2, T_3, T_4\}$ is a decomposition of $T$ by definition 1. According to definition 2, the tree $T'$ in figure 2b is a composition of $D$, since $D$ is a valid decomposition of $T'$. The tree $T'$ is also called a recomposition of $T$ with respect to decomposition $D$. In the example we find that, for both trees, the server has to use six encryptions to update three new auxiliary keys. Theorem 1 shows that in general, recomposition does not incur encryption overhead.

**Definition 3** *A binary tree is said to be fully-branched, if every internal node has two child nodes.*

**Theorem 1** *If a join operation or a leave operation happens along a path of a fully-branched binary key tree $T$, it decomposes $T$ to a set of binary key trees $D$. The cost of auxiliary key generation and encryption is the same to update both $T$ and a fully-branched recomposition $C(D)$ in such a group operation.*

## 2.2. An Optimal Composition Algorithm

In section 2.1, it has been shown that the server can recompose the binary key tree the way it chooses without additional auxiliary keys and encryptions. It is obvious that given a set of binary key trees, there must be an arrangement that can minimize the path length of the composed

binary key tree. Here, we introduce an algorithm that gives maximum path length reduction.

A group operation can decompose the original binary key tree to a set of subtrees $D$, the algorithm is to compose a new binary key tree $C_O(D)$, such that its path length is not greater than that of any other valid composition $C(D)$. Assume that $|D| = k > 1$, and for each $T_i \in D$, $1 \leq i \leq k$, its path length and number of leaf node are $w(T_i) = w_i$ and $n(T_i) = n_i$, respectively. The algorithm is listed in Figure 3.

**algorithm** $C_O(D)$
  **while** $|D| > 1$ **do**
    $T_l \leftarrow$ ExtractMin($D$)
    $T_r \leftarrow$ ExtractMin($D$)
    $T_0 \leftarrow$ BuildTree($T_l, T_r$)
    **if** $|D| = 0$ **then return** $T_0$
    $D \leftarrow D \cup \{T_0\}$
**algorithm** ExtractMin($D$)
  *find* $T_0 \in D$ *such that* $(\forall T_i \in D) n_0 \leq n_i$
  $D \leftarrow D - \{T_0\}$
  **return** $T_0$
**algorithm** BuildTree($T_l, T_r$)
  $T_0 \leftarrow$ (*new k-node*, $T_l, T_r, n_l + n_r$)
  **return** $T_0$

**Figure 3. Optimal Composition Algorithm.**

Algorithm *ExtractMin* removes a binary key tree with the minimum number of leaf k-node from $D$. Algorithm *BuildTree* builds a new binary key tree with a root k-node, the left subtree, the right subtree, and an additional field to indicate the number of leaf k-nodes. It is interesting that no $w_i$ is considered by the algorithm, and, from theorem 2, they are safe to be ignored.

**Theorem 2** *If $D$ is an ordered set of $k \geq 1$ fully-branched binary trees, and a fully-branched binary tree $T = C(D)$ is composed of $D$, the path length of $T$ is*

$$w(T) = \sum_{i=1}^{k} w(T_i) + v_T(D),$$

*where $v_T(D) = \sum_{i=1}^{k} n(T_i) d_T(T_i)$, and $d_T(T_i)$ is the depth of the root node of $T_i$ in $T$.*

It implies that only the numbers of leaf k-nodes in subtrees are the determining factor. Theorem 3 guarantees us that $v_T(D)$ is minimal, so is the path length of the binary key tree composed by algorithm *OptCompose*.

**Theorem 3** *If $D$ is an ordered set of $k \geq 2$ binary trees and tree $T = C_O(D)$, then $v_T(D)$ is minimal.*

Note that the proofs of these theorems are omitted for brevity.

## 3. Group Operations with Recomposition

In this section, we focus on how a centralized server maintaining the binary key tree of the group can perform recomposition in user join operations and user leave operations. Integrating the recomposition operation to other group operations rather than making it another independent one because it involves no key or encryption overhead from theorem 1.

### 3.1. User Join Operation

Each time a new user wishes to join a group, it issues a user join request to the group manager. The join process is two-phased. In the setup phase, the server authenticates the identity of the new user, and negotiates an individual key $k$ shared with the user. Usually, $k$ will be used throughout the membership of the user until it leaves the group, with the exception that it might be explicitly updated for security concern or other special reasons. If everything goes fine, the server enters the update phase, and runs the protocol in figure 4.

**algorithm** UserJoinWithRecomposition()
  *select a path $P$ of the shortest length*
  *decompose the key tree to a set $D$ of key trees along $P$*
  $T_0 \leftarrow$ *new k-node for the new user*
  $D \leftarrow D \cup \{T_0\}$
  $T \leftarrow C_O(D)$
  **foreach** *new k-node* $\in T$ **do**
    *assign a new auxiliary key*
    *encrypt the auxiliary key with the keys of its child k-nodes*
  $m \leftarrow$ *encode update message*
  **multicast** $m$

**Figure 4. User join protocol.**

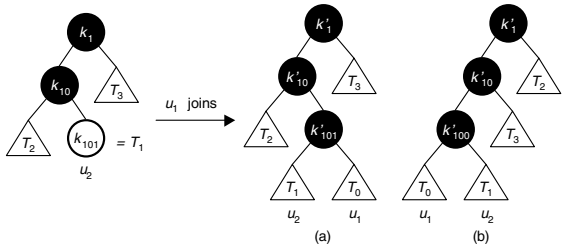


**Figure 5. Join: (a) ordinary (b) recomposed ($n_1 = 1$, $n_2 = 4$, $n_3 = 3$).**

We use the key trees in figure 5 to explain recomposition in join operations. In the figure, a new user $u_1$ is going to join the group, and the path length from k-node $k_{101}$ to k-node $k_1$ is the shortest. The server selects k-node $k_{101}$ as the place to insert a new k-node $k'_{101}$, whose children are $T_1$, which contains the k-node associated with $u_2$ and $T_0$, which contains the k-node for $u_1$, and the key tree will be

like the one in figure 5a. Algorithm $C_O$ will build a new key tree as figure 5b.

### 3.2. User Leave Operation

Each time a user is going to leave a group, the member leave operation is triggered, and the process is also two-phased. For a voluntary leave request from the user who is going to leave, the server verifies the authenticity of the request in the setup phase. For compulsory leave operations triggered by special events, such as network failure, that force the server to expel a user from the group, the setup phase is skipped. In the update phase, the server removes the k-node associated with the leaving user and updates the key tree. The protocol is listed in figure 6.

**algorithm** UserLeaveWithRecomposition()
  *find path $P$ of the k-node associated with the leaving user*
  *decompose the key tree to a set $D$ of key trees along $P$*
  $T_1 \leftarrow$ *the tree of the k-node associated with leaving user*
  $D \leftarrow D - \{T_1\}$
  $T \leftarrow C_O(D)$
  **foreach** *new k-node* $\in D$ **do**
    *assign a new auxiliary key*
    *encrypt the auxiliary key with the keys of its child k-nodes*
  $m \leftarrow$ *encode update message*
  **multicast** $m$

**Figure 6. User leave protocol.**

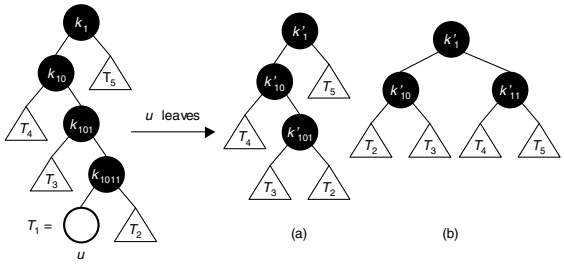


**Figure 7. Leave: (a) ordinary (b) recomposed ($n_2 = 1$, $n_3 = 2$, $n_4 = 2$, $n_5 = 2$).**

Figure 7 shows the difference of an ordinary key tree and its recomposition. In the figure, user $u$ just left the group, and its u-node is associated with k-node $k_{10110}$. For an ordinary leave operation, the server replaces the parent k-node $k_{1011}$ with another child subtree $T_2$, as shown in figure 7a. Algorithm $C_O$ will compose a new key tree as figure 5b.

## 4. Analysis and Discussion

In this section, the effectiveness of algorithm $C_O$ is analyzed. To simplify the analysis, it is assumed that the binary key tree can be equiprobable any one of all possible binary key trees. The amortized cost of the algorithm is shown in the end of the section, and system performance tuning suggestions are also provided.

### 4.1. Effectiveness of the Optimal Algorithm

Algorithm $C_O$ is consider effective if it can compose a new key tree whose path length is shorter than that of an un-recomposed one. If it cannot be achieved for a key tree with respect to a decomposition $D$, the key tree is said to be optimal with respect to $D$.

Let $\mathcal{D}^J_{N,k}$ be the set of decompositions of order $k$ of a key tree with $N$ leaf k-nodes by a join operation, that the key tree is optimal with respect to any such decomposition. Similary, $\mathcal{D}^L_{N,k}$ denotes the set of decompositions that occur in a leave operation. The probabilities that these trees occur in a join operation and a leave operation are

$$p^J_{N,k} = \sum\nolimits_{D \in \mathcal{D}^J_{N,k}} \frac{2^{k-1} \prod_{i=0}^{k} b_{n_i - 1}}{N b_{N-1}},$$

$$p^L_{N,k} = \sum\nolimits_{D \in \mathcal{D}^L_{N,k}} \frac{2^{k-1} \prod_{i=2}^{k} b_{n_i - 1}}{N b_{N-1}},$$

respectively, where $b_n$'s are Catalan numbers, and each $n_i$ is the number of leaf k-nodes of a subtree in $D$. Further, the probabilities of the occurrences of optimal trees in a join operation and a leave operation can be computed by summing the probabilities over all possible values of $k$, and we denote them $p^J_N$ and $p^L_N$, respectively. All these probabilities are calculated for selected values of $N \leq 512$ and all possible values of $k$. The data are depicted in figure 8.

The analysis shows that $p^J_{128} \approx 0.00978$ and $p^L_{256} \approx 0.00810$. In other words, the chances the algorithm $C_O$ improves the path length of the key tree in a join and a leave operation are greater than 99% for $N$ greater than 128 and 256, respectively. For $N \leq 512$ and $k \geq 15$, there are no optimal trees with respect to any decompositions.

The lower bounds of the reduction of average path length of a join operation, $\overline{r}^J_{N,k}$, and a leave operation, $\overline{r}^L_{N,k}$, are computed. For clarity, only values for $k \leq 20$ are shown in figure 9. For small $k$ values, the values of average path length reduction become negative due to our overestimation of the path length of an optimally recomposed tree, and path length reductions cannot be negative in practice. For both join operations and leave operations, as $N$ increases, the amount of reduction increases almost linearly. Therefore, after the server recomposes the key tree, it is very likely that the next group operation involves one less auxiliary key update, which reduces both encryption time and multicast bandwidth.

### 4.2. Complexity of the Optimal Algorithm

Heap [2] is a data structure that is both time and space efficient to support algorithm *ExtractMin* in figure 3. The

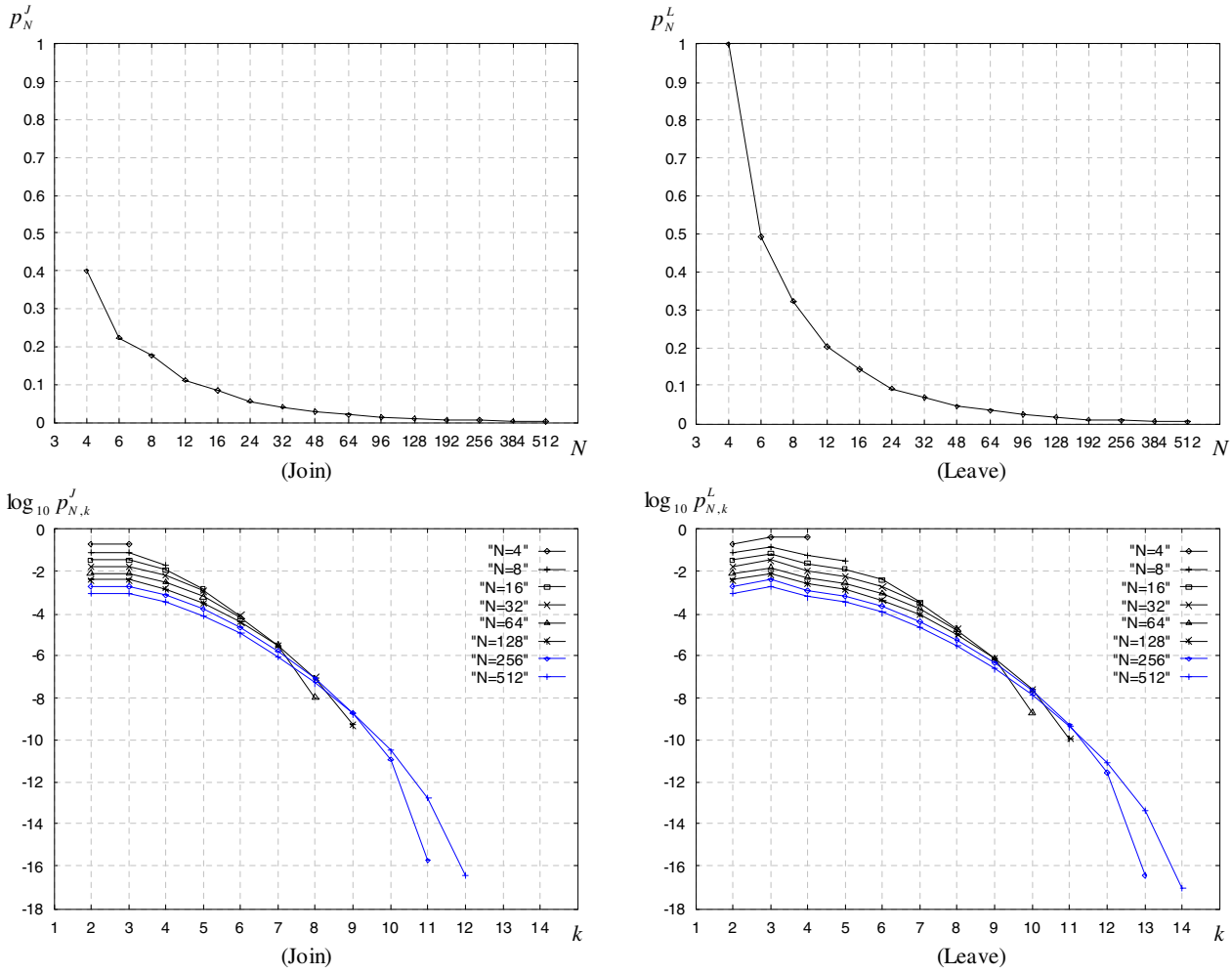


**Figure 8. Probabilities of the occurrences of optimal trees.**

cost to optimally compose a binary tree with an ordered set $D$ of binary key trees using a *min-heap* is $O(|D| \log |D|)$. Theorem 4 ensures us that the amortized cost of algortihm $C_O$ is $O(\log N \log \log N)$, where $N$ is the size of the group.

**Theorem 4** *For a group with one user in the beginning, the amortized tree height after a sequence of join operations and leave operations with optimal recomposition is* $O(\log N)$*, where* $N$ *is the maximum number of users the group ever has.*

### 4.3. Performance Tuning

There are several observations that can help us tune the system performance:

- The server should not perform the optimal composition algorithm in every group operation to reduce the computation overhead.
- If $k$ is large, to save time, the server can only recompose those subtrees that are near the end of the path selected by a group operation.
- The server can compute the amount of path length reduction to decide whether recomposition is necessary.
- If the server has extra computation power, it can precompute a hypothetical recomposed tree for a join operation.

## 5. Related Work

The related researches include secure multicast routing [8], reliable group key delivery [13], and special cryptographic algorithms [14]. Moyer *et al.* [7] proposed techniques to keep a centralized binary key tree balanced. They can either use a modified leaf deletion operation or a periodic rebalancing approach. Snoeyink *et al.* [9] showed that

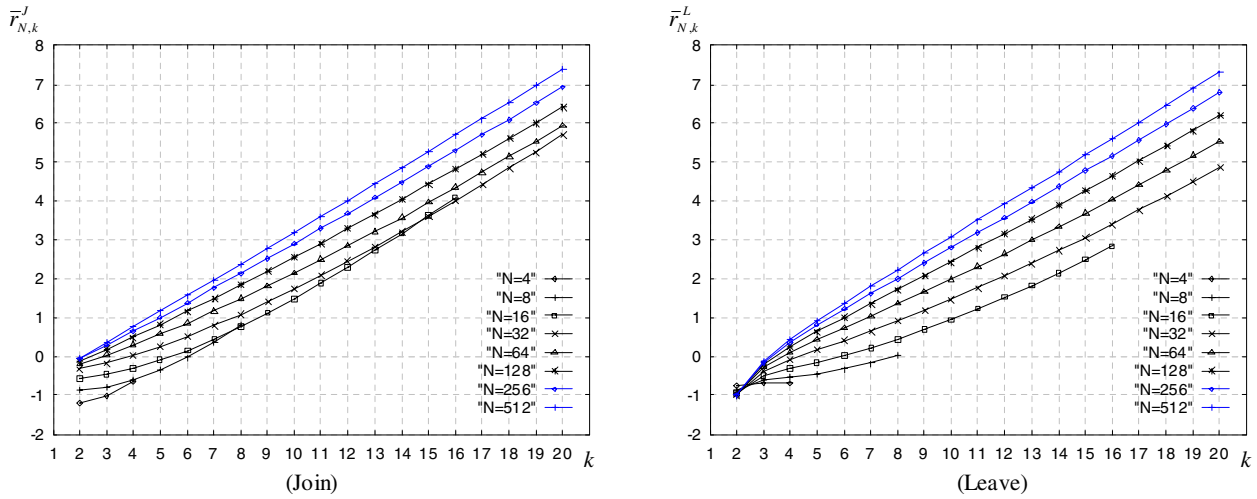


**Figure 9. Lower bounds of average path length reduction.**

an optimal key distribution tree for a centralized key server with simple private key encryption is a special case of 2-3 tree. Goshi and Ladner [3] also proposed algorithms to height-balance or weight-balance the 2-3 key trees.

## 6. Conclusion and Future Work

We present a novel idea, binary key tree recomposition, to balance a binary key tree by reducing its path length, such that subsequent group operations can benefit from fewer auxiliary key updates, fewer encryptions, and less multicast bandwidth. It is attractive that recomposition can be used in join operations and leave operations without extra auxiliary keys or encryptions. An optimal composition algorithm is also presented in the paper. Our analysis shows that the algorithm is effective to reduce the path length of a binary key tree in join operations and leave operations.

The concept of recomposition can be extended to more general d-ary trees. Optimally recomposing a key tree in every join operation or leave operation is usually not necessary due to the diminishing return of path length reduction after repeatedly recompositions, and it depends on actual communication environments and behavior of group users that when or how often should recomposition be used. We are also working on other composition algorithms that do not produce optimal results but run faster.

## References


[1] Y. Chawathe, S. McCanne, and E. Brewer. RMX: Reliable multicast for heterogeneous networks. In *Proceedings of IEEE INFOCOM*, pages 795–804, March 2000.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Revest. *Introduction to Algorithms*. The MIT Press, 1989.

[3] J. Goshi and R. E. Ladner. Algorithms for dynamic multicast key distribution trees. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 243–251, Boston, Massachusetts, 2003. ACM Press.

[4] H. Holbrook and D. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. In *Proceedings of the ACM SIGCOMM*, pages 65–78, Cambridge, Massachusetts, USA, 1999.

[5] T. Liao. Webcanal: a multicast web application. In *Proceedings of the Sixth Intenational WWW Conference*, Santa Clara, California, April 1997.

[6] S. Mittra. Iolus: A framework for scalable secure multicasting. In *ACM SIGCOMM*, pages 277–288, Sept. 1997.

[7] M. Moyer, J. Rao, and P. Rohatgi. *Maintaining Balanced Key Trees for Secure Multicast*. IETF, June 1999. draft-irtf-smug-key-tree-balance-00.txt.

[8] C. Shields and J. J. Garcia-Luna-Aceves. KHIP - a scalable protocol for secure multicast routing. In *SIGCOMM*, pages 53–64, 1999.

[9] J. Snoeyink, S. Suri, and G. Varghese. A lower bound for multicast key distribution. In *Proceedings of IEEE INFOCOM*, volume 1, pages 422–431, April 2001.

[10] T. Tung. Mediaboard: A shared whiteboard application for the mbone. Master's thesis, U.C. Berkeley, 1998.

[11] T. Turletti and C. Huitema. Video-conferencing on the internet. *ACM/IEEE Trans. Networking*, 4(3):340–51, June 1996.

[12] C. Wong, M. Gouda, and S. Lam. Secure group communications using key graphs. *Proceedings of the ACM SIGCOMM'98*, pages 68–79, Sept. 1998.

[13] C. K. Wong and S. S. Lam. Keystone: A group key management service. In *International Conference on Telecommunications, ICT 2000*, 2000.

[14] K. P. Wu, S. J. Ruan, F. Lai, and C. K. Tseng. On key distribution in secure multicasting. In *Proceedings of the 25th Annual IEEE Conference on Local Computer Networks*, pages 208–212, Nov. 2000.