Optimal Tree Structures for Group Key Tree Management Considering Insertion and Deletion Cost*

Weiwei Wu[1], Minming Li[2], and Enhong Chen[3]

[1] USTC-CityU Joint Research Institute
Department of Computer Science, University of Science and Technology of China,
Department of Computer Science, City University of Hong Kong
wweiwei2@cityu.edu.hk
[2] Department of Computer Science, City University of Hong Kong
minmli@cs.cityu.edu.hk
[3] Department of Computer Science, University of Science and Technology of China
cheneh@ustc.edu.cn

Abstract. We study the optimal structure for group broadcast problem where the key tree model is extensively used. The objective is usually to find an optimal key tree to minimize the cost based on certain assumptions. Under the assumption that n members arrive in the initial setup period and only member deletions are allowed after that period, previous works show that when only considering the deletion cost, the optimal tree can be computed in $O(n^2)$ time. In this paper, we first prove a semi-balance property for the optimal tree and use it to improve the running time from $O(n^2)$ to $O(\log \log n)$. Then we study the optimal tree structure when insertion cost is also considered. We show that the optimal tree is such a tree where any internal node has at most degree 7 and children of nodes with degree not equal to 2 or 3 are all leaves. Based on this result we give a dynamic programming algorithm with $O(n^2)$ time to compute the optimal tree.

1 Introduction

Many recent works have researched group broadcast problem due to its cost effectiveness in the applications requiring content security. The applications based on multicast can be divided into two types, one-to-many (e.g., television broadcast, pay per view) and many-to-many (e.g., teleconference, collaborate work, distributed interactive game). All of them require content security which means only authorized users are allowed to access the data broadcasted. Moreover, to

* This work was supported in part by the National Basic Research Program of China Grant 2007CB807900, 2007CB807901, a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 116907], Program for New Century Excellent Talents in University (No.NCET-05-0549) and National Natural Science Foundation of China (No.60775037).

X. Hu and J. Wang (Eds.): COCOON 2008, LNCS 5092, pp. 521–530, 2008.

deliver messages on insecure channels, we should guarantee confidentiality when users dynamically change in the group with the help of cryptography. Two kinds of confidentiality are usually considered in the literature: future confidentiality (to prevent users deleted from the group from accessing any future keys which will be used to encrypt data) and past confidentiality (to prevent users newly added into the group from accessing any past keys used to encrypt data).

To satisfy these security requirements, the basic strategy is to update the group key whenever a user is deleted from or added into the group. The group controller (GC) will maintain a key structure for the whole group. A recent survey on key management for secure group communications can be found in [2]. However, since encryption is most time consuming, the critical problem is how to decrease the number of encryptions when group members dynamically change. Key tree model proposed by Wong et al. [7] is widely used for key management problem. In this model, it is assumed that a leaf node represents a user and stores his individual key and an internal node stores a key shared by all leaf descendants of that node. The above two assumptions imply that every user possesses all the keys along the path from the leaf node to the root. Whenever a new user is added or deleted, the GC will update the keys along the path in a bottom-up fashion and notify the subset of nodes who share the keys. Wong et al. [7] pointed out that the average measured processing time increases linearly with the logarithm of the group size. Soneyink et al. [5] proved any distribution scheme has a worst-case cost of $\Omega(\log n)$ either for adding or for deleting a user. They also proved that the updating cost of a key tree with n insertions followed by n deletion is $\Theta(n \log n)$. Chen et al. [9] studied the structure of the optimal tree when a deletion sequence is performed. They show that the optimal tree is a tree where any internal node has at most degree 5 and children of nodes with degree not equal to 3 are all leaves. Based on this observation, they designed a dynamic programming algorithm of $O(n^2)$ time to compute an optimal tree. Another scenario where rekeying is done only periodically instead of immediately is studied as batch rekeying strategy in [4]. This strategy is further investigated in [8],[1] and [3] under the assumption that each user has a fixed probability p of being replaced during the batch period.

We further investigate the scenario proposed in [9]. Firstly, we find an important property of the optimal tree when only deletion cost is considered and use it to improve the time for computing the optimal tree from $O(n^2)$ to $O(\log \log n)$. Secondly, we study the optimal tree structure when insertion and deletion cost are simultaneously taken into consideration. Suppose in the initial setup period, the group only accepts membership joins. In the end of this period, a certain key tree is established by multicasting certain encryptions to the users. After that period, the group closes new membership and only accepts membership leaves. Notice that the GC update keys only at the end of the initial setup period and whenever a user leaves afterwards. This is different from the scenario of n insertions followed by n deletions considered in [5] where each insertion triggers an update on the key tree. We show that when considering both key tree establishment cost and deletion cost, the optimal tree is such a tree where any internal

node has at most degree 7 and children of nodes with degree not equal to 2 or 3 are all leaves.

The rest of this paper is organized as follows. In Section 2 we review the definition of the key tree model and introduce some related results. In Section 3, we prove an important property of the optimal tree for n deletions and reduce the computation time of the optimal tree from $O(n^2)$ to $O(\log \log n)$. In Section 4 we investigate a more general cost definition where the cost of the key tree establishment is also included. We prove the degree bound for the optimal tree in this scenario and propose an $O(n^2)$ dynamic programming algorithm. Finally, we conclude our work in Section 5. Due to space limit, most of the proofs are omitted in this version.

2 Preliminaries

In this section, we review the key tree model which is referred to in the literature either as key tree [7] or LKH (logical key hierarchy) [6].

In the key tree model, a group controller (GC) maintains the key structure for all the users. A group member holds a key if and only if the key is stored in an ancestor of the member. When a user leaves, GC will update any key that is known by him in a bottom-up fashion, and notify the remaining users who share that key. Take the structure shown in Figure 1 as an example, the user u_1 holds keys (k_1, k_6, k_8), and u_2 holds keys (k_2, k_6, k_8). When u_1 leaves, keys k_6 and k_8 need to be updated. GC will first encrypt the new k_6 with k_2 and multicast the message to the group. Notice that only u_2 can decrypt that message. Then GC encrypts the new k_8 with k_5, k_7 and with the new k_6 separately and multicast them to the group. Notice that all users except u_1 can obtain the new k_8 by decrypting one of those messages. Hence, the GC need 4 encryptions to maintain the key tree structure when u_1 leaves. Based on the updating rule, we introduce the definition of deletion cost.

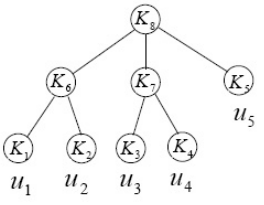

Fig. 1. An example key tree structure for a group with 5 members

Definition 1. *In a key tree T with n leaves, we say a node v has degree d_v if v has d_v children. We denote the set of ancestors of v as anc(v) (not including v itself) and define the ancestor weight of v as $w_v = \sum_{u \in anc(v)} d_u$. The number of leaf descendants of v is denoted as n_v.*

Given a leaf v_i in a key tree T, let $v_i u_1 u_2 \dots u_k$ be the longest path in T where u_j has only one child for $1 \leq j \leq k$. We define k as the *exclusive length* of v_i. Notice

that when the user v_i is deleted from the group, we need not update any key on the path $v_i u_1 u_2 \dots u_k$. Hence, we have the following deletion cost (defined in terms of the number of encryptions needed to update the keys after deletions). If not specified otherwise, we abbreviate w_{v_i} and n_{v_i} as w_i and n_i respectively.

Definition 2. *The deletion cost of v_i is $w_i - k - 1$ where k is the exclusive length of v_i, and we denote this deletion cost as c_i.*

Notice that when nodes are deleted, different deletion order may incur different deletion cost. In the tree shown in Figure 1, deletion sequence u_1, u_2, u_3, u_4, u_5 has cost $4+2+3+1+0 = 10$, while deletion sequence u_1, u_3, u_2, u_4, u_5 has cost $4+4+2+1+0 = 11$.

In our work, we further investigate the scenario where a group only accepts membership joins during the initial setup period. After that period, the only dynamic membership changes are deletions. This requires us to focus on the cost of a sequence instead of a single leaf. We first cite the following definitions from [9].

Definition 3. *In a key tree T with n leaf nodes, we define $\pi =< v_1, v_2, ..., v_n >$ as the sequence of all nodes to be deleted in T. Let $< c_1, c_2, ..., c_n >$ be the resulting sequence of deletion cost when the deletion sequence was performed. Let $C(T, \pi) = \sum_{i=1}^{n} c_i$ denote the deletion cost of the whole tree T under the deletion sequence π. The worst case deletion cost of the tree T is denoted as $C_{T,deletion} = \max_{\pi} C(T, \pi)$. We define the optimal tree $T_{n,opt}$ as a tree (not necessarily unique) which has the minimum worst case deletion cost over any tree T containing n leaf nodes.*

Definition 4. *Let T be a tree with n leaves. Given a tree T' with r leaves, we call T' a **skeleton** of T if T can be obtained by replacing r leaf nodes $v_1, v_2, \dots, v_r$ of T' with r trees $T_1, T_2, \dots, T_r$, where T_i has root v_i for $1 \le i \le r$.*

Under this definition, they proved a recursive formula for the worst case deletion cost $C_{T,deletion}$. Let T' be a skeleton of T as defined above. Given a deletion sequence π' for T' as well as a deletion sequence π_i for each T_i $(1 \le i \le r)$, they derive a deletion sequence π for T as follows. In the first step, π deletes all leaves in subtree T_i in the order specified by π_i, until there is only 1 leaf left. In the second step, π deletes the sole remaining leaf of each T_i in the order specified by π'. They denote the deletion sequence for T derived this way by $\pi =< \pi_1, \dots, \pi_r, \pi' >$.

Lemma 1. *[9] The sequence $\pi = \langle \pi_1, \pi_2, \dots, \pi_r, \pi' \rangle$ is a worst-case deletion sequence for T if π_i is a worst-case deletion sequence for T_i and π' is a worst-case deletion sequence for T'. The worst case deletion cost for T is*

$$C_{T,deletion} = C_{T',deletion} + \sum_{i=1}^{r} (C_{T_i,deletion} + (n_i - 1)w_i). \tag{1}$$

In this formula, $C_{T',deletion}$ is the worst-case deletion cost for the skeleton T', and $C_{T_i,deletion}$ is the worst-case deletion cost for the subtree T_i. The values n_i and w_i are the abbreviations of n_{v_i} and w_{v_i} respectively.

3 Semi-balance Property of Key Tree Structure for n Deletions

As [9] shows, to minimize the worst case deletion cost when all n subscribers are deleted from the group one by one, an optimal tree can be found among the trees satisfying the following two conditions: (1) every internal node has degree $d \leqslant 5$ and (2) children of nodes with degree $d \neq 3$ are all leaves. According to this, they gave an algorithm to compute the optimal tree in $O(n^2)$ time. In this section, we prove an important semi-balance property of the optimal tree.

According to the result of [9], if a node v has at least one child being an internal node, it must have degree 3. We further prove the following lemma.

Lemma 2. *There is an optimal tree where the children of any degree 3 node are either all leaves or all internal nodes.*

Proof. Suppose on the contrary in the optimal tree there exists an internal node v with degree 3, which has an internal node child v_1 and a leaf child v_0, then v_1 has at most two leaf descendants. Otherwise, if v_1 has $n_1 \geqslant 3$ leaf descendants, we show in the following that the cost of the tree can be decreased, which contradicts the optimality of the tree. Obviously, v_1 can only have degree $2 \leqslant d_1 \leqslant 5$. Firstly, when $d_1 \geqslant 3$, we can decrease the cost by moving a subtree T_2 (rooted at v_2) of v_1 and combine it with the leaf child (v_0) of v, as shown in Figure 2. Detailed proof is omitted here. On the other hand, when $d_1 = 2$, children of v_1 are all leaves because $d_1 \neq 3$, i.e. v_1 has at most two leaf descendants. Therefore, v has $n_v \leqslant 5$ leaf descendants, and in this case the optimal tree is such a tree where all v's children are leaves.

As a result, if a node has at least one leaf child, its children are all leaves. Furthermore, to make our explanation easier to read, we will give this kind of node a new name in the following.

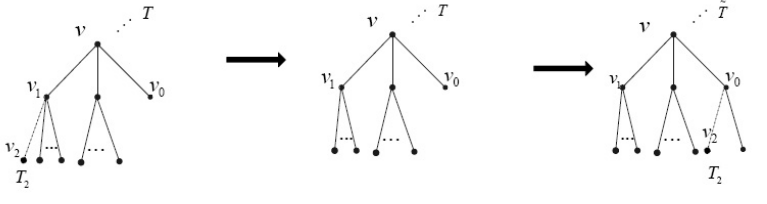

Fig. 2. A degree 3 node with at least one leaf child and one internal node child

Definition 5. *Given a tree T with L levels, we define the pseudo-leaf nodes to be the nodes whose children are all leaves. In addition, we use l_i to denote the level where the pseudo-leaf node u_i is placed and s_i to denote the number of its children.*

Obviously, all pseudo-leaf nodes can only be placed on level $1 \leqslant l_i \leqslant L-1$ and have $2 \leqslant s_i \leqslant 5$ children.

Lemma 3. *Given a tree T and a pseudo-leaf u_i in T, if we remove one child of u_i, the cost of the resulting tree $\bar{T}$ decreases by $3l_i + s_i - 1$.*

Proof. The lemma can be proved by choosing all the nodes in T except the children of u_i as the skeleton. The details are omitted here.

Lemma 4. *In an optimal tree $T_{n,opt}$, if a pseudo-leaf node u_i satisfies $1 \leqslant l_i \leqslant L-2$, then we have $s_i = 5$.*

Proof. Suppose $s_i < 5$ and another pseudo-leaf node u_j is on level $l_j = L - 1$. We can get a better tree by moving a child of u_j to be a child of the node u_i when $s_i \leqslant 4$, as shown by Figure 3. The detailed analysis is omitted here.

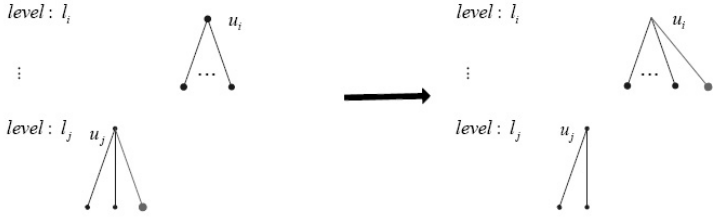

Fig. 3. Transformation of tree T where at least one pseudo-leaf node u_i with $s_i \leqslant 4$ children is on level l_i where $1 \leqslant l_i \leqslant (L-2)$

Lemma 5. *In an optimal tree $T_{n,opt}$, for any two pseudo-leaf nodes u_i and u_j satisfying $l_j > l_i$, we have $l_j - l_i \leq 1$, which means the pseudo-leaf nodes should only be on level $L-1$ or $L-2$.*

Proof. Suppose on the contrary there are two pseudo-leaf nodes u_i and u_j with $l_j - l_i \geqslant 2$ in the optimal tree. According to Lemma 4, we have $s_i = 5$. We can show that the cost decreases at least by 2 after moving a child of u_j to u_i as shown in Figure 4. Details are omitted here.

Lemma 6. *If all pseudo-leaf nodes are on the same level, we have the property that any node u_i, u_j on level $L-1$ satisfy the inequality $|n_i - n_j| \leqslant 1$ where n_i and n_j are the number of leaf descendants of u_i and u_j respectively.*

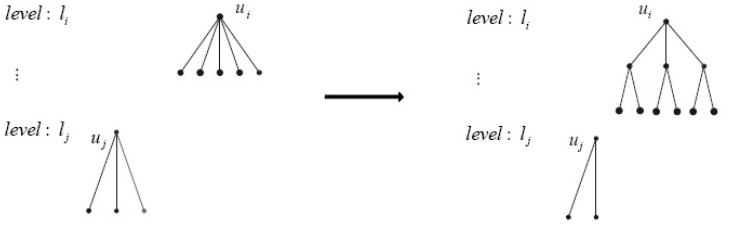

Fig. 4. Transformation of tree T where there are two pseudo-leaf nodes u_i and u_j who was respectively on level l_i and l_j, and $(l_j - l_i) \geqslant 2$

Proof. Because $n_i = s_i$ for all pseudo-leaf nodes, we only need to prove $|s_i - s_j| \leqslant 1$ for any pseudo-leaf u_i and u_j on the same level. In this case, according to Lemma 5, all of them can only be on level $L-1$. If $s_i - s_j \geqslant 2$, we can get a better tree by moving a child of u_i to u_j, because the cost decreased is a positive value, which is equal to $s_i - 1 - s_j \geqslant 1$. For the other case when $s_j - s_i \geqslant 2$, we can get a better tree in a similar way. Hence,the lemma is proved.

Lemma 7. *If some of the pseudo-leaf nodes are on different levels, then we have the property that for any node v_i, v_j on the same level ($L-2$ or $L-1$), the inequality $|n_i - n_j| \leqslant 1$ holds.*

Lemma 8. *Given a tree T, for any subtree T_i, T_j whose root v_i, v_j have the same ancestor weight $w_i = w_j$, if we exchange these two subtrees, the cost of the resulting tree does not change.*

Notice that when all pseudo-leaf nodes are on level $L-1$, the nodes v_i, v_j on level k ($1 \leqslant k \leqslant L-1$) have ancestor weight $w_i = w_j = 3k$. Therefore, the total deletion cost will not change when we exchange the subtrees with root nodes on the same level $L-1$. For the other case when some pseudo-leaf nodes are on level $L-2$ and others are on level $L-1$, the nodes v_i, v_j on level $1 \leqslant k \leqslant L-2$ also have ancestor weight $w_i = w_j = 3k$. According to this observation, for any two nodes v_i, v_j on the upper level, we prove inequality $|n_i - n_j| \leqslant 1$ also holds by exchanging the subtrees.

Lemma 9. *There is an optimal tree $T_{n,opt}$ where for any node v_i, v_j on the same level $1 \leqslant l_i = l_j \leqslant (L-2)$, the relation $|n_i - n_j| \leqslant 1$ holds.*

Based on these lemmas, we get the following theorem:

Theorem 1. *When the number of leaves $n \geqslant 6$, there is an optimal tree $T_{n,opt}$ where the sizes of its three subtrees differ by at most 1.*

The correctness of the theorem is evident with the support of Lemma 6, 7 and 9. Theorem 1 in fact implies that we can get an optimal tree by distributing the leaves into subtrees in a semi-balanced way until all the pseudo-leaf nodes u_i satisfy $2 \leqslant s_i \leqslant 5$. This can be interpreted as the following optimal tree constructing rule:

(1) When $n \leqslant 5$, optimal tree is a one level tree with all leaves on the same level.
(2) When $n \geqslant 6$, optimal tree is a tree with root degree 3. We distribute n leaves into its three subtrees in a semi-balanced way respectively with $\lceil \frac{n}{3} \rceil, \lceil \frac{n-1}{3} \rceil, \lceil \frac{n-2}{3} \rceil$ leaves.
(3) For each subtree, recursively construct its optimal structure according to (1) and (2).

The rule above implies that the deletion cost can be computed along with the constructing process. In fact, there are ($n \mod 3^k$) nodes which have $\lceil \frac{n}{3^k} \rceil$ leaf descendants and ($3^k - n \mod 3^k$) nodes which have $\lceil \frac{n}{3^k} \rceil - 1$ leaf descendants. Notice that when $2 \leqslant \lceil \frac{n}{3^k} \rceil \leqslant 5$ these nodes are pseudo-leaf nodes, while the optimal structure when $\lceil \frac{n}{3^k} \rceil = 6$ is the tree where root degree is 3 and each child has 2 leaves. Hence, we have

Theorem 2. *The worst case deletion cost of the optimal tree $T_{n,opt}$ can be computed in $O(\log\log n)$ time according to the equation below.*

$$T_{n,max} = (n \mod 3^k)\cdot T(\lceil \frac{n}{3^k} \rceil) + (3^k - n \mod 3^k)\cdot T(\lceil \frac{n}{3^k} \rceil - 1) + 3n\cdot k - 3^{k+1} + 3$$

$$where \quad k = \begin{cases} \lceil \log_3 n \rceil - 1 & if \ \ 2 \leqslant \lfloor \frac{n}{3^{\lceil \log_3 n \rceil - 1}} \rfloor \leqslant 5, \\ \lceil \log_3 n \rceil - 2 & if \ \ 2 \leqslant \lfloor \frac{n}{3^{\lceil \log_3 n \rceil - 2}} \rfloor \leqslant 5. \end{cases}$$

For basic cases $2 \leqslant n \leqslant 5$, we have $C(2) = 1, C(3) = 3, C(4) = 6, C(5) = 10$. Since it needs $O(\log\log n)$ time to compute the value of $3^{O(\log n)}$, the cost of the optimal tree can be computed in $O(\log\log n)$ time, which is much better than the dynamic programming algorithm with $O(n^2)$ time. Furthermore, to construct the optimal tree, we can distribute the users into subtrees in a semi-balanced way.

4 Optimal Tree Structures for n Insertions Followed by n Deletions

In this section, we investigate a more general setting where the cost of the initial group setup is also considered. In this new setting, the optimal tree we computed in the previous section is probably no longer optimal. We aim to study the optimal tree structure to minimize the cost for the initial setup followed by n deletions.

Lemma 10. *The number of encryptions needed to build the initial tree equals $N - 1$ where N is the number of the nodes in the tree.*

Proof. The tree is built after all the members arrive, we distribute the keys to the users securely in the bottom-up fashion with respect to the tree. Therefore, every key except the key stored in the root will be used once as an encryption key in the whole process, which amounts to $N - 1$ encryptions in total. The lemma is then proved.

To represent the total cost of insertion and deletion in one formula, we modify the cost of *skeleton* a bit as follows.

Definition 6. *Suppose the skeleton T' has t non-root nodes, the worst-case cost for T' is $C_{T',max} = C_{T',deletion} + t$.*

In fact, $C_{T',deletion}$ is the worst-case cost when only deletion is considered, and t is the number of messages encrypted by the keys stored in the t non-root nodes when establishing the initial key tree.

Lemma 11. *If the skeleton T' of T has r leaves, the worst-case cost for T is $C_{T,max} = C_{T',max} + \sum_{i=1}^{r}(C_{T_i,max} + (n_i - 1)w_i)$.*

Proof. By distributing the initial setup cost to every non-root node in the tree, it is easy to see that this lemma is implied by Lemma 1. We omit the detailed proof here.

We then prove the following structural property of the optimal tree which then enables us to find an optimal tree in $O(n^2)$ time. We omit our proof of this property in this version due to space limit.

Lemma 12. *There is an optimal tree $T_{n,opt}$ where every internal node v has degree at most 7 and children of nodes with degree not equal to 2 or 3 are all leaves.*

Based on this lemma, we have the following theorem:

Theorem 3. *Algorithm 1 can compute an optimal tree in $O(n^2)$ time.*

Proof. In Algorithm 1, we use R_i to denote the minimum cost of trees with i leaves and root degree restricted to be 3, D_i to denote the minimum cost of trees with i leaves and root degree restricted to be 2, and C_i to denote the minimum cost of all the trees with i leaves. The analysis is similar with the dynamic programming algorithm in [9] and omitted here.

Algorithm 1. *Sequence_OPT*

1. $R_1 = 1$; $R_2 = 3$; $R_3 = 6$; $R_4 = 10$; $R_5 = 15$;$R_6 = 21$; $R_7 = 28$;
2. $D_1 = 1$; $D_2 = 3$; $D_3 = 8$; $D_4 = 13$; $D_5 = 18$; $D_6 = 23$;$D_7 = 29$;
4. $C_1 = 1$; $C_2 = 3$; $C_3 = 6$; $C_4 = 10$; $C_5 = 15$;$C_6 = 21$; $C_7 = 28$;
5. for $i = 8$ to n
6. $\quad D_i = i^2$; $R_i = i^2$; $C_i = i^2$;
7. $\quad$ for $k1 = 1$ to $i/2$
8. $\quad\quad k2 = i - k1$;
9. $\quad\quad$ if $D_i > C_{k1} + C_{k2} + 2 \cdot i - 1$ then
10. $\quad\quad\quad D_i = C_{k1} + C_{k2} + 2 \cdot i - 1$;
11. $\quad\quad$ if $R_i > C_{k1} + D_{k2} + i + 2 \cdot k1 - 2$ then
12. $\quad\quad\quad R_i = C_{k1} + D_{k2} + i + 2 \cdot k1 - 2$;
13. $\quad$ end for
14. $\quad C_i = min(D_i, R_i)$;
15. end for

5 Conclusion

While many works focus on fixing the cost bound under some multicast protocol, we try to find the optimal structure to minimize the cost. We investigate the scenario where the members all arrive in the initial setup time and then leaves one by one. This can be applied in teleconferencing or applications where the member list can be fixed beforehand. Chen et al. [9] found the optimal tree structure when only deletion cost is considered. We prove an important property of the optimal key tree based on their work. We show that the members can be distributed in a semi-balance way in the optimal tree. Using this property we improve the running time from $O(n^2)$ to $O(\log \log n)$. We then focus on the optimal tree structure when insertion cost for the initial period is simultaneously considered. We obtain a recursive formula and use it to eliminate the impossible degrees in

the optimal tree. Based on this observation, we give an algorithm to compute the optimal tree with $O(n^2)$ time.

One possible direction of the future work is to investigate whether there is similar balanced structure for the optimal tree when insertion cost of the initial setup is considered together with the deletion cost.

References

1. Graham, R.L., Li, M., Yao, F.F.: Optimal Tree Structures for Group Key Management with Batch Updates. SIAM Journal on Discrete Mathematics 21(2), 532–547 (2007)
2. Goodrich, M.T., Sun, J.Z., Tamassia, R.: Efficient Tree-Based Revocation in Groups of Low-State Devices. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 511–527. Springer, Heidelberg (2004)
3. Li, M., Feng, Z., Graham, R.L., Yao, F.F.: Approximately Optimal Trees for Group Key Management with Batch Updates. In: Proceedings of the Fourth Annual Conference on Theory and Applications of Models of Computation, pp. 284–295 (2007)
4. Li, X.S., Yang, Y.R., Gouda, M.G., Lam, S.S.: Batch Re-keying for Secure Group Communications. In: Proceedings of the Tenth International Conference on World Wide Web, pp. 525–534 (2001)
5. Snoeyink, J., Suri, S., Varghese, G.: A Lower Bound for Multicast Key Distribution. In: Proceedings of the Twentieth Annual IEEE Conference on Computer Communications, pp. 422–431 (2001)
6. Wallner, D., Harder, E., Agee, R.C.: Key Management for Multicast: Issues and Architectures. RFC 2627 (1999)
7. Wong, C.K., Gouda, M.G., Lam, S.S.: Secure Group Communications Using Key Graphs. IEEE/ACM Transactions on Networking (8)(1), 16–30 (2000)
8. Zhu, F., Chan, A., Noubir, G.: Optimal Tree Structure for Key Management of Simultaneous Join/Leave in Secure Multicast. In: Proceedings of Military Communications Conference, pp. 773–778 (2003)
9. Chen, Z.-Z., Feng, Z., Li, M., Yao, F.F.: Optimizing Deletion Cost for Secure Multicast Key Management. Theoretical Computer Science (2008), doi:10.1016/j.tcs.2008.03.016