Optimal Key Tree Structure for Deleting Two or More Leaves*

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

[1] Department of Computer Science, University of Science and Technology of China
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 tree structure for the key management problem. In the key tree, when two or more leaves are deleted or replaced, the updating cost is defined to be the number of encryptions needed to securely update the remaining keys. Our objective is to find the optimal tree structure where the worst case updating cost is minimum. We first prove the degree upper bound $(k+1)^2 - 1$ when k leaves are deleted from the tree. Then we focus on the 2-deletion problem and prove that the optimal tree is a balanced tree with certain root degree $5 \leq d \leq 7$ where the number of leaves in the subtrees differs by at most one and each subtree is a 2-3 tree.

1 Introduction

In the applications that require content security, encryption technology is widely used. Asymmetric encryption is usually used in a system requiring stronger security, while symmetric encryption technology is also widely used because of the easy implementation and other advantages. In the applications such as teleconferencing and online TV, the most important security problem is to ensure that only the authorized users can enjoy the service. Centralized key management technology can achieve efficiency and satisfy the security requirement of the system. Hence, several models based on the key tree management are proposed to safely multicast the content. Two kinds of securities should be guaranteed in these applications: one is Future Security which prevents a deleted user from accessing the future content; the other is Past Security which prevents a newly joined user from accessing the past content. Key tree model, which was proposed by Wong el al. [8] is widely studied in recent years. In this model, the Group

* 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).

S.-H. Hong, H. Nagamochi, and T. Fukunaga (Eds.): ISAAC 2008, LNCS 5369, pp. 77–88, 2008.

Controller (GC) maintains a tree structure for the whole group. The root of the tree stores a Traffic Encryption Key (TEK) which is used to encrypt the content that should be broadcast to the authorized users. To update the TEK securely, some auxiliary keys are maintained. Whenever a user leaves or joins, the GC would update keys accordingly to satisfy Future Security and Past Security. Because updating keys for each user change is too frequent in some applications, [5] proposed Batch Rekeying Model where keys are only updated after a certain period of time. [10] studied the scenario of popular services with limited resources which always has the same number of joins and leaves during the batch period (because there are always users on the waiting list who will be assigned to empty positions whenever some authorized users leave). A recent survey for key tree management can be found in [2].

An important research problem in the key tree model is to find an optimal structure for a certain pattern of user behaviors so that the total number of encryptions involved in updating the keys is minimized. Graham et al. [3] studied the optimal structure in Batch Rekeying Model where every user has a probability p to be replaced in the batch period. They showed that the optimal tree for n users is an n-star when $p > 1 - 3^{-1/3} \approx 0.307$, and when $p \leq 1 - 3^{-1/3}$, the optimal tree can be computed in $O(n)$ time. Specially when p tends to 0, the optimal tree resembles a balanced ternary tree to varying degrees depending on certain number-theoretical properties of n. Feng et al. [1] studied the optimal structure in Key Tree Model under the assumption that users in the group are all deleted one by one. Their result shows that the optimal tree is a tree where every internal node has degree at most 5 and the children of nodes which have degree $d \neq 3$ are all leaves. [9] improved the result of [1] and showed that a balanced tree where every subtree has nearly the same number of leaves can achieve the optimal cost. They then investigate the optimal structure when the insertion cost in the initial setup period is also considered and showed that the optimal tree is a tree where every internal node has degree at most 7 and children of nodes which have degree $d \neq 2$ and $d \neq 3$ are all leaves.

More related to this paper, Soneyink et al. [6] proved that any distribution scheme has a worst-case cost of $\Omega(logn)$ for deleting a user. They also found an optimal structure when only one user is deleted from the tree. In this paper, we further investigate the problem when two or more users are deleted from a tree. We first prove a degree upper bound $(k+1)^2 - 1$ for the problem of deleting k users. Then we give a tighter degree bound for the problem of deleting two users. After that, we investigate the maximum number of leaves that can be placed on the tree given a fixed worst case deletion cost. Based on this, we prove that a balanced tree with certain root degree $5 \leq d \leq 7$ where the number of leaves in the subtrees differs by at most one and each subtree is a 2-3 tree can always achieve the minimum worst case 2-deletion cost.

The rest of this paper is organized as follows. We review the key tree model in Section 2 and then prove the degree bound of the optimal tree for the k-deletion problem in Section 3. From Section 4 on, we focus on the 2-deletion problem

and remove the possibility of root degree 2 and 3. In Section 5, we study the maximum number of leaves that can be placed on a tree given a fixed deletion cost and use the result to prove that the optimal tree for the 2-deletion problem is a tree where each subtree of the root is a 2-3 tree and has the number of leaves differed by at most 1. Finally, we conclude our work and propose a conjecture on the optimal tree cost for the general k-deletion problem in Section 6.

2 Preliminaries

We first review the Key Tree Model [8] which is also referred to in the literature as LKH(logical key hierarchy) [7].

In the Key Tree Model, there is a Group Controller maintaining a key tree for the group. A leaf on the key tree represents a user and stores an individual key that is only known by this user. An internal node stores a key that is shared by all its leaf descendants. Thus a user always knows the keys stored in the path from the leaf to the root. To guarantee content security, the GC encrypts the content by the Traffic Encryption Key (TEK) which is stored in the root and then broadcast it to the users. Only the authorized users knowing the TEK can decrypt the content. When a user joins or leaves, the GC will update the keys in a bottom-up fashion. As shown in Figure 1(a), there are 7 users in the group. We take the deletion of user u_4 as an example, since u_4 knows k_4, k_9 and k_{10}, the GC need to update the keys k_9 and k_{10} (the node that stores k_4 disappears because u_4 is already deleted from the group). GC will encrypt the new k_9 with k_5 and broadcast it to notify u_5. Note that only u_5 can decrypt the message. Then GC encrypts the new k_{10} with k_6, k_7, k_8 and the new k_9 respectively, and then broadcast the encrypted messages to notify the users. Since all the users and only the users in the group can decrypt one of these messages, the GC can safely notify the users except user u_4 about the new TEK. The deletion cost measured as the number of encryptions is 5 in this example.

In the following, we say that a node u has degree d if it has d children in the tree. Note that the worst case deletion cost of the tree shown in Figure 1(a) is 6 where one of the users u_1, u_2, u_3 is deleted. In [6], the authors investigate the optimal tree structure with n leaves where the worst case single deletion cost is minimum. Their result shows that the optimal tree is a special kind of 2-3 tree defined as follows.

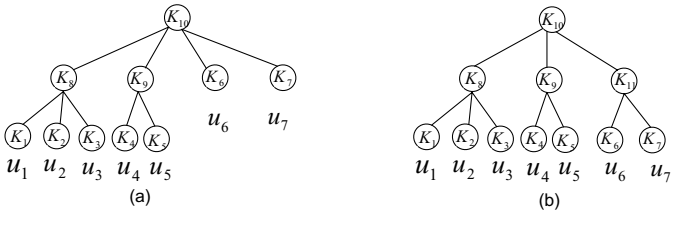

Fig. 1. Two structures of a group with 7 users

Definition 1. *In the whole paper, we use 2-3 tree to denote a tree with n leaves constructed in the following way.*

(1)When $n \geq 5$, the root degree is 3 and the number of leaves in three subtrees of the root differs by at most 1. When $n = 4$, the tree is a complete binary tree. When $n = 2$ or $n = 3$, the tree has root degree 2 and 3 respectively. When $n = 1$, the tree only consists of a root.

(2)Each subtree of the root is a 2-3 tree defined recursively.

In fact, [6] showed that 2-3 tree is a tree where the maximum ancestor weight (summation of degrees of all ancestors of a node) of the leaves on the tree is minimum among all the trees having the same number of leaves. As shown in Figure 1(b), the optimal tree for a single deletion for a group with 7 users has a worst case deletion cost 5.

In this paper, we study the scenario where two or more users leave the group during a period and we update keys at the end of this period. There are two versions of this problem to be considered here.

We denote the problem to find the optimal tree when k users are deleted as *pure k-deletion problem*. For example, deleting two users u_1 and u_4 in Figure 1(a) will incur cost 7 because we need to update k_8 and k_9 with 2 and 1 encryptions respectively and then to update k_{10} with 4 encryptions. This is also the worst case deletion. The objective is to find the optimal structure where the worst case cost is minimum.

In popular applications, there is a fixed number of positions and new users are always waiting to join. In such a scenario the number of joins and the number of leaves during the period are the same, which means that the newly joined k users will take the k positions which are vacant due to the leave of k users. In this setting, when two users u_1 and u_4 are replaced on Figure 1 (a), the updating cost is 9 which equals the summation of the ancestors' degrees of these two leaves where the common ancestors' degrees are only computed once. We denote the problem to find the optimal tree when k users are replaced as *k-deletion problem*.

We first define the k-deletion problem formally as follows.

Definition 2. *Given a tree T, we denote the number of encryptions incurred by replacing $u_{i_1}, \ldots, u_{i_k}$ with k new users as $C_T(u_{i_1}, \ldots, u_{i_k}) = \sum_{v \in (\bigcup_{1 \leq j \leq k} ANC(u_{i_j}))} d_v$ where $ANC(u)$ is the set of u's ancestor nodes and d_v is v's degree. We use k-deletion cost to denote the maximum cost among all possible combinations and write it as $C_k(T, n) = \max_{i_1, i_2, \ldots, i_k} C_T(u_{i_1}, u_{i_2}, \ldots, u_{i_k})$.*

We further define an optimal tree $T_{n,k,opt}$ (abbreviated as $T_{n,opt}$ if the context is clear) for k-deletion problem as a tree which has the minimum k-deletion cost over all trees with n leaves, i.e. $C_k(T_{n,opt}, n) = \min_T C_k(T, n)$. We also denote this optimal cost as $OPT_k(n)$. The k-deletion problem is to find the $OPT_k(n)$ and $T_{n,k,opt}$.

The pure k-deletion cost and the pure k-deletion problem are defined similarly by using the cost incurred by permanently deleting the leaves instead of the cost by updating the leaves (Some keys need not be updated if all its leaf descendants are deleted and the number of encryptions needed to update that key is also

reduced if some branches of that node totally disappear after deletion). We will show the relationship between these two problems in the following.

Definition 3. *We say a node v is a pseudo-leaf node if its children are all leaves. In the following two lemmas, we use t to denote the number of pseudo-leaf nodes in a tree T.*

Lemma 1. *If $t \leq k$, then the pure k-deletion cost of T is at least $n - k$.*

Proof. When $t \leq k$, we claim that in order to achieve pure k-deletion cost, we need to delete at least one leaf from each pseudo-leaf node. Suppose on the contrary there exists one pseudo-leaf node v where none of its children belongs to the k leaves we delete. We divide the discussion into two cases.

First, if each of the k leaves is a child of the remaining $t-1$ pseudo-leaf nodes, then there exists one pseudo-leaf node u with at least two children deleted. In this case, a larger pure deletion cost can be achieved if we delete one child of v while keeping one more child of u undeleted.

Second, if some of the k leaves are not from the remaining $t - 1$ pseudo-leaf nodes, then we assume u is one of them whose parent is not a pseudo-leaf. Then there exists one of u's sibling w that contains at least one pseudo-leaf w' (w' can be w itself). If no children of w' belong to the k leaves, then deleting a child of w' while keeping u undeleted incurs larger pure deletion cost. If at least one child of w' belongs to the k leaves, then deleting a child of v while keeping u undeleted incurs larger pure deletion cost.

We see that in the worse case deletion, each pseudo-leaf node has at least one child deleted, which implies that all the keys in the remaining $n - k$ leaves should be used once as the encryption key in the updating process. Hence the pure k-deletion cost of T is at least $n - k$.

Lemma 2. *If $t > k$, then the pure k-deletion cost is $C_k(T, n) - k$ where $C_k(T, n)$ is the k-deletion cost.*

Proof. Using similar arguments as in the proof of Lemma 1, we can prove that when $t > k$, the pure k-deletion cost can only be achieved when the k deleted leaves are from k different pseudo-leaf nodes. Then it is easy to see that the pure k-deletion cost is $C_k(T, n) - k$ where $C_k(T, n)$ is the k-deletion cost.

Theorem 1. *When considering trees with n leaves, the optimal pure k-deletion cost is $OPT_k(n) - k$ where $OPT_k(n)$ is the optimal k-deletion cost.*

Proof. Note that in the tree where all n leaves have the same parent (denoted as one-level tree), the pure k-deletion cost is $n - k$. By Lemma 1, any tree with the number of pseudo-leaf nodes at most k has the pure k-deletion cost at least $n - k$. Hence we only need to search the optimal tree among the one-level tree and the trees with the number of pseudo-leaf nodes larger than k. Moreover, in the one-level tree T, the pure k-deletion cost is $n - k = C_k(T, n) - k$ where $C_k(T, n)$ is the k-deletion cost. Further by Lemma 2, all the trees in the scope for searching the optimal tree have pure k-deletion cost $C_k(T, n) - k$, which implies

that the optimal pure k-deletion cost is $OPT_k(n) - k$ where $OPT_k(n)$ is the optimal k-deletion cost (The structure of the optimal trees in both problems are also the same).

The above theorem implies that the optimal tree for the pure k-deletion problem and the k-deletion problem are in fact the same. Therefore, we only focus on the k-deletion problem in the following and when we use "deleting", we in fact mean "updating".

3 Degree Bound for the k-Deletion Problem

In this section, we try to deduce the degree bound for the k-deletion problem. In the following proofs, we will often choose a template tree T and then construct a tree T' by deleting from T some leaves together with the exclusive part of leaf-root paths of those leaves. Here, the exclusive part of a leaf-root path includes those edges that are not on the leaf-root path of any of the remaining leaves. We also say that T is a template tree of T'. By the definition of the k-deletion cost, we have the following fact.

Fact 1. *If T is a template tree of T', then the k-deletion cost of T' is no larger than that of T.*

Lemma 3. *$OPT_k(n)$ is non-decreasing when n increases.*

Proof. Suppose on the contrary $OPT_k(n_1) > OPT_k(n_2)$ when $n_1 \leq n_2$, then there exist two trees T_1 and T_2 satisfying $C_k(T_1, n_1) = OPT_k(n_1)$ and $C_k(T_2, n_2) = OPT_k(n_2)$. We can take T_2 as a template tree and delete the leaves until the number of leaves decreases to n_1. The resulting tree T_1' satisfies $C_k(T_1', n_1) \leq C_k(T_2, n_2) < OPT_k(n_1)$ by Fact 1, which contradicts the definition of $OPT_k(n_1)$. The lemma is then proved.

Lemma 4. *$T_{n,opt}$ has root degree upper bounded by $(k+1)^2 - 1$.*

Proof. We divide the value of root degree $d \geq (k+1)^2$ into two sets, $\{d|(k+t)^2 \leq d < (k+t)(k+t+1), d, k, t \in N, t \geq 1\}$ and $\{d|(k+t-1)(k+t) \leq d < (k+t)^2, d, k, t \in N, t \geq 2\}$. Take $k = 2$ for instance, the first set is $\{9, 10, 11, 16, 17, 18, 19, 25, \ldots\}$ while the other is $\{12, 13, 14, 15, 20, 21, 22, 23, \ldots\}$.

Case 1: $(k+t)^2 \leq d < (k+t)(k+t+1)$ $(t \geq 1)$.
We write d as $(k+t)^2 + r$ where $0 \leq r < k+t$. Given a tree T, we can transform it into a tree with root degree $k+t$ as Figure 2 shows. In the resulting tree T', subtrees $T_{u_1}, \ldots, T_{u_{k+t}}$ are $k+t$ subtrees where the root $u_1, \ldots, u_{k+t}$ are on level one. Among the $k+t$ subtrees, there are r subtrees with root degree $k+t+1$ and $k+t-r$ subtrees with root degree $k+t$. Suppose that the k-deletion cost of T' is incurred by deleting $k_1, k_2, \ldots, k_s$ users from subtree $T_{i_1}, T_{i_2}, \ldots, T_{i_s}$ respectively where $k_1 + k_2 + \ldots + k_s = k$ and $s \leq k$. The corresponding cost is $C_k(T', n) = \sum_{j=1}^{s} C_{k_j}(T_{i_j}, n_{i_j}) + D_0$ where n_{i_j} is the number of leaves in T_{i_j} and D_0 is the cost incurred in the first two levels.

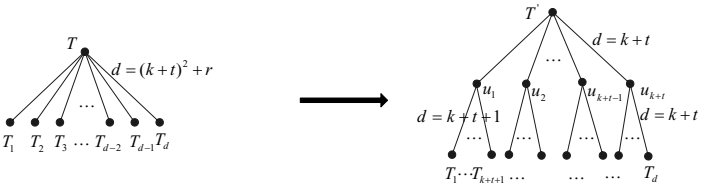

Fig. 2. Transformation of the tree which has root degree $d = (k+t)^2 + r$

In the original tree T, the corresponding cost for deleting those leaves is $C_k(T,n) = \sum_{j=1}^{s} C_{k_j}(T_{i_j}, n_{i_j}) + d = \sum_{j=1}^{s} C_{k_j}(T_{i_j}, n_{i_j}) + (k+t)^2 + r$. We will prove that when $t \geq 1$ we always have $C_k(T,n) \geq C_k(T',n)$, i.e. $D_0 \leq (k+t)^2 + r$.

Firstly, if $r \leq k$, the cost D_0 is at most $r(k+t+1) + (k-r)(k+t) + k + t$ where there are r users coming from r subtrees with root degree $k+t+1$ and $k-r$ users coming from $k-r$ subtrees with root degree $k+t$. Therefore, we have $D_0 \leq (k+t+1)r + (k+t)(k-r) + k + t = (k+t)(k+1) + r \leq (k+t)^2 + r$.

Secondly, if $r > k$, the cost D_0 is at most $(k+t) + (k+t+1)k$ where the k users are all from k subtrees which have root degree $k+t+1$. Therefore, we have $D_0 \leq (k+t) + (k+t+1)k \leq (k+t)(k+1) + k \leq (k+t)^2 + r$.

Hence, in both situations, the condition $t \geq 1$ ensures that the transformation from T to T' does not increase the k-deletion cost.

Case 2 can be proved similarly.

Lemma 4 suggests that we can find an optimal tree for k-deletion cost among trees whose root degree is at most $(k+t)^2 - 1$. Note that our degree bound in Lemma 4 is only for the root. We can also extend this property to all the internal nodes (proof is also omitted).

Lemma 5. *Any internal node in $T_{n,opt}$ has degree upper bounded by $(k+1)^2 - 1$.*

4 Degree Bound for 2-Deletion Problem

From this section on, we focus on the 2-deletion problem.

Definition 4. *We denote the maximum cost to delete a single leaf in a tree T as S_T and the maximum cost to delete two leaves as D_T, i.e. $S_T = C(T,1)$ and $D_T = C(T,2)$.*

According to Lemma 5, for 2-deletion, $T_{n,opt}$ has degree upper bounded by 8. Furthermore, in a tree T with root degree 1, the two deleted users in any combination are from the only subtree T_1. Therefore, the tree T_1 is better than T because the 2-deletion cost of T_1 is one less than that of T. Thus we need not consider root degree $d = 1$ when we are searching for the optimal tree.

Fact 2. *For 2-deletion problem, suppose that a tree T has root degree d where $d \geq 2$ and the d subtrees are $T_1, T_2, \ldots, T_d$. We have*

$$D_T = \max_{1 \leq i,j \leq d} \{D_{T_i} + d, S_{T_i} + S_{T_j} + d\}.$$

Proof. We know that deleting any two leaves from a subtree T_i will incur a cost at most $D_{T_i} + d$, while deleting two leaves from two different subtrees T_i and T_j will incur a cost at most $S_{T_i} + S_{T_j} + d$. The 2-deletion cost comes from one of the above cases and therefore the fact holds.

We can further remove the possibility of degree 8 by the following lemma.

Lemma 6. *For 2-deletion problem, we can find an optimal tree among the trees with node degrees bounded between 2 and 7.*

In the following, we show two important properties of the optimal tree (monotone property and 2-3 tree property) and then further remove the possibility of root degree 2 and 3 to reduce the scope of trees within which we search for the optimal tree. Due to space limit, most of the proofs are omitted in this version.

Lemma 7. *(monotone property) For 2-deletion problem, suppose a tree T has root degree d where $d \geq 2$ and d subtrees are $T_1, T_2, \ldots, T_d$. Without loss of generality, we assume that T has a non-increasing leaf descendant vector $(n_1, n_2, \ldots, n_d)$, where n_i is the number of leaves in subtree T_i. Then, there exists an optimal tree where $S_{T_1} \geq S_{T_2} \geq \ldots \geq S_{T_d}$ and T_i is a template of T_{i+1} for $2 \leq i \leq d-1$.*

Fact 3. *For trees satisfying Lemma 7, we have*

$$D_T = \max\{D_{T_1} + d, S_{T_1} + S_{T_2} + d\}.$$

Proof. By Fact 2 we have $D_T = \max_{1 \leq i,j \leq d}\{D_{T_i} + d, S_{T_i} + S_{T_j} + d\}$. Lemma 7 further ensures that $\max_{1 \leq i,j \leq d}\{D_{T_i} + d, S_{T_i} + S_{T_j} + d\} = \max\{D_{T_1} + d, D_{T_2} + d, S_{T_1} + S_{T_2} + d\}$. Since $D_{T_2} < 2S_{T_2} \leq S_{T_1} + S_{T_2}$, we have $D_T = \max\{D_{T_1} + d, S_{T_1} + S_{T_2} + d\}$.

In the following, we further reduce the scope for searching the optimal tree by proving the following lemma.

Lemma 8. *(2-3 tree property) For a tree T satisfying Lemma 7, we can transform subtrees $T_2, \ldots, T_d$ into 2-3 trees without increasing the 2-deletion cost.*

Proof. Given a tree T satisfying Lemma 7 and Fact 3, we transform subtrees $T_2, T_3, \ldots, T_d$ into 2-3 trees $T'_2, T'_3, \ldots, T'_d$ to get a new tree T'. For $2 \leq i \leq d$, since $S_{T'_i} = OPT_1(n_i)$, we have $S_{T'_d} \leq \ldots \leq S_{T'_3} \leq S_{T'_2} \leq S_{T_2}$ (Lemma 3) and $D_{T'_i} \leq 2S_{T'_i} \leq 2S_{T'_2} \leq S_{T_1} + S_{T'_2}$ $(2 \leq i \leq d)$. Thus $D_{T'} = \max\{D_{T_1} + d, D_{T'_2} + d, S_{T_1} + S_{T'_2} + d\} \leq \max\{D_{T_1} + d, S_{T_1} + S_{T_2} + d\} = D_T$, which implies the transformation does not increase 2-deletion cost. The lemma is then proved.

We denote the trees satisfying Lemma 8 as *candidate-trees*. By Lemma 8, we can find an optimal tree among all the candidate trees. For a candidate tree T with root degree d, we define branch B_i to be the union of T_i and the edge connecting the root of T with the root of T_i. We say the branch B_1 is the *dominating branch* and other branches $B_2, \ldots B_d$ are *ordinary branches*. We then prove the following theorem to further remove the possibility of root degree 2 and 3 in the optimal tree (details are omitted in this version).

Theorem 2. *For 2-deletion problem, a tree T with root degree 2 or 3 can be transformed into a tree with root degree 4 without increasing the 2-deletion cost.*

5 Optimal Structure of 2-Deletion Problem

Although we have removed the possibility of the root degree 2 and 3 in Section 4 and have fixed the structure of the ordinary branches, we still do not have an effective algorithm to exactly compute the optimal structure because we need to enumerate all the possible structures of the dominating branch. In this section, we will prove that among the candidate trees with n leaves, a balanced structure can achieve 2-deletion cost $OPT_2(n)$. The basic idea is to first investigate the capacity $g(R)$ for candidate trees with 2-deletion cost R (Theorem 3). Note that the optimal tree has the minimum 2-deletion cost with n leaves, which reversely implies that if we want to find a tree with 2-deletion cost R and at the same time has the maximum possible number of leaves, then computing the optimal tree for increasing n until $OPT_2(n) > R$ will produce one such solution. We then analyze and calculate the exact value for the capacity (maximum number of leaves) given a fixed 2-deletion cost R (Theorem 4). Finally, we prove that certain balanced structure can always be the optimal structure that minimizes the 2-deletion cost (Theorem 5).

Definition 5. *We use capacity to denote the maximum number of leaves that can be placed in a certain type of trees given a fixed deletion cost. According to [6], function $f(r)$ defined below is the capacity for 1-deletion cost r (among all the possible trees). We use function $g(R)$ to denote the capacity for 2-deletion cost R (among all the possible trees). In other words, when $g(R-1) < n \leq g(R)$, we have $OPT_2(n) = R$.*

$$f(r) = \begin{cases} 3 \cdot 3^{i-1} & \text{if} \quad r = 3i \\ 4 \cdot 3^{i-1} & \text{if} \quad r = 3i+1 \\ 6 \cdot 3^{i-1} & \text{if} \quad r = 3i+2 \end{cases}$$

To facilitate the discussion, according to Fact 3, we can divide the candidate trees with 2-deletion cost R and root degree d into two categories as summarized in the following definition.

Definition 6. *Candidate trees of category 1: The two leaves whose deletion cost achieves 2-deletion cost are from different branches, i.e. $D_T = S_{T_1} + S_{T_2} + d$, which implies $S_{T_1} + S_{T_2} \geq D_{T_1}$.*

Candidate trees of category 2: The two leaves whose deletion cost achieves 2-deletion cost are both from the dominating branch B_1, i.e. $D_T = D_{T_1} + d$, which implies $S_{T_1} + S_{T_2} < D_{T_1}$.

Correspondingly, we denote the capacity of the candidate trees belonging to category 1 with 2-deletion cost R as $g_1(R)$ and denote the capacity of the candidate trees belonging to category 2 with 2-deletion cost R as $g_2(R)$. Note that we can find the optimal tree among the candidate trees according to Lemma 8, which implies that with the same 2-deletion cost R, the best candidate tree can always have equal or larger number of leaves than the general trees. That is, we have $g(R) = max\{g_1(R), g_2(R)\}$. Thus in the following discussions, we only focus on

the candidate trees. On the other hand, because we are finding trees with the maximum number of leaves, it is easy to see that we can assume the number of leaves in ordinary branches are all the same (Otherwise, we can make the tree bigger without affecting the 2-deletion cost).

In all candidate trees with 2-deletion cost R, by Fact 3, we only need to consider the case where at most one of the two leaves whose deletion cost achieves 2-deletion cost are from the ordinary branches. Suppose each ordinary branch has 1-deletion cost r^-, and correspondingly T_1 has 1-deletion cost r^+ where $r^+ \leq R - d - r^-$ (otherwise we have $D_T \geq r^+ + r^- + d > R$, a contradiction). For fixed cost R, Lemma 7 (monotonous property) implies that $r^+ \geq r^-$. We first prove the following capacity bound (details are omitted in this version).

Theorem 3. *We have* $g_i(R) \leq (R - 2r^-) \cdot f(r^-)(i = 1, 2)$.

In the following theorem, among these candidate trees we study the optimal structure which achieves the maximum capacity for different values of R.

Theorem 4. *For 2-deletion cost R, the maximum capacity is*

$$g(R) = \begin{cases} 6 \cdot 3^{i-1} & if \quad R = 6i \\ 7 \cdot 3^{i-1} & if \quad R = 6i+1 \\ 8 \cdot 3^{i-1} & if \quad R = 6i+2 \\ 10 \cdot 3^{i-1} & if \quad R = 6i+3 \\ 12 \cdot 3^{i-1} & if \quad R = 6i+4 \\ 15 \cdot 3^{i-1} & if \quad R = 6i+5 \end{cases}$$

After we have obtained the capacity for the 2-deletion cost R, we finally prove that among the candidate trees with n leaves, the optimal cost can be achieved by some balanced structure as shown below.

Definition 7. *We use the balanced tree to denote a tree with root degree d where each subtree is 2-3 tree and has number of leaves differed by at most 1.*

Theorem 5. *Among trees with n leaves,*

(1)when $n \in (15 \cdot 3^{i-1}, 18 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree 6 and 2-deletion cost 6i+6.*
(2)when $n \in (12 \cdot 3^{i-1}, 15 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree 5 and 2-deletion cost 6i+5.*
(3)when $n \in (10 \cdot 3^{i-1}, 12 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree 6 and 2-deletion cost 6i+4.*
(4)when $n \in (8 \cdot 3^{i-1}, 10 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree 5 and 2-deletion cost 6i+3.*
(5)when $n \in (7 \cdot 3^{i-1}, 8 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree 6 and 2-deletion cost 6i+2.*
(6)when $n \in (6 \cdot 3^{i-1}, 7 \cdot 3^{i-1}]$, *the optimal tree is a balanced tree which has root degree 7 and 2-deletion cost 6i+1.*

Proof. When $n \in (15 \cdot 3^{i-1}, 18 \cdot 3^{i-1}]$, we have $OPT_2(n) = 6i+6$. We will prove that the balanced tree with root degree 6 can always achieve this optimal cost. In the balanced tree, each subtree T_j $(1 \leq j \leq 6)$ has the number of leaves $n_j = \lceil \frac{n-j+1}{6} \rceil \in [\lfloor \frac{5}{2} \cdot 3^{i-1} \rfloor, 3 \cdot 3^{i-1}]$. By function $f(\cdot)$, we have $S_{T_i} \leq 3i$. Thus any two leaves from the tree will incur a deletion cost at most $2 \cdot 3i + 6 = 6i + 6$.

When $n \in (12 \cdot 3^{i-1}, 15 \cdot 3^{i-1}]$ we have $OPT_2(n) = 6i+5$. Then we will prove that the balanced tree with root degree 5 can always achieve the optimal cost. In the balanced tree, each subtree T_j $(1 \leq j \leq 5)$ has the number of leaves $n_j = \lceil \frac{n-j+1}{5} \rceil \in [\lfloor \frac{12}{5} \cdot 3^{i-1} \rfloor, 3 \cdot 3^{i-1}]$. By function $f(\cdot)$, we have $S_{T_i} \leq 3i$. Thus any two leaves from the tree will incur a deletion cost at most $2 \cdot 3i + 5 = 6i + 5$.

When $n \in (10 \cdot 3^{i-1}, 12 \cdot 3^{i-1}]$ we have $OPT_2(n) = 6i+4$ and $n_j = \lceil \frac{n-j+1}{6} \rceil \in [\lfloor \frac{5}{3} \cdot 3^{i-1} \rfloor, 2 \cdot 3^{i-1}]$. By function $f(\cdot)$, we have $S_{T_i} \leq 3i-1$. Thus any two leaves from the tree will incur a deletion cost at most $2 \cdot (3i-1) + 6 = 6i + 4$.

When $n \in (8 \cdot 3^{i-1}, 10 \cdot 3^{i-1}]$, we have $OPT_2(n) = 6i+3$ and $n_j = \lceil \frac{n-j+1}{5} \rceil \in [\lfloor \frac{8}{5} \cdot 3^{i-1} \rfloor, 2 \cdot 3^{i-1}]$. By function $f(\cdot)$, we have $S_{T_i} \leq 3i-1$. Thus any two leaves from the tree will incur a deletion cost at most $2 \cdot (3i-1) + 5 = 6i + 3$.

When $n \in (7 \cdot 3^{i-1}, 8 \cdot 3^{i-1}]$, we have $OPT_2(n) = 6i+2$ and $n_j = \lceil \frac{n-j+1}{6} \rceil \in [\lfloor \frac{7}{6} \cdot 3^{i-1} \rfloor, \frac{4}{3} \cdot 3^{i-1}]$. By function $f(\cdot)$, we have $S_{T_i} \leq 3i-2$. Thus any two leaves from the tree will incur a deletion cost at most $2 \cdot (3i-2) + 6 = 6i + 2$.

When $n \in (6 \cdot 3^{i-1}, 7 \cdot 3^{i-1}]$, we have $OPT_2(n) = 6i+1$. The balanced tree with root degree 7 where $n_j = \lceil \frac{n-j+1}{7} \rceil \in [\lfloor \frac{6}{7} \cdot 3^{i-1} \rfloor, 3^{i-1}]$ can always achieve the optimal cost. By function $f(\cdot)$, we have $S_{T_i} \leq 3i-3$. Thus any two leaves from the tree will incur a deletion cost at most $2 \cdot (3i-3) + 7 = 6i + 1$.

Note that in some cases a balanced tree with degree 4 can also be an optimal tree, but it is not necessary to consider this possibility because we do not need to find all the possible structures of an optimal tree with n leaves.

Finally we have fixed the structure of the dominating branch and obtained the optimal tree structure for the 2-deletion problem. We conjecture the general result for the k-deletion problem in the next section.

6 Conclusion

In this paper, we study the optimal structure for the key tree problem. We consider the scenario where two or more users are deleted from the key tree and aim to find an optimal tree in this situation. We first prove a degree upper bound $(k+1)^2 - 1$ for the k-deletion problem. Then we focus on the 2-deletion problem by firstly removing the possibility of root degree 2 and 3 to reduce the scope for searching the optimal tree. Then, we investigate the capacity of the key tree. Based on this, we prove that the optimal tree for the 2-deletion problem is a balanced tree with certain root degree $5 \leq d \leq 7$ where the number of leaves in each subtree differs by at most 1 and each subtree is a 2-3 tree.

The capacity $f(\cdot)$ where $k = 1$ and $g(\cdot)$ where $k = 2$ stimulates us to conjecture the general form of capacity $G_k(R)$ which denotes the maximum number of leaves that can be placed in a tree given the k-deletion cost R in the k-deletion problem. Based on the form of $f(\cdot)$ and $g(\cdot)$, we conjecture the capacity $G_k(R)$ to be of

the form shown in Equation (1). Furthermore, if the conjecture is proved to be correct, it is also possible to obtain the optimal structure in a similar way as in the proof of Theorem 5.

$$G_k(R) = \begin{cases} (3k+\alpha)\cdot 3^{i-1} & \text{if} \quad R = 3k\cdot i + \alpha, \alpha \in [0,k) \\ (4k+2(\alpha-k))\cdot 3^{i-1} & \text{if} \quad R = 3k\cdot i + \alpha, \alpha \in [k,2k) \\ (6k+3(\alpha-2k))\cdot 3^{i-1} & \text{if} \quad R = 3k\cdot i + \alpha, \alpha \in [2k,3k) \end{cases} \quad (1)$$

One of the possible future work is therefore to investigate the capacity and optimal structure for the general k-deletion problem. We believe that the concept of capacity will also be very important to this problem.

References

1. Chen, Z.Z., Feng, Z., Li, M., Yao, F.F.: Optimizing Deletion Cost for Secure Multicast Key Management. Theoretical Computer Science 401, 52–61 (2008)
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. 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)
4. Li, M., Feng, Z., Graham, R.L., Yao, F.F.: Approximately Optimal Trees for Group Key Management with Batch Updates. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 284–295. Springer, Heidelberg (2007)
5. 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)
6. 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)
7. Wallner, D., Harder, E., Agee, R.C.: Key Management for Multicast: Issues and Architectures, RFC 2627 (June 1999)
8. Wong, C.K., Gouda, M.G., Lam, S.S.: Secure Group Communications Using Key Graphs. IEEE/ACM Transactions on Networking 8(1), 16–30 (2003)
9. Wu, W., Li, M., Chen, E.: Optimal Tree Structures for Group Key Tree Management Considering Insertion and Deletion Cost. In: Proceedings of the 14th Annual International Computing and Combinatorics Conference, pp. 521–530 (2008)
10. 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)