Optimizing Deletion Cost for Secure Multicast Key Management*

Ze Feng, Minming Li and Frances Yao

Department of Computer Science
City University of Hong Kong
{fengze,minmli}@cs.cityu.edu.hk, csfyao@cityu.edu.hk

Abstract. Multicast and broadcast are efficient ways to deliver messages to a group of recipients in a network. Due to the growing security concerns in various applications, messages are often encrypted with a secret group key. The key tree model which has been widely adopted maintains a set of keys in a tree structure so that in case of group member change, the group key can be updated in a secure and efficient way. In this paper, we focus on the updating cost incurred by member deletions. To implement a sequence of member deletions in any key tree, a certain number of encrypted messages need to be broadcast to accomplish the updates. Our goal is to identify the best key tree which can minimize the worst case deletion cost (i.e., the amortized cost over n member deletions). We prove that there is an optimal tree where all internal nodes have degree at most 5. Furthermore, any internal node with degree not equal to 3 must be such that all of its children are leaves. Based on these characterizations, we present a dynamic programming algorithm that computes an optimal key tree in $O(n^2)$ time.

1 Introduction

With the advances in network technologies, many interesting applications based on group communications are emerging. Security is often an important concern in these applications due to either privacy (e.g. teleconferencing) or profit (e.g. pay-per-view) reasons. There are two ways to achieve security in these systems: one is to use sophisticated cryptographic techniques, while the other is to use simple symmetric encryption where the same key is used for both encrypting and decrypting messages. In many of the applications, especially those implemented on mobile devices or requiring immediate response like teleconferencing, the symmetric encryption is preferred because it is more efficient. To achieve security while guaranteeing efficiency, the system needs to manage the keys in an appropriate way so that it is still safe when there are member changes. We can summarize the above requirement into the following group broadcast problem, where we have n subscribers and a group controller (GC) that periodically broadcasts messages (e.g., a video clip) to all subscribers over an insecure channel. To guarantee that only the authorized users can decode the contents of the messages, the GC will dynamically maintain a key structure for the whole group. Whenever a user leaves or joins, the GC will generate some new keys as necessary and notify the remaining users of the group in

* 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 Hong Kong SAR, China (Project No. CityU 122105) and a grant from City University of Hong Kong (Project No. 7200072).

some secure way. Surveys on the key management for secure group communications can be found in [7][1].

The key tree model [10] is widely used for the key management problem. We describe this model briefly as follows. Every leaf node of the key tree represents a user and stores his individual key. Every internal node stores a key shared by all leaf descendants of that node. Every user possesses all the keys along the path from the leaf node (representing the user) to the root. To prevent revoked users from knowing future message contents and also to prevent new users from knowing past message contents, the GC updates a subset of keys, whenever a new user joins or a current user leaves, as follows. As long as there is a user change among the leaf descendants of an internal node v, the GC will 1) replace the old key stored at v with a new key, and 2) broadcast the new key after encrypting it with the key stored at each child node of v. Note that only users corresponding to the leaf descendants of v can decipher useful information from the broadcast. Furthermore, this procedure must be done in a bottom-up fashion (i.e., starting with the lowest v whose key must be updated) to guarantee that a revoked user will not know the new keys. The cost of the above procedure counts the number of encryptions used in step 2) above.

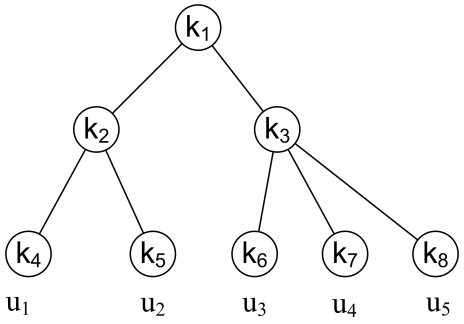

Fig. 1. An example key tree with 5 users.

An example key tree for a group with 5 members is shown in Figure 1. A group member holds a key if and only if the key is stored in an ancestor of the member. For example, u_1 holds keys (k_1, k_2, k_4), and u_2 holds keys (k_1, k_2, k_5). When u_2 leaves, we need to update k_2 and k_1. GC will first encrypt the new k_2 with k_4 and multicast the message to the group. This message can only be decrypted by u_1. Then GC encrypts the new k_1 with k_3 and with the new k_2 separately and multicast them to the group. All users except u_2 can decrypt one of these two messages to obtain the new k_1. Therefore, after the deletion of u_2, the GC will use 3 encryptions to maintain the key tree.

There has been a lot of work on managing the key trees or finding the optimal key trees according to different behavior of the dynamic membership change of the group. Snoeyink et al.[8] proved that the updating cost of a key tree with n insertions followed by n deletions is $\Theta(n \log n)$. They also considered the case of a single deletion and showed that a special 2-3 tree can achieve the minimum deletion cost. Later, Goshi and Ladner

[2] designed some scalable on-line algorithms for maintaining balanced key trees in face of arbitrary dynamic membership changes. Hao et al. [4] investigated the key tree problem when the user departure time is predictable. They applied a new scheme based on AVL trees to reduce the communication cost.

There is an alternative strategy for key management whereby rekeying is done only periodically instead of immediately after each membership change [6]. This batch rekeying model is further investigated in [11][3][5] under the assumption that during the batch period every user has a probability p of being replaced by another user.

In this paper, we further investigate the scenario proposed in [8] by focusing on the deletion cost in key trees. Suppose a group only accepts membership joins during the initial setup period. After that period, it is closed to new membership and the only dynamic membership changes are deletions. This is an interesting special case of key tree maintenance that can be applied for example to teleconferencing where the group members are usually set up at the beginning of the conference and subsequently members may leave at different points in time. This may occur in other similar situations when the group membership is quite selective and the complete list is drawn up beforehand. By using the updating rule described above, different deletion sequence (user leave order) will incur different deletion cost for updating the keys. Given a specific key tree, we consider its amortized deletion cost (i.e., the average cost per deletion over the worst sequence of n deletions). We are interested in identifying the optimal tree which can achieve the minimum amortized deletion cost among all trees. We will prove that there exists an optimal tree where all internal nodes have degree at most 5, and only nodes of degree 3 can have non-trivial subtrees. Based on these characterization, we present a dynamic programming algorithm which can compute an optimal tree in $O(n^2)$ time. Notice that without these characterizations, a brute-force exhaustive search for an optimal tree would take exponential time.

The remainder of this paper is organized as follows. In Section 2, we define the model used in our work. We characterize the worst case deletion sequence in Section 3 and derive some degree bounds for the optimal tree in Section 4. Finally, we summarize our work and give some open problems in Section 5.

2 Models and Preliminaries

We first review the basic key tree model for group key management. This model is referred to in the literature either as key tree [10] or LKH (logical key hierarchy) [9].

In the key tree model, there are a Group Controller (GC), represented by the root, and n subscribers (or users) represented by the n leaves of the tree. The tree structure is used by the GC for key management purposes. Associated with every node of the tree (whether internal node or leaf) is an encryption key. The key associated with the root is called the Traffic Encryption Key (TEK), which is used by the subscribers for accessing encrypted service contents. The key k_v associated with each non-root node v is called a Key Encryption Key (KEK) which is used for updating the TEK when necessary. Each subscriber possesses all the keys along the path from the leaf representing the subscriber to the root. When a user leaves, any key that is known both by him and some other

users needs to be updated. When a user joins, any key that is going to be known by him needs to be updated. For example, in Figure 1, if u_4 leaves, then k_1 and k_3 need to be updated; if u_6 joins and connects directly to k_2, then k_1 and k_2 need to be updated. The way to update a key k_i is to encrypt the new key k_i' separately using the keys stored in the children of the node storing k_i. Therefore, the number of encryptions needed to update a key equals the number of children it has. After defining some notations in trees, we will give a formal definition of insertion cost and deletion cost.

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 (not including v itself) by $anc(v)$ and define the ancestor weight of v as $w_v = \sum_{u \in anc(v)} d_u$. The number of leaf descendants of node 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 tree 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 $v_i u_1 u_2 \dots u_k$. Therefore, we have the following insertion cost and deletion cost (number of encryptions needed to update the keys after insertions or deletions). If not specified otherwise, we abbreviate w_{v_i} and n_{v_i} as w_i and n_i respectively.

Definition 2. *The insertion cost of a leaf node v_i is w_i. The deletion cost of a leaf node v_i is $w_i - k - 1$ where k is the exclusive length of v_i. We denote the deletion cost of v_i by c_i.*

Notice that the total deletion cost of all the nodes in T depends on the sequence of the nodes to be deleted. In the tree shown in Figure 2, the deletion cost of $< v_1, v_2, v_3 >$ is $1 + 2 + 0 = 3$, while the deletion cost of $< v_2, v_1, v_3 >$ is $3 + 1 + 0 = 4$.

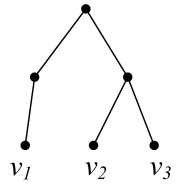

Fig. 2. An example where different deletion sequences incur different deletion cost.

In this paper, we consider a scenario where a group only accepts membership joins during the initial setup period. After that period, the only dynamic membership changes are deletions. Our objective is to find a best tree which minimizes the worst case deletion cost. Notice that there is also a tree construction cost associated with the initial setup which can be shown easily in the next lemma.

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

Proof. Because the tree is built after all the members arrive, we can 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. The lemma then follows.

Notice that one may also consider the special case where only insertions happen in the dynamic membership change. In that problem, minimizing the total insertion cost is equivalent to finding a best tree which minimizes the best case deletion cost. The best case deletion sequence is exactly the reversed order of best case insertion sequence. It is an interesting problem in its own right, but will not be considered in this paper.

3 Worst Case Deletion Sequence for a Given Tree

In this section, we characterize the worst case deletion sequence. We accomplish this by utilizing the recursive structures of trees.

Definition 3. *Given a tree T with n leaf nodes, we define $\pi =< v_1, v_2, \ldots, v_n >$ as the sequence of all leaf nodes to be deleted in T. Let $< c_1, c_2 \ldots c_n >$ be the resulting sequence of deletion cost to delete each leaf node. 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 a tree T is denoted as $C_{T,max} = \max_{\pi} C(T, \pi)$.*

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

Suppose the worst case deletion sequence for T' is $\pi' =< v_1, v_2, \ldots, v_r >$. For each T_i, which has n_i nodes for $1 \leq i \leq r$, suppose π_i is a deletion sequence for T_i. For each $\pi_i =< u_{i,1}, u_{i,2}, \ldots, u_{i,n_i} >$ $(1 \leq i \leq r)$, let $\pi_i^* =< u_{i,1}, u_{i,2}, \ldots, u_{i,n_i-1} >$. Notice that if T_i is a single path with some leaf u, then the updating cost for deleting u equals the updating cost for deleting v_i in T' because we need not update any of the keys in T_i when deleting u. Therefore, deleting a leaf node v_i in T' corresponds to deleting the last leaf in T_i. In this way, given $\pi_1, \pi_2, \ldots, \pi_r$ and π', we have a corresponding deletion sequence $\pi =< \pi_1^*, \pi_2^*, \ldots, \pi_r^*, \pi' >$ for T.

Lemma 2. *The sequence $\pi =< \pi_1^*, \pi_2^*, \ldots, \pi_r^*, \pi' >$ is a worst case deletion sequence for T if π_i is a worst case deletion sequence for T_i and π' is worst for T'. The worst case deletion cost for T is*

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

Proof. We adopt two steps in deleting leaf nodes in T according to π: in the first step we delete all the subtrees T_i under the skeleton T' with each of the subtree having only one leaf left (the maximum cost is $\sum_{i=1}^{r} (C_{T_i,max} + (n_i - 1)w_i)$. In the second step, we

delete the remaining leaves which is equivalent to deleting the leaf nodes in the skeleton T' (the maximum cost is $C_{T',max}$).

To prove the lemma, we interpret the deletion cost in the following way. Whenever we delete a node v, we attribute an additional cost 1 to a node u if a new key needs to be encrypted by the key stored in u. Therefore, when we delete a leaf in T_i, we will attribute cost 1 to the root v_j of every other T_j ($j \neq i$) if T_j is not deleted to empty, but will never attribute any cost to non-root nodes in T_j. On the other hand, we know that the cost attributed to the non-root nodes in T_i is only decided by the relative deletion sequence for leaves in T_i.

We prove the lemma by dividing the cost into three parts. The cost incurred by deleting the last leaf in T_i ($1 \leq i \leq r$), the cost attributed to the non-root nodes in T_i ($1 \leq i \leq r$) and the cost attributed to nodes in T' by the deletion of non-last leaves in T_i ($1 \leq i \leq r$). Firstly, the cost incurred by deleting the last leaf in T_i is only attributed to the nodes in T' and therefore is only decided by T'. The sequence π achieves the maximum cost of this part by deleting those last leaves in the worst case π'. Next, it is easy to see that the sequence π attributes the maximum cost to non-root nodes of each T_i because π_i^* is a worst case deletion sequence for T_i (Notice that the last leaf deleted in the tree incurs no cost within the tree). Finally, the deletion of every non-last leaf in T_i attributes the maximum possible cost on nodes in T' because every other T_j ($j \neq i$) is not empty at the time of deletion.

Because the cost incurred by any deletion sequence consists of the above three types of cost, and deletion sequence π achieves the maximum in each part, the lemma is finally proved.

As shown in [8], $C_{T,max}$ has a lower bound of $\sum_{i=1}^{n}(3\log_3 i - 1) = \Omega(n \log n)$. By using a complete ternary tree, one can bound $C_{T,max}$ from above by $O(n \log n)$. But there is still a gap between the upper bound and the lower bound for the worst case deletion cost, which makes it interesting to find a best tree that minimizes the worst cast deletion cost.

Lemma 2 gives us a recursive way to construct a worst case deletion sequence π and to calculate $C_{T,max}$ for a tree T. Consider Figure 3 as an example. We first consider the subtrees in the bottom layer rooted at v_2, v_3, v_4. For each of these three subtrees, we have the worst case deletion sequence $< v_5, v_6 >$, $< v_7, v_8 >$ and $< v_9, v_{10} >$. Then we consider one layer above, the subtree rooted at v_1 which has three children v_2, v_3, v_4. For this subtree, we have the worst case deletion sequence $< v_2, v_3, v_4 >$. Therefore the worst case deletion sequence for the whole tree is $< v_5, v_7, v_9, v_6, v_8, v_{10} >$. Hence the worst case deletion cost sequence is $< 4, 4, 4, 2, 1, 0 >$ and $C_{T,max} = 15$.

By Lemma 2 and the definition of ancestor weight, the worst case deletion cost $C_{T,max}$ for a tree T can be divided into two parts. We define $C_{T',max}$ as the **skeleton cost** and $\sum_{j=1}^{r} C_{T_i,max} + \sum_{i=1}^{r}(n_i - 1)w_i$ as the **subtree cost**.

We assume that w_i is in non-increasing order, and represent the *ancestor weight vector* $(w_1, w_2, \ldots, w_r)$ as $(k_1^{e_1}, k_2^{e_2}, \ldots, k_s^{e_s})$. For example, $(6, 6, 5, 5, 5, 4)$ is represented as $(6^2, 5^3, 4)$.

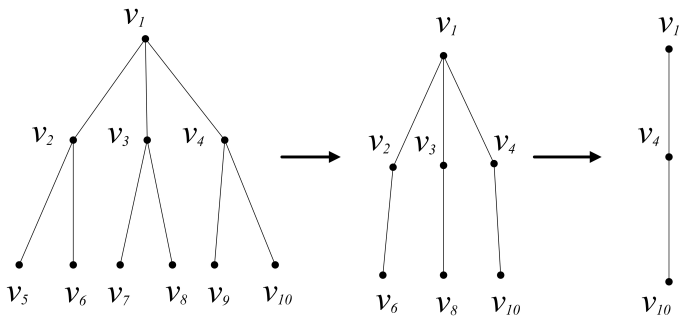

Fig. 3. An example of the worst case deletion sequence.

Definition 5. *We define the optimal tree $T_{n,opt}$ as a tree which has the minimum worst case deletion cost $C_{T,max}$ over any tree T containing n leaf nodes.*

A direct conclusion drawn from the above analysis is the following lemma.

Lemma 3. *There is an optimal tree whose leaf descendant vector $(n_1, n_2, \ldots, n_r)$ is non-decreasing.*

Proof. Because the ancestor weight vector is non-increasing, the subtree cost is minimized when the leaf descendant vector is non-decreasing.

4 Degree Bounds in the Optimal Tree

In this section, we will derive some characteristics of the degree of any internal node in the optimal tree $T_{n,opt}$. First we observe the following lemma.

Lemma 4. *If T is an optimal tree, then any subtree of T is an optimal tree.*

Proof. This follows from the recursive formulation of the worst case deletion cost by Lemma 2.

We seek to characterize the structure of optimal trees by gradually changing the structure of an optimal tree without increasing the deletion cost, while at the same time reducing the possibilities of the degrees or the positions of those degrees.

Lemma 5. *There is an optimal tree $T_{n,opt}$ where every internal node v has at most 5 children.*

Proof. Suppose in an optimal tree, there exists an internal node v where $d_v \geq 6$ and d_v is even ($d_v = 2m$). For any such node, we will transform the structure below it so that it has two children, each having degree m. We denote the original subtree rooted at v as T and the new subtree as T'. We focus on the skeletons in both trees which contain the $2m$ leaves that are originally the children of v.

After the transformation, the skeleton cost is reduced by $C_{T,max} - C_{T',max} = (\sum_{i=1}^{2m-1} i) - (2\sum_{i=3}^{m+1} i + 1) = m^2 - 4m + 3 \geq 0$. For the subtree cost, before the transformation each descendant of v has ancestor weight $2m$. After the transformation, each descendant has ancestor weight $m + 2 < 2m$. Therefore, the subtree cost will be reduced by $(m-2)(n_v - 1) \geq 0$. Hence, the total worst case deletion cost will not increase after the transformation.

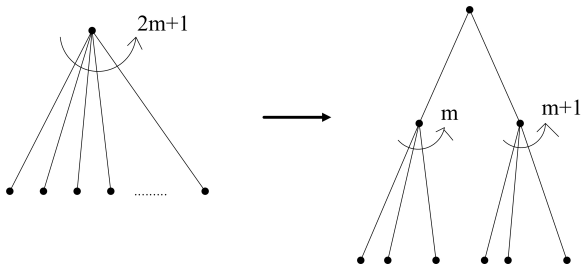

Fig. 4. $d_v \geq 6$ and $d_v = 2m + 1$.

When the node v has odd degree $d_v \geq 7$ and $d_v = 2m + 1$, we can get similar results (refer to Figure 4) . After the transformation, the skeleton cost decreases by $C_{T,max} - C_{T',max} = (\sum_{i=1}^{2m} i) - (m + 2 + 2\sum_{i=3}^{m+1} i + 1) = m^2 - 3m + 1 > 0$. The ancestor weight of each child node of v decreases either by $m - 1$ or $m - 2$ and thus the subtree cost will not increase.

Therefore, the tree after the transformation is still optimal but has no internal node with more than 5 children.

Based on Lemma 4 and Lemma 5, it is possible to design a dynamic programming algorithm to compute an optimal tree with running time $O(n^2)$. However, we will first derive some further properties in Lemma 6, 7 and 8 which will enable us to simplify the dynamic programming algorithm.

Lemma 6. *There is an optimal tree $T_{n,opt}$ where every internal node v has degree at most 5 and the children of nodes with degree 5 are all leaves.*

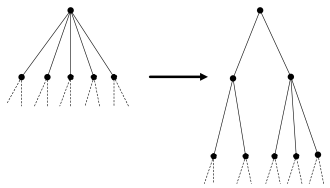

Fig. 5. $d_v = 5$.

Proof. We first find an optimal tree T that satisfies Lemma 5. Suppose in T there exists an internal node v with degree 5 which has at least one child being an internal node. We can transform the subtree T rooted at node v into a new subtree T' rooted at v but with only 2 children where one child has degree 2 and the other has degree 3 as shown in Figure 5.

For the skeleton cost (skeletons are in solid lines), $C_{T,max} = 4+3+2+1 = 10$. And $C_{T',max} = 3+4+3+1 = 11$. The cost has increased by 1 after the transformation. If all of v's children are leaf nodes, then T is better than T', and T can not be transformed into any other better structure in this case. Therefore a degree 5 node may appear in an optimal tree.

However, if at least one of v's children u is an internal node (the subtree rooted at u has at least 2 leaf descendants and therefore $n_u \geq 2$), we will prove that T' is at least as good as T. We know that the ancestor weight vector is (5^5) for T and $(5^3, 4^2)$ for T'. According to Lemma 3, the last entry in the leaf descendant vector is no smaller than 2, which implies that the subtree cost of T' is smaller than that of T by at least 1. This decrease of subtree cost compensates the increase of skeleton cost and therefore makes T' no worse than T. This completes the proof of the lemma.

Lemma 7. *There is an optimal tree $T_{n,opt}$ where every internal node v has degree at most 5 and the children of nodes with degree 5 or 2 are all leaves.*

Proof. We first find an optimal tree T that satisfies Lemma 6. Suppose in T there exists an internal node v with $d_v = 2$ and at least one of its children is not a leaf node. We investigate the node v which is in the lowest level among those that violates Lemma 7. In other word, if v has a child u which has two children, then u's two children are both leaves.

Case 1. The node v has two children, where its left child v_L has $i (2 \leq i \leq 5)$ children and its right child v_R is a single leaf node.

As shown in Figure 6, the subtree rooted at v can be transformed into a new subtree with its worst case deletion cost decreased.

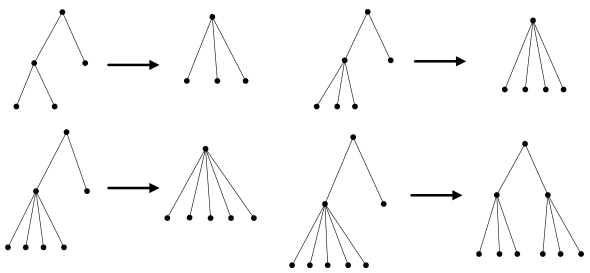

Fig. 6. Case 1.

Case 2. One of v's two children has 2 children and the other child has $i (2 \leq i \leq 5)$ children.

As shown in Figure 7, the subtree rooted at v can be transformed into a new subtree with its worst case deletion cost decreased. The reason we consider the violating node in the lowest level is the following. We take the transformation of the 5-leaf skeleton as an example. Although the skeleton cost is reduced by 1 after the transformation, the ancestor weight vector is increased from $(5^3, 4^2)$ to (5^5). Therefore, if there are large subtrees connected to those 5 positions, the subtree cost will increase a lot after the transformation. However, the increase of the subtree cost will be zero if the degree 2 child of v has no non-trivial subtrees.

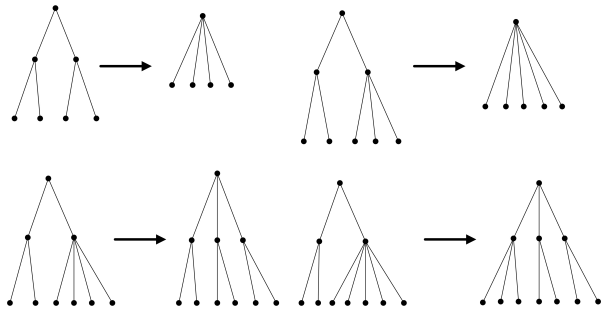

Fig. 7. Case 2.

Case 3. Both of v's children have more than 2 children.

There are in total 5 different combinations. In Table 1, we use (d_1, d_2) to denote the skeleton of a tree rooted at v where v has two children of degree d_1 and d_2 respectively. The skeleton contains v, the children of v and the children of v's children. Let (d'_1, d'_2, d'_3) denote the skeleton of a tree rooted at v having three children of degree d'_1, d'_2 and d'_3 respectively after the transformation. Let C_v denote the skeleton cost for the tree rooted at v before the transformation and C'_v for the skeleton cost after the transformation. Let θ denote the ancestor weight vector for the leaves in the original skeleton and θ' for the leaves in the transformed skeleton.

(d_1, d_2)	(d'_1, d'_2, d'_3)	C_v	C'_v	θ	θ'
$(3,3)$	$(2,2,2)$	15	15	(5^6)	(5^6)
$(3,4)$	$(3,2,2)$	20	20	$(6^4, 5^3)$	$(6^3, 5^4)$
$(4,4)$	$(3,3,2)$	25	25	(6^8)	$(6^6, 5^2)$
$(4,5)$	$(3,3,3)$	31	30	$(7^5, 6^4)$	(6^9)
$(5,5)$	$(4,3,3)$	37	36	(7^{10})	$(7^4, 6^6)$

Table 1. Case 3.

We see that after the transformation the skeleton cost is not increased, while the ancestor weight vector after the transformation is smaller or equal which implies a non-increase in subtree cost.

By the discussion of the previous three cases, we may transform all internal nodes of degree 2 in a bottom-top fashion (i.e. we always deal with the nodes on the lowest level first) without increasing the deletion cost of the tree. We may have to repeat this procedure several rounds, but it will terminate in the end, because in each round the highest level of degree 2 nodes that violate Lemma 7 will go down by at least one.

Lemma 8. *There is an optimal tree $T_{n,opt}$ where every internal node v has degree at most 5 and the children of nodes with degree not equal to 3 are all leaves.*

Proof. We first find an optimal tree T that satisfies Lemma 7. Suppose in T there is a node v which has 4 children and at least one of its children is not a leaf node. There are several cases to consider. We denote the degree of v's four children as a sequence (d_1, d_2, d_3, d_4). We transform this skeleton into a skeleton with root degree 3, and denote the children degree sequence as (d'_1, d'_2, d'_3). Let C_v denote the skeleton cost for the original tree and let C'_v denote the skeleton cost for the tree after transformation. Let θ denote the ancestor weight vector of leaves in the original skeleton and let θ' denote the ancestor weight vector of leaves in the transformed skeleton.

We prove this lemma by enumerating all possible degrees for the four children of the node v. We divide all these different combinations into Case 1 to 5.

Case 1. Three of v's children are leaf nodes (see Figure 8).

There are 3 combinations in this case, all of which become better after the transformation.

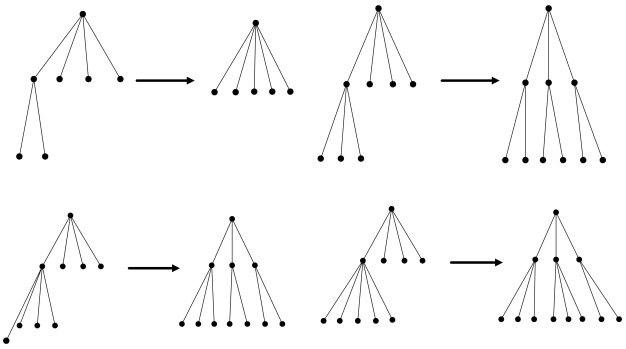

Fig. 8. Case 1.

Case 2. Two of v's children are leaf nodes. (See Table 2.)

There are $C_4^2 = 6$ different combinations in this case. We may find that for all possible degrees of the children of v, there is a transformation of the skeleton so that its worst case deletion cost is decreased. The skeleton cost is decreased from C_v to C'_v. The subtree cost also decreases because the ancestor weight of each node containing non-trivial subtrees is reduced as shown by θ and θ' in the table. Notice that the smallest two entries in the ancestor weight vector are not counted in comparison because they are associated with leaves. Similar arguments apply in Case 3, Case 4 and Case 5.

(d_1, d_2, d_3, d_4)	(d'_1, d'_2, d'_3)	C_v	C'_v	θ	θ'
$(2, 2, 1, 1)$	$(2, 2, 2)$	16	15	$(6^4, 4^2)$	(5^6)
$(3, 2, 1, 1)$	$(3, 2, 2)$	22	20	$(7^3, 6^2, 4^2)$	$(6^3, 5^4)$
$(3, 3, 1, 1)$	$(3, 3, 2)$	28	25	$(7^6, 4^2)$	$(6^6, 5^2)$
$(4, 2, 1, 1)$	$(3, 3, 2)$	29	25	$(8^4, 6^2, 4^2)$	$(6^6, 5^2)$
$(4, 3, 1, 1)$	$(3, 3, 3)$	35	30	$(8^4, 7^3, 4^2)$	(6^9)
$(4, 4, 1, 1)$	$(4, 3, 3)$	42	36	$(8^8, 4^2)$	$(7^4, 6^6)$
$(5, 2, 1, 1)$	$(3, 3, 3)$	37	30	$(9^5, 6^2, 4^2)$	(6^9)
$(5, 3, 1, 1)$	$(4, 3, 3)$	43	36	$(9^5, 7^3, 4^2)$	$(7^4, 6^6)$
$(5, 4, 1, 1)$	$(4, 4, 3)$	50	42	$(9^5, 8^4, 4^2)$	$(7^8, 6^3)$
$(5, 5, 1, 1)$	$(4, 4, 4)$	58	48	$(9^{10}, 4^2)$	(7^{12})

Table 2. Case 2.

Case 3. Only one of v's children is a leaf node. (See Table 3.)

There are $C_5^2 = 10$ different combinations in this case. We find that for all possible degrees of the children of v, there is a transformation of the skeleton so that its worst case deletion cost is decreased.

(d_1, d_2, d_3, d_4)	(d'_1, d'_2, d'_3)	C_v	C'_v	θ	θ'
$(2, 2, 2, 1)$	$(2, 2, 3)$	21	20	$(6^6, 4)$	$(5^4, 6^3)$
$(3, 2, 2, 1)$	$(3, 3, 2)$	27	25	$(7^3, 6^4, 4)$	$(6^6, 5^2)$
$(4, 2, 2, 1)$	$(3, 3, 3)$	34	30	$(8^4, 6^4, 4)$	(6^9)
$(3, 3, 2, 1)$	$(3, 3, 3)$	33	30	$(7^6, 6^2, 4)$	(6^9)
$(3, 4, 2, 1)$	$(4, 3, 3)$	40	36	$(8^4, 7^3, 6^2, 4)$	$(7^4, 6^6)$
$(3, 3, 3, 1)$	$(4, 3, 3)$	39	36	$(7^9, 4)$	$(7^4, 6^6)$
$(3, 3, 4, 1)$	$(4, 4, 3)$	46	42	$(8^4, 7^6, 4)$	$(7^8, 6^3)$
$(4, 4, 2, 1)$	$(4, 4, 3)$	47	42	$(8^8, 6^2, 4)$	$(7^8, 6^3)$
$(4, 4, 3, 1)$	$(4, 4, 4)$	53	48	$(8^8, 7^3, 4)$	(7^{12})
$(4, 4, 4, 1)$	$(5, 4, 4)$	60	55	$(8^{12}, 4)$	$(8^3, 7^{10})$
$(5, 2, 2, 1)$	$(4, 3, 3)$	42	36	$(9^5, 6^4, 4)$	$(7^4, 6^6)$
$(5, 3, 2, 1)$	$(4, 4, 3)$	48	42	$(9^5, 7^3, 6^2, 4)$	$(7^8, 6^3)$
$(5, 3, 3, 1)$	$(4, 4, 4)$	54	48	$(9^5, 7^6, 4)$	(7^{12})
$(5, 4, 2, 1)$	$(4, 4, 4)$	55	48	$(9^5, 8^4, 6^2, 4)$	(7^{12})
$(5, 4, 3, 1)$	$(5, 4, 4)$	61	55	$(9^5, 8^4, 7^3, 4)$	$(8^5, 7^8)$
$(5, 4, 4, 1)$	$(5, 5, 4)$	68	62	$(9^5, 8^8, 4)$	$(8^{10}, 7^4)$
$(5, 5, 2, 1)$	$(5, 4, 4)$	63	55	$(9^{10}, 6^2, 4)$	$(8^5, 7^8)$
$(5, 5, 3, 1)$	$(5, 5, 4)$	69	62	$(9^{10}, 7^3, 4)$	$(8^{10}, 7^4)$
$(5, 5, 4, 1)$	$(5, 5, 5)$	76	69	$(9^{10}, 8^4, 4)$	(8^{15})
$(5, 5, 5, 1)$	$(6, 5, 5)$	84	77	$(9^{15}, 4)$	$(9^6, 8^{10})$

Table 3. Case 3.

Case 4. All v's children are internal nodes and no child has degree 5.

There are $C_6^2 = 15$ different combinations in this case. We find that in all of the cases, the degree 4 node can be transformed into a degree 3 node with adjustments in the tree structure, while the worst case deletion cost of the whole tree does not increased. (See Table 4.) Note that $((2, 3), 3, 4)$ represents a skeleton with 12 leaves.

Case 5. All v's children are internal nodes and at least one child has degree 5.

(d_1, d_2, d_3, d_4)	(d'_1, d'_2, d'_3)	C_v	C'_v	θ	θ'
$(2,2,2,2)$	$(3,3,2)$	26	25	(6^8)	$(6^2,5^2)$
$(3,2,2,2)$	$(3,3,3)$	32	30	$(7^3,6^6)$	(6^9)
$(4,2,2,2)$	$(4,3,3)$	39	36	$(8^4,6^6)$	$(7^4,6^6)$
$(3,3,2,2)$	$(4,3,3)$	38	36	$(7^6,6^4)$	$(7^4,6^6)$
$(3,3,3,2)$	$(4,4,3)$	44	42	$(7^9,6^2)$	$(7^8,6^3)$
$(3,3,3,3)$	$(4,4,4)$	50	48	(7^{12})	(7^{12})
$(4,3,2,2)$	$(4,4,3)$	45	42	$(8^4,7^3,6^4)$	$(7^8,6^6)$
$(4,3,3,2)$	$((2,3),3,4)$	51	50	$(8^4,7^6,6^2)$	$(6^3,7^6,8^3)$
$(4,3,3,3)$	$((2,3),4,4)$	57	56	$(8^4,7^9)$	$(8^3,7^{10})$
$(4,4,2,2)$	$((2,2,1),3,4)$	52	50	$(8^8,6^4)$	$(8^4,7^4,6^4)$
$(4,4,3,2)$	$((2,3),(2,3),3)$	58	58	$(8^8,7^3,6^2)$	$(8^6,7^4,6^3)$
$(4,4,3,3)$	$((2,3),5,4)$	64	63	$(8^8,7^6)$	$(8^3,7^{11})$
$(4,4,4,2)$	$((2,2,1),(2,2,1),4)$	65	64	$(8^{12},6^2)$	$(8^8,7^4,6^2)$
$(4,4,4,3)$	$((2,3),(2,3),5)$	71	71	$(8^{12},7^3)$	$(8^{11},7^4)$
$(4,4,4,4)$	$(5,5,(2,2,2))$	78	77	(8^{16})	(8^{16})

Table 4. Case 4.

We can transform the nodes with degree 5 to a node with 2 children $(2,3)$. Although we increase the skeleton cost by 1, we can still counteract this increase by rearranging the subtrees to reduce the subtree cost. We use one example to illustrate the way we counteract the increase in cost.

Suppose the skeleton we choose for the subtree rooted at v is $(5,3,3,3)$. If all subtrees below the skeleton are leaves which means that there is no subtree cost, then we can change the skeleton to $(5,5,4)$ so that the skeleton cost (also the total deletion cost) is reduced. If some of the subtrees below the skeleton are non-trivial subtrees, then we can change (5) to$(2,3)$ and choose the skeleton to be $(3,3,3,2)$. By doing so, we increase the total deletion cost by 1 and reduce the case to Case 4. Then we refer to Table 4 and find that $(3,3,3,2)$ can be changed to $(4,4,3)$ with skeleton cost reduced by 2 and ancestor weight vector decreased. Therefore, with these two transformations, the worst case deletion cost is reduced by at least 1.

There are more intricate instances in Case 5 which may possibly use the absolute decrease in subtree cost (which usually results from a a large number of non-trivial subtrees) to counteract the increase. We summarize all the subcases in Table 5. Notice that when there are at least k leaf subtrees attached to the skeleton, then we needn't compare the largest k entries in the ancestor weight vector, e.g., $(5,4,3,3)$ has at least 5 leaf subtrees which makes (10^3) contribute nothing to the subtree cost. For $(5,4,4,2)$, we know that subtrees attached to $(9^5,6^2)$ are leaves. Therefore, we only need to compare (8^8) and (7^8). For the subcases $(5,4,4,3)$ and $(5,5,4,3)$, another observation is needed. Notice that it is always better to attach large subtrees to the positions with small ancestor weight. When we attach the three largest subtrees to $(8,6^2)$ and (7^3) respectively, the subtree cost on $(8,6^2)$ is no larger than that on (7^3), which makes the structures after the transformation better in these two subcases.

After proving the previous cases, we conclude that all the children of any degree 4 node can be leaf nodes in an optimal tree. This completes the proof.

(d_1, d_2, d_3, d_4)	reduce to	(d'_1, d'_2, d'_3)	C_v	C'_v	θ	θ'
$(5,2,2,2)$		$(4,4,3)$	47	42	$(9^5, 6^6)$	$(7^8, 6^3)$
$(5,3,2,2)$	$((3,2),3,2,2)$	$((3,2,1),3,3)$	53	52	$(9^5, 7^3, 6^4)$	$(9^3, 8^2, 6^7)$
$(5,4,2,2)$	$((3,2),4,2,2)$	$(4,(3,2,1),3)$	60	58	$(9^5, 8^4, 6^4)$	$(9^3, 8^2, 7^4, 6^4)$
$(5,3,3,2)$	$((3,2),3,3,2)$	$(4,(3,2,1),3)$	59	58	$(9^5, 7^6, 6^2)$	$(9^3, 8^2, 7^4, 6^4)$
$(5,3,4,2)$	$((3,2),3,4,2)$	$(4,4,(3,2,1))$	66	64	$(9^5, 8^4, 7^3, 6^2)$	$(9^3, 8^2, 7^8, 6)$
$(5,3,3,3)$	$((3,2),3,3,3)$	$(4,4,(3,2,1))$	65	64	$(9^5, 7^9)$	$(9^3, 8^2, 7^8, 6)$
$(5,4,3,3)$	$((3,2),4,3,3)$	$(4,4,(3,2,1,1))$	72	70	$(9^5, 8^4, 7^6)$	$(10^3, 9^2, 7^{10})$
$(5,4,4,2)$	$((3,2),4,4,2)$	$(4,4,(3,2,1,1))$	73	70	$(9^5, 8^8, 6^2)$	$(10^3, 9^2, 7^{10})$
$(5,4,4,3)$		$((2,2,2),(2,2,1),(2,2,1))$	79	79	$(9^5, 8^8, 7^3)$	$(8^{14}, 6^2)$
$(5,4,4,4)$	$((3,2),4,4,4)$	$(4,(2,2,2),(3,2,2))$	86	86	$(9^5, 8^{12})$	$(9^3, 8^{10}, 7^3)$
$(5,5,2,2)$	$((3,2),(3,2),2,2)$	$(3,(2,2,1),(3,3))$	68	65	$(9^{10}, 6^4)$	$(8^{10}, 6^4)$
$(5,5,3,2)$	$((3,2),(3,2),3,2)$	$((3,3,1),(2,2,1),3)$	74	74	$(9^{10}, 7^3, 6^2)$	$(9^6, 8^4, 6^5)$
$(5,5,4,2)$	$((3,2),(3,2),4,2)$	$(4,(3,3,1),(2,2,1))$	81	80	$(9^{10}, 8^4, 6^2)$	$(9^6, 8^4, 7^4, 6^2)$
$(5,5,3,3)$	$((3,2),(3,2),3,3)$	$(4,(3,3,1),(2,2,1))$	80	80	$(9^{10}, 7^6)$	$(9^6, 8^4, 7^4, 6^2)$
$(5,5,4,3)$		$((3,2,1),(2,2,2),(2,2,1))$	87	87	$(9^{10}, 8^4, 7^3)$	$(9^3, 8^{12}, 6^2)$
$(5,5,4,4)$		$((3,2,2),(3,2,2),4)$	94	94	$(9^{10}, 8^4, 7^3)$	$(9^6, 8^8, 7^4)$
$(5,5,5,2)$		$((2,2,2),(2,2,2),(2,2,1))$	89	86	$(9^{15}, 6^2)$	$(8^{16}, 6)$
$(5,5,5,3)$		$((3,2,2),(3,2,2),4)$	95	94	$(9^{15}, 7^3)$	$(9^6, 8^8, 7^4)$
$(5,5,5,4)$		$((3,2,2),(2,2,2),(2,2,2))$	102	101	$(9^{15}, 8^4)$	$(9^3, 8^{16})$
$(5,5,5,5)$		$((3,2,2),(3,2,2),(2,2,2))$	110	109	(9^{20})	$(9^6, 8^{14})$

Table 5. Case 5.

We summarize the above discussion into the following theorem.

Theorem 1. *There is an optimal tree $T_{n,opt}$ where*
(1) All internal nodes have degree at most 5*.*
(2) The children of a node with degree not equal to 3 are all leaf nodes.

Denote the deletion cost of the optimal tree $T_{i,opt}$ as C_i. We use D_i to represent the minimum deletion cost of a tree with i leaves when the root degree is restricted to be 2. Based on Theorem 1, we can design a dynamic programming algorithm $DELETE\_OPT$ to compute C_n with running time $O(n^2)$.

Algorithm 1 $DELETE\_OPT$

1. $C_1 = 0$; $C_2 = 1$; $C_3 = 3$; $C_4 = 6$; $C_5 = 10$;
2. $D_1 = 0$; $D_2 = 1$; $D_3 = 4$; $D_4 = 7$; $D_5 = 11$;
3. for $i = 6$ to n
4. $\quad D_i = i^2$; $C_i = i^2$;
5. $\quad$ for $k1 = 1$ to $i/2$
6. $\quad\quad k2 = i - k1$;
7. $\quad\quad$ if $D_i > C_{k1} + C_{k2} + 2 \cdot i - 3$ then
8. $\quad\quad\quad D_i = C_{k1} + C_{k2} + 2 \cdot i - 3$;
9. $\quad\quad$ if $C_i > C_{k1} + D_{k2} + i + 2 \cdot k1 - 3$ then
10. $\quad\quad\quad C_i = C_{k1} + D_{k2} + i + 2 \cdot k1 - 3$;
11. $\quad$ end for
12. $\quad$ end for
13. end for

Theorem 2. *The optimal tree can be computed in $O(n^2)$ time.*

Proof. We first prove that Algorithm $DELETE\_OPT$ computes the deletion cost of the optimal tree. Step 1 assigns basic optimal values for $n = 1, 2, 3, 4, 5$. When $n > 5$, by Theorem 1, the root degree of the optimal tree should be 3. We assume the smallest subtree of the root has $k1$ leaves and treat the remaining structure with $k2$ leaves as a tree with root degree 2. The inner loop tries all the possible combinations of these two parts and returns the combination which minimizes the worst case deletion cost. Meanwhile, we also compute the value D_i by enumerating the possible sizes of the two branches. Therefore, the value C_n output by Algorithm $DELETE\_OPT$ is the worst case deletion cost of an optimal tree. The recursion used to compute D_i can be understood in the following way. We choose the root together with its two children as the skeleton, then deleting the $k1-1$ leaves on one subtree will cost $C_{k1}+2\cdot(k1-1)$ because each leaf deletion will incur an extra cost of 2 on the skeleton compared to its cost in the subtree. The same situation happens on the other subtree. Therefore the total deletion cost will be $C_{k1}+2\cdot(k1-1)+C_{k2}+2\cdot(k2-1)+1 = C_{k1}+C_{k2}+2i-3$. The recursion to compute the value C_i can be interpreted in similar ways. The running time is $O(n^2)$ because there are two nested loops in the algorithm. The structure of $T_{n,opt}$ can be obtained by keeping the branching information in Step 7 and Step 9.

5 Conclusions

In this paper, we have investigated the problem of optimizing deletion cost in multicast key trees. We consider the scenario when all users join in the beginning and subsequently the only membership changes are deletions. This could arise in teleconferencing or other applications where group membership is quite selective and the complete list is drawn up beforehand. We first characterize the worst case deletion sequence which results in the maximum number of encryptions during the whole key updating process. Then we investigate the properties of the optimal tree which minimizes the amortized deletion cost. We prove that there exists an optimal tree where any internal node has degree at most 5. We further show that the children of a node with degree not equal to 3 are all leaf nodes. Based on this, we propose a dynamic programming algorithm to compute the optimal key tree in $O(n^2)$ time.

One area for future work is to study the structure of the optimal trees when some k, $k \leq n$, of the users may leave. One would need to be able to characterize the worst case deletion sequence among all possible combinations of k users. Another interesting direction is to investigate the insertion scenario for a best insertion strategy.

Acknowledgement

We would like to thank the anonymous referees for their valuable comments; especially one referee pointed out that a dynamic programming algorithm based on Lemma 5 would only need $O(n^2)$ time rather than $O(n^3)$ time as stated in our earlier version.

References

1. M. T. Goodrich, J. Z. Sun, and R. Tamassia, *Efficient Tree-Based Revocation in Groups of Low-State Devices*, Proceedings of the Twenty-Fourth Annual International Cryptology Conference (CRYPTO), p.511-527, 2004.
2. J. Goshi and R. E. Ladner, *Algorithms for dynamic multicast key distribution trees*, Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing, p.243-251, 2003.
3. R. L. Graham, M. Li and F. F. Yao, *Optimal Tree Structures for Group Key Management with Batch Updates*, to appear in SIAM Journal on Discrete Mathematics.
4. G. Hao, N. V. Vinodchandran, R. Byrav and X. Zou, *A balanced key tree approach for dynamic secure group communication*, Proceedings of the Fourteen International Conference on Computer Communications and Networks, p.345-350, 2005
5. M. Li, Z. Feng, R. L. Graham and F. F. Yao, *Approximately Optimal Trees for Group Key Management with Batch Updates*, Proceedings of the Fourth Annual Conference on Theory and Applications of Models of Computation, p.284-295, 2007.
6. X. S. Li, Y. R. Yang, M. G. Gouda, and S .S. Lam, *Batch Re-keying for Secure Group Communications*, Proceedings of the Tenth International Conference on World Wide Web, p.525-534, 2001.
7. S. Rafaeli and D. Hutchison, *A Survey of Key Management for Secure Group Communication*, ACM Computing Surveys, v.35 n.3, p.309-329, 2003.
8. J. Snoeyink, S. Suri and G. Varghese, *A lower bound for multicast key distribution*, In Proceedings of the Twentieth Annual IEEE Conference on Computer Communications, p.422-431, 2001.
9. D. Wallner, E. Harder, and R. C. Agee, *Key Management for Multicast: Issues and Architectures*, RFC 2627, June 1999.
10. C. K. Wong and M. G. Gouda, and S. S. Lam, *Secure Group Communications Using Key Graphs*, IEEE/ACM Transactions on Networking, v.8 n.1, p.16-30, 2003.
11. F. Zhu, A. Chan, and G. Noubir, *Optimal Tree Structure for Key Management of Simultaneous Join/Leave in Secure Multicast*, Proceedings of Military Communications Conference, p.773-778, 2003.