Theoretical Computer Science 607 (2015) 272–281

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Optimal trees for minimizing average individual updating cost ☆

Sicen Guo [a], Minming Li [a,*], Yingchao Zhao [b]

[a] City University of Hong Kong, Hong Kong
[b] Caritas Institute of Higher Education, Hong Kong

ARTICLE INFO

Article history:
Received 1 April 2015
Received in revised form 18 August 2015
Accepted 24 August 2015
Available online 29 August 2015

Keywords:
Key tree
Optimality
Individual re-keying
Average case

ABSTRACT

Key tree is a popular model to maintain the security of group information sharing by using a tree structure to maintain the keys held by different users. Previously, researchers proved that to minimize the worst case updating cost in case of single user deletion, one needs to use a special 2–3 tree. In this paper, we study the average case for user update. We prove that in the optimal tree, the branching degree of every node can be bounded by 3 and furthermore the structure of the optimal tree can be pretty balanced. We also show the way to construct the optimal tree when there are loyal users in the group. Finally we discuss about the weighted case where different users have different probabilities to be the first one leaving the group. We design a polynomial time algorithm to construct the optimal tree when the number of different probabilities is a constant.

1. Introduction

Broadcasting is widely used in our daily lives, such as paid television channel and group-chat. For either privacy or profit reason, security (message integrity, confidentiality and authority) has become an important design issue in broadcasting systems. To satisfy the security requirement, key encryption must be adopted in broadcasting/multicasting communication. Only authorized users hold a shared group key, which is used for decrypting broadcasting messages. When a user leaves/joins the group, broadcasting a new shared group key is compulsory to achieve both Backward Access Control (the new users cannot access previous messages) and Forward Access Control (the leaving users cannot decrypt further messages) [10].

There are two re-keying strategies: individual re-keying, and batch re-keying. Individual re-keying means that the group needs to update keys for every user join/leave request, while batch re-keying updates keys only after a certain period instead of immediately. Wong et al. [6] proposed a key tree model to specify secure groups and discussed the performance of individual re-keying. They proved that the complexity of individual re-keying is $\mathcal{O}(\log n)$, where n is the group size. Individual re-keying can achieve both Backward Access Control and Forward Access Control perfectly, but synchronization problem will occur when there is a batch of updates [9]. Meanwhile, batch re-keying can alleviate out-of-sync problems and also improve scalability. Since batch re-keying updates keys in a certain period, it cannot entirely fulfill the Forward Access Control. In most situations, batch re-keying is acceptable if the update period is not too long. However, individual re-keying is more suitable than batch re-keying in certain situations.

☆ This work was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. City 122512].

* Corresponding author.
E-mail addresses: minming.li@cityu.edu.hk (M. Li), zhaoyingchao@gmail.com (Y. Zhao).

http://dx.doi.org/10.1016/j.tcs.2015.08.030
0304-3975/

In the key tree model proposed by Wong et al. [6], the group membership is maintained by a Group Controller (GC). There are three kinds of keys: the group key, individual keys and auxiliary keys, all of which are maintained through a key tree by the GC. The group key, a unique symmetric key (traffic encryption key, shorted as TEK), is used to encrypt the content and the GC should notify all the group users about this key. The individual key is the key known by each individual user and the GC only. In the key tree, the root stores the TEK and each leaf stores an individual key, while the rest (non-root internal node) stores auxiliary keys (key encryption key, shorted as KEK). Whenever a user joins or leaves, the TEK should be updated and notified to the remaining users to guarantee the content security. Since all the users (leaves) know about the keys in the path from itself to the root, the GC should rekey all these keys when certain user change happens. The updating procedure is in bottom-up fashion. For batch re-keying, Graham et al. [2] studied the optimal key tree structure with the assumption that all the users have the same probability p of being replaced by a new user in the batch period, which is extended from [10]. They proved that when $p > 1 - 3^{\frac{1}{3}} \approx 0.307$, the star is optimal, and when $p \leq 1 - 3^{\frac{1}{3}}$, the branching degree of each non-root internal node has an upper bound four. Based on these findings, they provided an $\mathcal{O}(n)$ algorithm for constructing the optimal tree for n users with a fixed probability p. Based on the work of [2], Chan et al. [1] extended the model by introducing loyal users to batch re-keying. They showed that when $p \geq 0.43$, the star is optimal and similar structural results are obtained which enable a dynamic programming algorithm.

For individual re-keying, Snoeyink et al. [5] investigated 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.

(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.

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

Optimal trees with other user join/leave behaviors are studied in [3,4,7,8].

In this paper we study the average case and aim to find the optimal tree structures to minimize the average cost for a single deletion.

The remaining part of this paper is organized as follows. In Section 2, we describe our model and problem in detail. In Section 3, we prove that a ternary balanced tree is optimal in the normal case. Later in Section 4, we show that the same result still holds for one loyal user case. In Section 5, we extend our results to the case of weighted users where the weight represents a user's probably to be the next one leaving the group. Finally, we conclude our work in Section 5.

2. Preliminaries

In the key tree model, each leaf represents a user. Every user has an individual key which is only known by the user itself and the group controller (GC). The root stores a group key called traffic encryption key (TEK), which is used for transmitting encrypted contents. Except for the root, other internal nodes store a key encryption key (KEK), which is used for updating TEK and shared by all its leaf descendants.

In individual re-keying scenario for a popular broadcasting service, a user will be replaced by a new user upon leaving. The GC will assign a new individual key to the new user. At the same time, the GC needs to update all the keys which belong to the ancestors of the updating spot to ensure both Backward Access Control and Forward Access Control. The key updating process is done in bottom-up fashion. Suppose v is the parent node of the updated user and its original group key is k_v. At first, GC will assign the new user an individual key. After that, GC needs to update the key of node v. Assume that the new KEK for v is k_v^{new}. To inform all the children of v of k_v^{new}, the GC uses the individual key of every child of v to encrypt k_v^{new} and broadcasts the encrypted message. In this way, the old user is unable to get the new KEK k_v^{new}. Also, the new user is not aware of the old KEK k_v. In total, the GC needs to broadcast d_v messages, where d_v stands for the number of children of v. The updating process of $v's$ key is done. After that, the parent of v needs to update its key. The GC repeats the updating process until it comes to the root, where the stored TEK is updated.

If there is only one user leaving the system, this user can be any leaf in the tree. Notice that different user's leaving may bring different update costs. We are interested in the tree with minimum average updating cost. When the number of users in the key tree is n, then minimizing average updating cost is equivalent to minimizing total cost of updating any leaf.

Denote the set of leaves in key tree $T = (V, E)$ as $L(T)$ and denote $v's$ ancestor set to be $anc(v)$. Let d_v be the number of children of v. Throughout the paper, we also use degree to denote d_v. In other words, we use "degree" to mean "branching degree". For example, a node with degree 3 means that the node has three children. Our goal is to find a tree with n leaves to minimize the total updating cost $\sum_{v \in L(T)} \sum_{u:u \in anc(v)} d_u$. We use $OPT(n)$ to denote the total updating cost of such a tree with n leaves. Alternatively, this total updating cost can also be expressed as $\sum_{u \in V - L(T)} d_u N_u$ where N_u is the number of leaf descendants of u. When we use the alternative expression of the total cost, we say the cost is contributed by internal nodes. We define a *totally balanced ternary tree* to be a ternary tree where each subtree of the root is a totally balanced ternary tree and the number of leaves in these three subtrees differs by at most 1.

3. Optimal tree structures in the normal case

In this section, we will discuss the structures of the optimal tree. First of all, to minimize the cost, it is easy to see that degree 1 internal nodes cannot exist in the optimal tree. Then we start bounding the degree from above. In all the following figures, a small circle always represents a leaf.