Preventing Collusion Attacks on the One-Way Function Tree (OFT) Scheme

Xuxin Xu[1], Lingyu Wang[1], Amr Youssef[1], and Bo Zhu[2]

[1] Concordia Institute for Information Systems Engineering
Concordia University
Montreal, QC H3G 1M8, Canada
{xuxin_xu,wang,youssef}@ciise.concordia.ca
[2] Center for Secure Information Systems
George Mason University
Fairfax, VA 22030-4444, USA
bzhu@gmu.edu

Abstract. The one-way function tree (OFT) scheme proposed by Balenson *et. al* is widely regarded as an efficient key management solution for multicast communication in large dynamic groups. Following Horng's claim that the original OFT scheme was vulnerable to a collusion attack, Ku *et. al* studied the collusion attack on OFT and proposed a solution to prevent the attack. The solution, however, requires to broadcast about $h^2 + h$ (h is the height of the key tree) keys for every eviction operation, whereas the original OFT scheme only requires about h keys. This modified OFT scheme thus loses a key advantage that the original OFT has over the logical key hierarchy (LKH) scheme, that is a halving in broadcast size. In this paper, we revisit collusion attacks on the OFT scheme. We generalize the examples of attacks given by Horng and Ku *et. al* to a generic collusion attack on OFT, and derive necessary and sufficient conditions for such an attack to exist. We then show a solution for preventing collusion attacks while minimizing the average broadcast size. Our simulation results show that the proposed solution allows OFT to outperform LKH in many cases.

1 Introduction

Multicast communications can greatly save bandwidth and sender resources in delivering data to groups of recipients. However, cryptographic key management schemes are required to ensure the confidentiality of a multicast communication. More specifically, *backward security* requires that a joining member cannot learn previous messages, and *forward security* requires that an evicted member cannot learn future messages. The adjective *perfect* can be added to the two properties, if they can be satisfied against an arbitrary number of colluding members [2].

To satisfy perfect forward and backward security, the group key must be changed whenever a member is added to or evicted from a group. The new key needs to be conveyed to all members at the minimum communication cost since the group is usually large and dynamically changing. Among other methods, the OFT (one-way function tree) scheme, originally proposed by Balenson *et. al*, is one of the most popular schemes for this purpose [1–4, 17]. A key advantage of OFT over another popular method, the Logical Key Hierarchy (LKH) [6], is that OFT halves the number of bits broadcasted upon adding or evicting a member. Specifically, if a key has k bits and the key tree used by OFT and LKH has a height h, then the broadcast size of OFT is $hk + h$ bits, whereas that of LKH is $2hk + h$ bits. OFT achieves such a halving in broadcast size by deriving its key tree in a bottom-up manner, in contrast to LKH's top-down approach. Consequently, unlike the independently chosen keys in LKH, the keys in an OFT key tree are functionally dependent, and this functional dependency allows OFT to save half of the broadcasted bits.

Unfortunately, the same functional dependency among keys that brings OFT the reduced communication cost also subjects it to collusion attacks. Although OFT was claimed to achieve perfect forward and backward security [2], only the collusion among evicted members was considered. A collusion that includes current members was claimed to be uninteresting, because *a (current) member knows the group key*. However, the claim implicitly assumes the colluding members are trying to learn the current group key, which is not necessarily true. An evicted member may collude with a current member to learn group keys that were used after the former was evicted but before the latter joins the group. In this case, OFT will fail on both forward security and backward security. In 2002, Horng first showed an example of collusion attacks on OFT [16]. In 2003, Ku and Chen provided new attack examples to show that the two assumptions required by Horng's attack were actually not necessary conditions [11]. Ku and Chen also proposed a modified OFT scheme that is immune to the collusion attack. The solution, however, needs to broadcast $(h^2 + h)k$ bits on every member eviction (and hk bits on each member addition). Ku and Chen's scheme thus loses a key advantage which OFT has over LKH, that is a halving in broadcast size. Because their scheme requires a broadcast of quadratic size on evicting any member, it is only suitable for applications where member eviction is rare.

In this paper, we revisit collusion attacks on the OFT scheme. To better understand collusion attacks on OFT, we first generalize the examples of attacks given by Horng and Ku *et. al* to a generic attack. Instead of these examples of two or three members, we study the collusion among arbitrary number of evicted and joining members with arbitrary number of other, non-colluding members leaving or joining in between. Based on this understanding of the general attack, we derive necessary and sufficient conditions for a collusion attack on OFT to exist. These conditions reveal that the solution by Ku *et. al* is unnecessarily conservative. Their solution prevents potential collusion attacks by invalidating any knowledge that is brought out of the group by evicted members. However, our results show that such knowledge is not always useful to a joining member in colluding.

We study a different approach where such leaked knowledge is not immediately invalidated but is recorded by a key manager who is responsible for managing the group. When a member joins the group, the key manager then checks whether it is possible for this new member to collude with previously evicted members. If a potential collusion exists, the key manager will update keys as part of the joining operation such that the collusion becomes impossible. Because additional re-keying is performed only when a collusion is possible, this solution has the advantage of minimizing broadcast size. Following the discussion of a straightforward stateful method that has an unacceptable storage requirement, we present a modified version of the method whose storage requirement is proportional to the size of the key tree. These methods pose no additional communication cost on evicting a member but may require more broadcasted bits when a member joins. We study the average performance of the scheme using experiments, and the result show that our scheme is more efficient than LKH in many cases.

The contribution of this paper is two fold. First, our study provides a better understanding of the collusion attack on the OFT scheme. The previous work by Horng and Ku *et. al* have only described specific examples of collusion attacks involving two or three colluding nodes but left the general case open [16]. Our results show exactly what can be computed by an arbitrary collection of joining and evicted nodes. Second, the solution we shall propose makes the OFT scheme secure against general collusion attacks while minimizing the communication overhead. Ku and Chen's solution renders OFT strictly less efficient than LKH, whereas our experimental results show that the solution in this paper enables OFT to outperform LKH in small to medium groups. The results also reveal that OFT's approach of using functionally dependent keys actually renders the scheme less efficient in large groups, if collusion attacks are to be prevented. The rest of the paper is organized as follows. Section 2 reviews the OFT scheme and examples of collusion attacks given by Horng and Ku *et. al*. Section 3 generalizes these examples to a generic attack and derives the necessary and sufficient conditions for the collusion attack. Section 4 studies a solution that minimizes broadcast size while preventing collusion attack. Section 5 studies the performance of our solution through experiments. Section 6 concludes the paper and gives future directions.

2 Related Work

Various aspects of multicast security, including group key management, have been extensively studied, as surveyed in [5, 7, 8, 12, 13]. In [9], an architecture is provided for the management of cryptographic keys for multicast communications. Various security aspects, including ephemeral secrecy, long-term secrecy, and perfect forward secrecy, are outlined in [14]. Popular tree-based group key management schemes include the Logical Key Hierarchy (LKH) scheme [6, 15, 22], the One-way Function Tree (OFT) scheme [1–4, 17], and the One-way Function Chain (OFC) scheme [8]. Unlike many solutions that depend on a trusted group controller, the authors in [10] propose a group key management scheme based on El Gamal, which only requires a partially trusted controller who does not need accesses to the communication keys. The solution in [19] integrates the one-way key derivation with key trees to reduce the communication overhead of rekeying operations. In the solution, the total number of encrypted keys transmitted during a rekeying operation is reduced by not sending new keys to those members who can derive the keys by themselves. The solution proposed in [20] inherits the architecture of the logical key tree algorithm but rekeys the group using a new algorithm. The batch rekeying scheme in [21] is based on one-way function tree and minimum exact covering.

The LKH scheme is shown to be immune to collusion attacks in [18]. On the other hand, Horng first showed that the OFT scheme is vulnerable to a collusion attack in [16]. This result was later revisited by Ku et. al in [11]. We first review the original OFT scheme in Section 2.1 and then review the examples of collusion attack on OFT given by Horng and Ku *et. al* in Section 2.2. In this paper, we do not address collusion attacks on the OFC scheme [8], which comprises an interesting future work.

2.1 The OFT Scheme

The original OFT scheme is an efficient key management scheme for large, dynamically changing groups [1–4, 17]. A *key manager* maintains a balanced binary key tree for each group. The key trees are computed bottom up using a one-way function $g()$ and a concatenation function $f()$ as follows. First, each leaf node v is assigned a randomly chosen *node key* x_v, and a *blinded node key* is computed from the node key as $g(x_v)$. The node key of each interior node v is then computed by concatenating the blinded node keys of its left child $left(v)$ and right child $right(v)$ as: $x_v = f(g(x_{left(v)}), g(x_{right(v)}))$. For example, the key tree in the left hand side of Figure 1 can be constructed as $x_4 = f(g(x_8), g(x_9))$, $x_2 = f(g(x_4), g(x_5))$,$x_7 = f(g(x_14), g(x_15))$, $x_3 = f(g(x_6), g(x_7))$, and $x_1 = f(g(x_2), g(x_3))$.

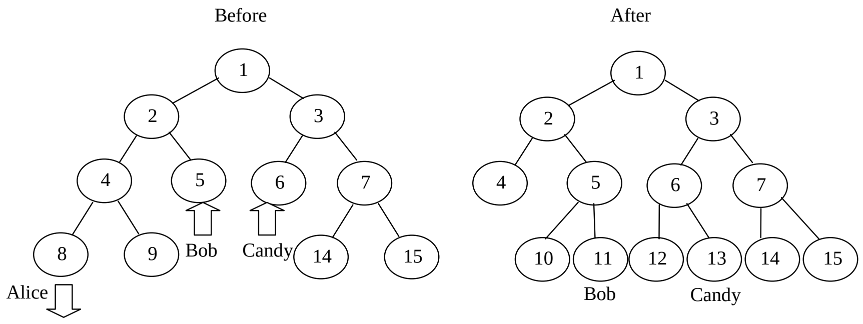


Fig. 1. OFT Key Tree and Collusion Attacks

Each group member is associated with a leaf node in the key tree, and is given its node key. For each node v on the path from that leaf node to the root, the group member is also given the blinded node key of v's sibling. The group member can thus compute the group key, that is the node key of the root [3]. For example, in the left hand side of Figure 1, a member Alice who is associated with the node 8 will be given the blinded keys $g(x_9)$, $g(x_5)$, and $g(x_3)$. Alice can then compute the group key as: $x_4 = f(g(x_8), y_9)$, $x_2 = f(g(x_4), y_5)$, and $x_1 = f(g(x_2), y_3)$.

A new member always joins at a leaf node closest to maintain the balance of the key tree. After the joining, the existing leaf node becomes the left child of a new interior node and is assigned a new node key. The right child is a new node associated with the joining member. The whole path from the interior node to the root will be updated due to the two new keys, and the updated blinded keys must be conveyed to those members who need them. For example, in Figure 1, the joining member Bob causes the existing node 5 to be split into two nodes, with each assigned a new node key. The node keys of node 5, node 2, and node 1 then need to be updated, and their blinded version will be broadcasted to the members who need them (for example node 8 and 9 will need the updated $g(x_5)$). A similar process applies to the other joining member Candy.

The eviction of a member is similar to the addition with following differences. The sibling of the node associated with the leaving member replaces its parent, and is assigned a new node key. Keys on the path leading that node to the root are then updated and their blinded versions are broadcasted, as in the case of addition. However, if the sibling of the leaving member is an interior node, then we cannot directly change its node key due to the functional dependency among keys. Instead, we need to change the node key of a leaf node in the

[3] In a later version of the scheme, the key used for communication is not the node key itself but is derived from the node key using another one-way function [2]

subtree whose root is that interior node. For example, in Figure 1, the evicted member Alice causes the node 9 to replace the node 4. The node keys of nodes 4, node 2, and node 1 will then be updated, and their blinded version will be broadcasted to those who need them.

Let the height of a balanced key tree be h. Then approximately h new blinded keys must be broadcasted on each member addition or eviction (on the other hand, a unicast is used to send the joining member its blinded keys). In addition, h bits are broadcasted to notify members about the position of the joining or eviction. In contrast, the broadcast size of Logical Key Hierarchy (LKH) is $2h$ multiplied by the key size (plus the same h bits for the position of the addition or eviction). The reason that OFT can achieve a halving in broadcast size is that keys in an OFT key tree are functionally dependent, but keys in a LKH key tree are all independent. In OFT, an updated node key is propagated through the sibling of the node, whereas in LKH the key is propagated through the children of the node. The fact that a node has two children but only one sibling explains the difference in the broadcast size of LKH and OFT.

2.2 Examples of Collusion Attack on OFT

Horng observed that the functional dependency among keys in an OFT key tree subjects the OFT scheme to a special collusion attack [16]. Horng gave two conditions for such an attack to exist. Referring to Figure 1, the attack example given by Horng can be described as follows. Suppose Alice, associated with the node 8, is evicted at time t_1, and later Candy joins the group at time t_2 (ignore Bob's joining for the time being). By the OFT scheme, the node key of node 3 is not affected by the eviction of Alice, so Alice knows the blinded version of this key between t_1 and t_2. Moreover, the node key of node 2 is updated when Alice is evicted, and then remains the same even after Candy joins. Candy can thus see the blinded version of this key between t_1 and t_2. Knowing the blinded node key of both node 3 and node 2 between t_1 and t_2, Alice and Candy can collude to compute the group key during that time interval. The OFT scheme thus fails to provide forward security (Alice knows future group key) and backward security (Candy knows previous group key).

Intuitively, the above example is a result of the unchanging keys of the root's children. Horng thus stated two necessary conditions for such an attack to exist, that is the two colluding nodes evicted and joining at different side of the root and no key update happening between time t_1 and t_3 [16]. Later, Ku and Chen showed through two more attack examples that Horng's conditions are actually not necessary [11]. First, referring to Figure 1, if Alice is evicted at time t_1 and Bob joins later at time t_2, then they can collude to compute the node key of node 2 between t_1 and t_2 due to a similar reason. In addition, both Alice and Bob know the blinded node key of node 3 between t_1 and t_2, so they can compute the group key between the same time interval. Second, assume Alice is evicted at time t_1, and Bob and Candy join at time t_2 and t_3, respectively, with $t_1 < t_2 < t_3$. By similar arguments, Alice knows the blinded node key of node 3 between t_1 and t_3, and Candy knows the blinded node key of node 2 between t_2 and t_3. They can thus collude to compute the group key between t_2 and t_3. The two examples show that Horng's two conditions are actually not necessary.

Ku and Chen also provided a solution to prevent the collusion attack on OFT [11]. Intuitively, an evicted member brings out knowledge about some keys that will remain the same for a certain time interval after the eviction. Ku and Chen modify the OFT scheme to change all the keys known by an evicted member upon the eviction. For example, when Alice is evicted in Figure 1, the node key of node 5 and node 3 will be updated (in addition to that of node 4, node 2, and node 1, as required by the original OFT scheme). With this solution, no evicted member can bring out any knowledge about future keys, so a collusion with future joining members is prevented. However, the solution updates the node key of all the h siblings on the path of an evicted node (node 5 and node 3 in above example). Each such update requires the broadcast of h keys (for example, to update the node key of node 3, we must update one of the leaf nodes in the subtree rooted as node 3). The broadcast size is thus h^2 multiplied by the key size plus h bits. Because such a broadcast is required for every eviction, the modified OFT is less efficient than LKH (which broadcast $2h$ keys on an eviction) in most cases, unless member eviction is rare.

3 Generic Collusion Attack on OFT

Section 3.1 first studies a special case, that is an evicted node colludes with another node who joins later. This turns out to be the only interesting case. Section 3.2 then discusses the general case where multiple evicted nodes and joining nodes may collude.

3.1 Collusion Between An Evicted Node and A Joining Node

We first consider the collusion attack between a node A evicted at time t_A and a node C joining the group at time t_C ($t_A < t_C$). Without loss of generality, we assume A is the leftmost node in the key tree, as shown in Figure 2 (notice that this figure actually combines two different key trees at t_A and t_C, which will be justified later in this section). We also need following notations. For any node v, we use $x_{v[t_1,t_2]}$ and $y_{v[t_1,t_2]}$ for its node key and blinded node key between time t_1 and t_2, respectively. We shall also interchangeably refer to a node and the member who is associated with that node. I is the node where the path of A to the root and that of C merges. Let L, R, I', I'' be the left child, right child, parent of I, and parent of I', and let R' and R'' be the right child of I' and I'', respectively. Let B, D, E, and F denote the subtree with the root L, R, $right(I')$, and $right(I'')$, respectively. Let t_{DMIN}, t_{EMIN}, and t_{FMIN} be the time of the first key update after t_A that happens in D, E, and F, respectively. Let t_{BMAX}, t_{EMAX}, t_{FMAX} be the time of the last key update before t_C that happens in B, E, and F, respectively. We then have the following result.

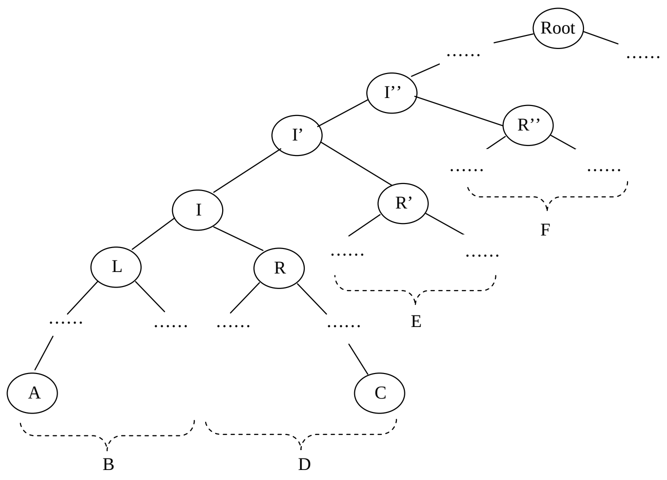


Fig. 2. A Generic Collusion Attack on OFT

Proposition 1. *Referring to Figure 2, the only node keys that can be computed by A and C when colluding are:*

– x_I *in the time interval* $[t_{BMAX}, t_{DMIN}]$,
– $x_{I'}$ *in* $[t_{BMAX}, t_{DMIN}] \cap ([t_A, t_{EMIN}] \cup [t_{EMAX}, t_C])$,
– $x_{I''}$ *in* $[t_{BMAX}, t_{DMIN}] \cap ([t_A, t_{EMIN}] \cup [t_{EMAX}, t_C]) \cap ([t_A, t_{FMIN}] \cup [t_{FMAX}, t_C])$,

and so on, up to the root. Notice that these intervals may be empty.

Proof: When the node A is evicted, it knows the blinded node key of each sibling on its path to the root before the time t_A. This includes $y_{R[-,t_A]}$ and $y_{R'[-,t_A]}$ (recall that the dash means the time when each key is last updated before t_A). By the OFT scheme, the node key of R will not change until a new node joins a node in D (that is, the subtree with the root R) or a node in D leaves, and similarly the node key of R' will not change until a key is updated in E. That is, $y_{R[-,t_A]} = y_{R[-,t_{DMIN}]}$ and $y_{R'[-,t_A]} = y_{R'[-,t_{EMIN}]}$. The node A thus knows these values even after it is evicted. On the other hand, when node C joins, it is given the blinded node key of the siblings on its path to the root. The node C then knows the values $y_{L[t_C,-]}$ and $y_{R'[t_C,-]}$ (recall that the dash here means the time of the next update of these keys after t_C). By the OFT scheme, the node key of L and R' will not be updated when C joins so they have remained the same since the last key update in B and E, respectively. Then we have $y_{L[t_C,-]} = y_{L[t_{BMAX},-]}$ and $y_{R'[t_C,-]} = y_{R'[t_{EMAX},-]}$, which are both known by C.

When A and C colludes, what can be computed depends on the relationship between the timestamps. As shown in Figure 3, A and C can first compute the subgroup key $x_{I[t_{BMAX},t_{DMIN}]} = f(y_{R[-,t_{DMIN}]}, y_{L[t_{BMAX},-]})$. We notice that this statement assumes $t_{BMAX} < t_{DMIN}$. Under this assumption, nodes A and C can compute $y_{I[t_{BMAX},t_{DMIN}]} = g(x_{I[t_{BMAX},t_{DMIN}]})$. This will enable them to further compute another subgroup key I' in two different time intervals. Let $t_{DEMIN} = MIN(t_{DMIN}, t_{EMIN})$ and $t_{BEMAX} = MAX(t_{BMAX}, t_{EMAX})$. Then $x_{I'[t_{BMAX},t_{DEMIN})]}$ can be computed by A and C as $f(y_{I[t_{BMAX},t_{DMIN}]}, y_{R'[-,t_{EMIN}]})$ and $x_{I'[t_{BEMAX},t_{DMIN}]}$ can be computed as $f(y_{I[t_{BMAX},t_{DMIN}]}, y_{R'[t_{EMAX},-]})$. In another word, they can compute the node key of I' in $[t_{BMAX}, t_{DMIN}] \cap ([t_A, t_{EMIN}] \cup [t_{EMAX}, t_C])$. Clearly, this result can be easily extended to the parent of I' and so on, up to the root.

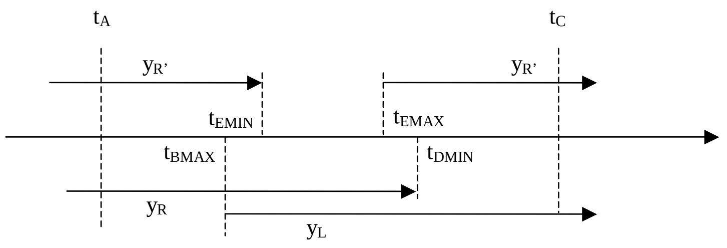


Fig. 3. The Timeline of Collusion Attacks

On the other hand, the above result also depicts all that A and C can compute by colluding. By the OFT scheme, when A is evicted all the node keys along its path to the root are updated, so A no longer knows them. Similarly, C cannot learn any node key on its path to the root prior to its joining. Besides the blinded keys of nodes R, R', and R'' (and all the sibling nodes on the path from I to the root), A may also know the blinded node key of sibling nodes in the subtree B for a time interval after t_A, and similarly C may know about nodes in the subtree D for a time interval before t_C. However, such knowledge does not help them in computing any keys. By the OFT scheme, a node key can only be computed from the blinded key of its two children, but we can never pick a node from the set $B - \{L\}$ and another from $D - \{R\}$ such that they are the children of the same node. $\Box$

One subtlety lies in the dynamics of the key tree. The key tree from which A is evicted is different from the one that C joins. Although we show A and C in the same key tree in Figure 2 for simplicity purpose, the tree structure may have been changed after A leaves and before C joins. However, the key facts that our results depend on will not be affected by such changes. First, A knows $y_{R[-,t_{DMIN}]}$ and $y_{R'[-,t_{EMIN}]}$ regardless of any changes that may happen to the subtree with root L, and the definition of t_{DMIN} and t_{EMIN} excludes any change in the subtree with root R and R' to happen before t_{DMIN} and t_{EMIN}, respectively. It is worth noting that the whole subtree with root L may disappear due to evictions, and consequently the node R will replace its parent I (and the node R will be replaced by $right(R)$) by the OFT scheme. In this case, it seems that A will no longer know y_R even when no key update happens in the set D, invalidating the result that A knows $y_{R[-,t_{DMIN}]}$. However, this is not true. When the node R replaces I, the OFT scheme also requires it to be assigned a new node key, which means at least one of the leaf nodes in the set D must change its node key. That is, a key update does happen in D in this operation, and our result still holds. Similarly, C knows the value $y_{L[t_{BMAX},-]}$ regardless of any change in the key tree after the last key update in the set B.

3.2 The General Case

We first consider other cases of collusion between pairs of evicted and joining nodes and show that the above eviction-joining scenario turns out to be the only interesting case, as explained by Proposition 2. We then discuss the collusion among more than two nodes, and we show that it is sufficient to only consider collusion between pairs of nodes, which is stated in Proposition 3.

Proposition 2. *A pair of colluding nodes A and C cannot compute any node key which they are not supposed to know by the OFT scheme, if*

- *A is evicted after C joins.*
- *A and C both join.*
- *A and C are both evicted.*

Proof: First, we consider the joining-eviction case. In Figure 2, suppose C first joins the group and later A is evicted. If A and C collude, then they trivially know all node keys in the intersection of their paths to the root (for example, node I and I') and the siblings (for example, node R') before C joins and after

A is evicted, because A is in the group before C joins and C stays in the group after A is evicted. In addition, although A knows the blinded node key of some siblings in the subtree B and C knows the blinded node key of some siblings in the subtree D, these keys cannot be combined to compute any node key since no two nodes share a parent. In summary, two nodes colluding in the joining-eviction case cannot compute any node key besides what they already know.

Next consider the eviction-eviction case. Suppose in Figure 2 A is first evicted at time t_A and later C is evicted at time t_C. Because C stays in the group longer than A does, their knowledge about the shared keys in the intersection of their paths (such as I and I') and the siblings (such as R') is the same as C's knowledge. That is, colluding with A does not help C with respect to these keys. Similar to the above cases, A's knowledge about nodes in the subtree B cannot be combined with C's knowledge about nodes in D to compute any node key. The only exception is their knowledge about L and R, which can potentially be combined to compute I (and consequently I' and so on). However, the OFT scheme updates the node key of R when C is evicted, so A can at best know $y_{R[-,t_A]} = y_{R[t_A,t_C]}$ (if no other key update happens between t_A and t_C), which is useless to C. In summary, two evicted nodes colluding cannot compute any node key in addition to what is already known by the later-evicted node. The joining-joining case is similar to the eviction-eviction case and is omitted. $\Box$

Proposition 3. *An arbitrary collection of evicted nodes and joining nodes can collude to compute some node key not already known, if and only if the same node key can be computed by a pair of nodes in the collection.*

Proof: The if part is trivial, and the only if part can be justified as follows. To compute $x_{v[t_1,t_2]}$, the colluding nodes must know both $y_{left(v)}$ and $y_{right(v)}$ for some time intervals that are supersets of $[t_1, t_2]$. Suppose $y_{left(v)}$ is known by m nodes in time period $[t_{ai}, t_{bi}] (1 \leq i \leq m)$, and $y_{right(v)}$ is known in $[t_{cj}, t_{dj}] (1 \leq j \leq n)$. Because $(\bigcup_{i=1}^{m}[t_{ai}, t_{bi}]) \cap (\bigcup_{j=1}^{n}[t_{cj}, t_{dj}])$ is a superset of the non-empty time interval $[t_1, t_2]$, it cannot be empty, either. Consequently, there must exist a pair of i and j such that $[t_{ai}, t_{bi}] \cap [t_{cj}, t_{dj}] \neq \phi$. The pair of nodes that has such knowledge (no single node can possess this knowledge because we assume $x_{v[t_1,t_2]}$ is not already known by the colluding nodes) can thus collude to compute x_v during the time interval $[t_{ai}, t_{bi}] \cap [t_{cj}, t_{dj}]$. $\Box$

We now show that the attack examples given by Ku et al., as described in Section 2.2, are special cases of our generic attack. Referring to Figure 1, the first example says that Alice evicted at t_1 colludes with Bob joining at t_2, and Candy joins at t_3 ($t_1 < t_2 < t_3$). This corresponds to the case where $A = 8$, $C = 5$, $I = 2$, $I' = 1$ (referring to Figure 2), and Candy joins at t_3 in the set E. We thus have $t_{BMAX} = t_1$, $t_{DMIN} = t_2$, and $t_{EMIN} = t_{EMAX} = t_3$. It then follows that Alice and Bob can collude to compute $x_{2[t_1,t_2]}$ and $x_{1[t_1,t_2]}$ (notice that $[t_1, t_2] \cap ([t_1, t_3] \cup [t_3, t_2]) = [t_1, t_2]$). The second example says that Alice evicted at t_1 colludes with Candy joining at t_3, with Bob joining in between at t_2. This corresponds to the case where $A = 8$, $C = 6$, $I = 1$ (I' does not exist), and Bob joins in the set B. We thus have $t_{BMAX} = t_2$ and $t_{DMIN} = t_3$, and consequently Alice colluding with Candy can learn $x_{1[t_2,t_3]}$.

4 A Solution For Preventing Collusion Attacks

The previous section shows that a joining node may collude with previous evicted nodes to compute node keys in certain time intervals, which none of them is supposed to know. However, these results also show that such a collusion is not always possible, and whether it is possible depends on the temporal relationship among joining and evicted nodes. As discussed in Section 2.2, Ku and Chen's solution prevents any evicted node from bringing out knowledge about future node keys. Although it suffices to prevent any collusion attack, this conservative approach has a quadratic broadcast size (in the height of the key tree) on every member eviction and thus is less efficient than the LKH scheme in most cases.

One apparent way to reduce the broadcast size is to update additional keys only when a collusion attack is indeed possible. Unfortunately, this cannot be achieved with Ku and Chen's approach of updating the siblings along the path of an evicted node, because at the time a node is evicted, we do not yet know with whom it may collude in the future. On the other hand, our results in Section 3 make it possible to check whether a joining node can collude with any previously evicted node. If a collusion is possible, we can update a minimum number of additional keys to prevent the joining node from combining its knowledge with the evicted node for that specific collusion. This approach minimizes the communication cost for each joining operation (the eviction operation has no additional communication cost) because a key is updated only when necessary.

We first describe a *stateful* method that explicitly records all the knowledge of evicted nodes. This straightforward method simply applies the results in the previous section to check for possible collusions. However, because the method needs to keep information about all evicted nodes, the storage requirement is proportional to the number of all evicted nodes, which is not acceptable in most applications. Later in this section, we modify this method such that its storage requirement becomes proportional to the size of the key tree. Both methods will eliminate collusion attacks while minimizing the broadcast size.

A Stateful Method For the stateful method, the key manager tracks all evicted nodes and checks whether a joining node can collude with any previously evicted node. If a collusion is possible, additional key updates are performed to remove the joining node's knowledge about past node keys such that the collusion becomes impossible. The key manager needs to record two kinds of knowledge. First, the knowledge about *future* node keys that each evicted node brings out of the group. Second, the knowledge about *past* node keys that a joining member is given when it joins. For this purpose, the key manager stores a modified key tree as follows. Each node in the OFT key tree is now associated with a pair $< t_u, L >$, where t_u is a timestamp and L is a collection of timestamp pairs $< t_{x1}, t_{y1} >, < t_{x2}, t_{y2} >, \ldots, < t_{xn}, t_{yn} >$.

The OFT scheme will be modified such that the timestamp t_u records the time that the current node was last updated, and each pair $< t_{xi}, t_{yi} >$ records the time interval in which some evicted node knows the blinded node key of the current node. For example, Figure 4 shows such a modified OFT tree. Due to space limitation, only the three nodes I, L, and R have part of their timestamps shown in the figure. In the example, nodes A, B, and D were evicted at time t_A, t_B, and t_D, respectively. Another node C joined at time t_C. The node R was only updated once between t_A and t_B, and the update happened at time t_2. The node I was last updated at time t_1, which is before t_D (t_1 is equal to either t_2 or t_3). In the table attached to R, the timestamp t_2 records the time of its last update. The first pair $< t_A, t_2 >$ records the fact that node A knows the value $y_{R[t_A,t_2]}$. The second pair $< t_B, - >$ records that B knows the value $y_{R[t_B,-]}$ (that is, the value of y_R from t_B until now). In the table attached to I, t_1 is the last update time of I, and $< t_D, - >$ records that node D knows the value $y_{I[t_D,-]}$. In the table of L, the timestamp t_3 records the time of its last update.

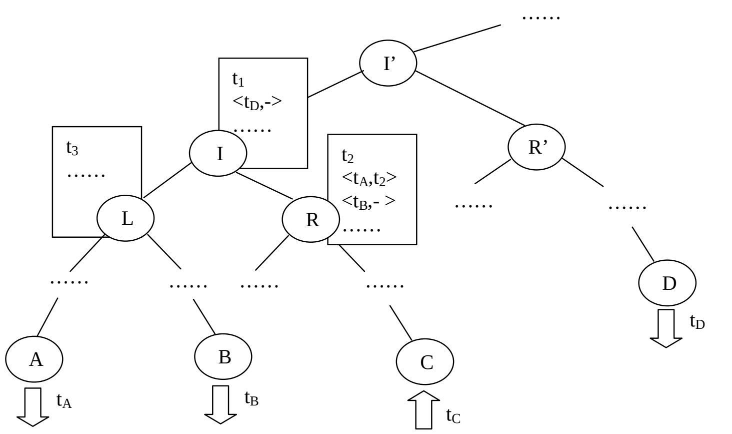


Fig. 4. A Stateful Method for Preventing Collusion Attack

The OFT scheme is modified as follows to update the timestamps and to stop collusions when they become possible. When a node v is evicted at time t, the key manager will also insert a pair $< t, - >$ into each sibling node along the path of v to the root. For example, in Figure 4 the pair $< t_B, - >$ is inserted to the table attached to node R when node B is evicted at time t_B because R is a sibling of L and L is on the path of B to the root. After a node v joins the group, the key manager will check if v can collude with any previously evicted node to compute any node key along the path of v to the root. In Figure 4, after the node C joins the group, for each node on the path of C to the root (excluding C), the key manager needs to do the following. Taken R as an example, the key manager will check whether the intersection $[t_3, -] \cap ([t_A, t_2] \cup [t_B, -])$ is empty. If the intersection is not empty, then the node key x_L will be updated, such that C can no longer collude with A and B to compute the node key x_I (in applications where only the root's key needs to be secure, the key manager can ignore the collusion of a subgroup key here).

Whenever the key manager updates the node key of a node v, regardless of the reason of this update, it will take following two additional actions. First, it will change the corresponding timestamp t_u associated with v to be the time of the current update. Second, it will scan all pairs of timestamps associated with v and change every dash in these pairs to the current time. The second action records the fact that the key update has invalidated the evicted node's knowledge about this node key. For example, in Figure 4 when the node A leaves, a pair $< t_A, - >$ is inserted into the table attached to R. Later at time t_2 the node key R is updated for some reason, and the dash in $< t_A, - >$ is replaced by the current time t_2, leading to the pair $< t_A, t_2 >$ shown in the figure. This reflects the fact that A no longer knows the new node key of R after time t_2.

An Improved Method With Linear Storage Requirement The stateful method keeps all necessary information for checking possible collusions. This requires the key manager to build up an infinitely increasing list of evicted nodes, which is not acceptable in most applications. A closer look at the method reveals that it is not necessary to keep the whole list, if no collusion is to be tolerated. Actually for each node, it suffices to only keep at most one pair of timestamps (plus the timestamp for its last update). The storage requirement is thus linear in the size of the key tree, because for each node at most three timestamps need to be stored. Following two observations jointly lead to this result.

First, in Figure 4, if $t_A < t_B < t_2$, then after B is evicted the list of timestamps associated with R will be $< t_A, - >, < t_B, - >$. However, the pair $< t_B, - >$ is redundant and can be removed because $[t_B, -]$ is a subset of $[t_A, -]$. In another word, after the first pair of timestamps with a dash appears in the list, no other pair of timestamps needs to be stored until the next key update happens to the current node. Second, suppose in Figure 4 $t_A < t_2 < t_B$ is true, so none of $< t_A, - >$ and $< t_B, - >$ is redundant. We then have that $t_2 < t_B \leq t_3$ ($t_B \leq t_3$ holds, because t_3 is the time when x_L is last updated and the eviction of B will update x_L). Now that we know $t_2 < t_3$, the pair $< t_A, t_2 >$ can be safely removed, because the interval $[t_A, t_2]$ will never have a non-empty intersection with $[t_3, -]$.

Based on these two observations, we modify the eviction operation and key update operation of the stateful method as follows. First, when a node v is evicted at time t and a pair of timestamps $< t, - >$ is to be inserted into each sibling node along the path of v to the root, the key manager inserts this pair only if the pair of timestamps already associated with v does not contain a dash. Second, whenever the node key of a node v is updated, the key manager deletes any pair of timestamps associated with the sibling of v that does not contain a dash. For example, in Figure 4 if another node in the subtree with root R' is evicted after t_D but before I is updated, then nothing will be inserted into the table shown in the figure. If I is updated and the dash in $< t_D, - >$ is replaced, then this new pair will stay until the next key update in the subtree with root R'.

5 Empirical Results

This section compares the average communication overhead of our solution, the LKH scheme, the original OFT scheme, and Ku and Chen's modified OFT scheme. Among the four schemes, the original OFT scheme is vulnerable to collusion attacks, and it is included as a baseline to show the additional overhead for preventing collusion attacks. Both our scheme and the modified OFT scheme by Ku and Chen can prevent collusion attacks. The keys in an LKH key tree are independently chosen, so LKH is not vulnerable to the collusion attack discussed in previous sections. We expect our scheme to outperform Ku and Chen's scheme in most cases, because the latter has a quadratic broadcast size for every eviction operation. We also expect our scheme to have a smaller average-case broadcast size than the LKH scheme in some cases.

The communication overhead is measured as the total number of keys broadcasted during a random sequence of joining and eviction operations. We do not consider the unicast of keys to a new member. As discussed in previous sections, collusions depend critically on the order of joining and eviction operations (on the other hand, the specific time duration between these operations is not significant). Starting from an initial key tree of G nodes, a sequence of totally N operations are performed using each of the four schemes. The probability that each operation is the eviction of a member is P (and that of a joining operation $1 - P$). As required by the OFT scheme, the position for each joining operation is chosen to be a leaf node closest to the root. For each eviction operation, the node to be evicted is randomly chosen among all existing leaf nodes.

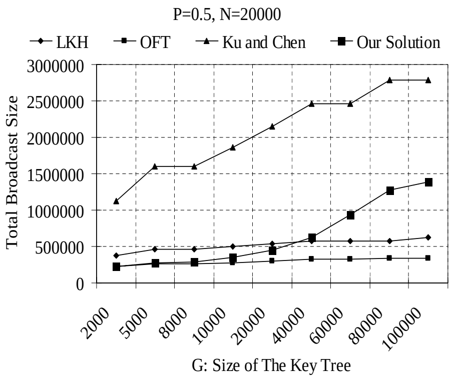
The left hand side of Figure 5 shows the total broadcast size (the number of keys to be broadcasted) versus the size of the key tree. Totally 20000 operations are performed (about half of them are evictions). As expected, the communication overhead of our solution is much less than that of Ku and Chen's scheme (their scheme broadcasts about five times more keys). Compared to the original OFT scheme, our scheme only has small additional overhead until the key tree size increases over 20000 nodes. The broadcast size of our scheme is also smaller than LKH when the key tree size is smaller than 40000. Table 1 shows a more detailed comparison between the two schemes.

For larger key trees, our scheme is less efficient than LKH. As shown in the second row of Table 1, the broadcast size of our scheme is about double the size of LKH when the key tree has 80000 or more nodes. This can be explained by the fact that more collusions are possible in a larger tree, as shown in Table 1, and the larger height of the tree also increases the number of keys to be broadcasted upon each key update. Ku and Chen's scheme also has a similar trend as ours, which confirms that to prevent collusion attacks, both modified OFT schemes are less scalable than LKH. However, because our scheme only perform additional key updates when necessary, the broadcast size for each operation is already minimal. This indicates an inherent disadvantage of using functionally dependent keys in the face of collusion attacks. For large groups where perfect forward and backward security is important, the LKH scheme will be a better choice.

Key Tree Size	2000	5000	8000	10000	20000	40000	60000	80000	100000
Our Solution/LKH	0.59	0.60	0.62	0.70	0.84	1.08	1.61	2.19	2.24
No. of Collusions	242	1063	2113	5154	10385	18991	38417	54720	61799
Height of The Tree	10	12	12	13	14	15	15	16	16

Table 1. Comparing Our solution to LKH

The right hand side of Figure 5 shows the total broadcast size versus the number of operations, with about half of the operations being evictions, on a key tree with 10000 keys. Because collusion attacks depend on the order of operations but not on the specific time durations, we can also regard the number of operations as the intensity of operations, and Figure 5 thus also shows the broadcast size versus the degree of group dynamics. The broadcast size of all four schemes increases with the number (intensity) of operations. The original OFT scheme, the LKH scheme, and our modified OFT scheme all scale in roughly the same manner, whereas Ku and Chen's scheme is less scalable. The column chart inside Figure 5 shows the total number of collusions. Interestingly, while



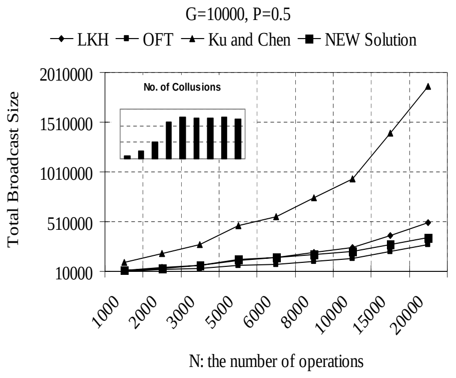


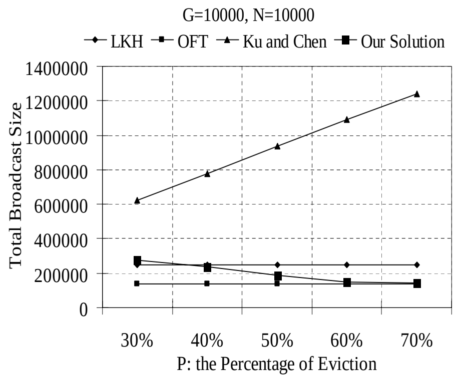
Fig. 5. The Broadcast Size Versus Key Tree Size and The Number of Operations

the number of collusions remains roughly the same when the number of operations goes over 6000, Ku and Chen's scheme still shows a significant increase in the broadcast size, because their scheme requires additional key updates for every eviction operation even when such operation do not cause collusion (in contrast, our scheme scales in the same way as the original OFT).

Figure 6 shows the total broadcast size versus the ratio of evictions among all operations. The two experiments differ in the key tree size and in the total number of performed operations. In both experiments, the original OFT scheme and the LKH scheme have a constant broadcast size because in both schemes the joining and eviction require the same amount of keys to be broadcasted. The broadcast size of Ku and Chen's scheme increases linearly in the ratio of eviction, because their scheme requires additional key updates and hence additional broadcasted bits on every eviction operation but not on the joining operation. Our scheme shows an interesting pattern. The broadcast size first increases with the eviction ratio and then decreases after the ratio reaches about 40%. This is explained by the column chart inside the figure, which shows the total number of collusions. Because a collusion requires both joining nodes and evicted nodes, the total number of collusions reaches a maximal value when about half of the operations are evictions. The maximal broadcast size shifts a little to the left (40% instead of 50%) because our scheme requires additional key updates for joining nodes, but not for evicted nodes. Each joining node thus contributes to the overall broadcast size slightly more than an evicted node does.

6 Conclusion

We studied collusion attacks on the one-way function tree (OFT) scheme. The OFT scheme achieves a halving in broadcast size in comparison to the LKH scheme. However, OFT's approach of using functionally dependent keys in



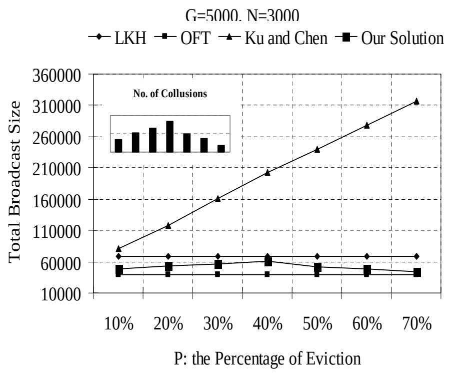


Fig. 6. The Broadcast Size Versus the Ratio of Eviction

the key tree also renders the scheme vulnerable to collusion attacks between evicted members and joining members. We have generalized previous observations made by Horng and Ku *et. al* [16] into a generic collusion attack on OFT. This generalization also gave a necessary and sufficient condition for the collusion attack on OFT. Based on this condition, we have proposed a modified OFT scheme. The scheme is immune to the collusion among an arbitrary number of joining and evicted members, and it minimizes the broadcast size for each operation. The scheme has a storage requirement proportional to the size of the key tree. Experiments show that our scheme has smaller communication overhead than the LKH scheme for small to medium groups. For large groups, the increasing number of collusions renders the OFT scheme a less efficient choice than LKH. As future work, we will investigate cases where the compromise of some sub-group keys is an acceptable risk. Such a relaxed security requirement will likely lead to reduced communication overhead.

Acknowledgements The authors are grateful to the anonymous reviewers for their valuable comments.

References

1. D. McGrew, A. David, T. Alan, and A. Sherman, Key establishment in large dynamic groups using one-way function trees, TIS Report 0755, TIS Labs at Network Associates, Inc., Glenwood, MD, 1998.
2. A.T. Sherman, D.A. McGrew, Key establishment in large dynamic groups using one-way function trees, *IEEE Transactions on Software Engineering*, Volume 29, Issue 5, Pages 444-458, May 2003.
3. D.M. Balenson, D.A. McGrew, and A.T. Sherman, Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization, InternetDraft(work in progress), Internet Engineering Task Force, draft-irtf-smug-groupkeymgmt-oft-00.txt., August 2000.

4. D.M. Balenson, D.A. McGrew, and A.T. Sherman, Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization, InternetDraft(work in progress), Internet Engineering Task Force, draft-balenson-groupkeymgmt-oft-00.txt., February 1999.
5. K. Peter, A survey of multicast security issues and architectures, In *Proceedings of 21st National Information Systems Security Conference*, Pages 408-420, October 1998, Arlington, VA.
6. D. Wallner, E. Harder, R. Agee, Key Management for Multicast: Issues and Architectures, IETF, Request for Comments (RFC) 2627, June 1999.
7. M.J. Moyer, J.R. Rao, P. Rohatgi, A survey of security issues in multicast communications, *IEEE Network*, Volume 13, Issue 6, Pages 12-23, 1999.
8. R. Canetti, J. Garey, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas, Multicast security: A taxonomy and efficient constructions, In *Proceedings of IEEE InfoComm'99*, vol. 2, Pages 708-716, Mar. 1999.
9. H. Harney, C. Muckenhirn, and T. Rivers, Group key management protocol architecture, IETF, RFC2093, 1997.
10. H. Khurana, R. Bonilla, A. Slagell, R. Afandi, H.S. Hahm, and J. Basney, Scalable Group Key Management with Partially Trusted Controllers, In *Proceedings of International Conference on Networking*, 2005.
11. W.C. Ku, S.M. Chen, An improved key management scheme for large dynamic groups using one-way function trees, In *Proceedings of 2003 International Conference on Parallel Processing Workshops*, Pages 391-396, October 2003.
12. D. Matthew, J. Moyer, J.R. Rao, and P. Rohatgi, A Survey of Security Issues in Multicast Communications, *IEEE Network Magazine*, November/December 1999.
13. T. Hardjono, and L.R. Dondeti, 2003. Multicast and Group Security. Artech House, Boston, London, ISBN 1-58053-342-6.
14. R. Canetti and B. Pinkas, A taxonomy of multicast security issues, dracanetti-secure-multicast-taxonomy-00.txt, IETF Internet Draft (work in progress), 1998.
15. H. Harney and E. Harder, Logical Key Hierarchy Protocol, Internet Draft (work in progress), draft-harney-sparta-lkhp-sec-00.txt, Internet Engineering Task Force, Mar. 1999.
16. G. Horng, Cryptanalysis of a Key Management Scheme for Secure Multicast Communications, *IEICE Trans. Commun.*, vol. E85-B, no. 5, Pages 1050-1051, 2002.
17. A.T. Sherman, A proof of security for the LKH and OFC centralized group keying algorithms, NAI Labs Technical Report No. 02-043D, NAI Labs at Network Associates, Inc., 2002.
18. J. Fan, P. Judge, M. Ammar, HySOR: Group Key Management with Collusion-Scalability Tradeoffs Using a Hybrid Structuring of Receivers, In *Proceedings of the IEEE International Conference on Computer Communications Networks*, Miami, 2002.
19. J.C. Lin, F. Lai, H.C. Lee, Efficient Group Key Management Protocol with One-Way Key Derivation, In *Proceedings of The 2005 IEEE Conference on Local Computer Networks*, Pages 336-343, 2005.
20. Y. Wang, J. Li, L. Tie, H. Zhu, An efficient method of group rekeying for multicast communication, In *Proceedings of the 6th IEEE Circuits and Systems Symposium*, Pages 273-276, June 2004.
21. S. Xu, Z. Yang, Y. Tan, W. Liu, S. Sesay, An efficient batch rekeying scheme based on one-way function tree, In *Proceedings of The IEEE International Symposium on Communications and Information Technology*, Pages 490- 493, 2005.
22. C.K. Wong, M. Gouda, and S.S. Lam, Secure group communications using key graphs, *ACM Computer Communication Review*, vol. 28, no. 4, Pages 68-79, September 1998.