Key Management for Simultaneous Join/Leave in Secure Multicast

G. Noubir, F. Zhu, A. H. Chan
College of Computer Science
Northeastern University, Boston, MA, USA
{noubir, zhufeng, ahchan}@ccs.neu.edu

Abstract: In this paper, we address the problem of key update for secure group communication. We focus on the case where multiple requests for join or leave are received within short intervals of time. We have shown that there exists an optimal tree structure that minimizes the communication complexity of the key update. We introduce and compare several algorithms for simultaneous key update, efficient tree construction, and tree adaptation to variable requests load. We also show how the bandwidth requirement for key management can be traded with coarser group access control.

1 Introduction

Multicast communication offers the potential of delivery of data and multimedia to multiple receivers using fewer resources than unicast communication. Several applications can benefit from multicast such as delivery of stock market information, electronic newspapers, video-conferencing, interactive games, digital battlefield, etc. To be commercially exploited multicast communication has to be secured such that only authorized group members can access the multicast session content. Management of the multicast group keys faces a serious scalability problem. Indeed, whenever the group membership changes (i.e., a new member joins or a member leaves) the multicast group key has to be changed. We have previously proposed an algorithm that reduces the communication complexity of key update for a single join/leave from O(*N*) to O(log *N*) [Noub98, Noub99, NA99]. This algorithm was independently discovered in [WGL98]. The principle of this algorithm is to map the users with the leaves of a tree. Each user is provided with all the keys on the path from his leave to the root. The root key is the group key. When a user leaves/joins all the keys in his possession have to be renewed. This can be done starting from the lowest level in the tree. If the tree is binary (Figure 1), at each level the new key can be securely communicated by encrypting it using the two lower level keys. Since the tree has O(log *N*) levels the communication complexity of the key update is also O(log *N*).

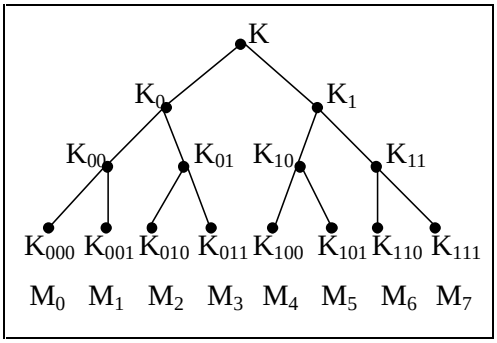

Figure 1. Example of a binary key-tree. A key update requires only two messages per tree-level.

The previously proposed algorithms process one request at a time and are optimal when the requests for key update are received one at a time. However, several multicast applications have high group dynamic or have time variant group dynamic. This results in simultaneous key update requests. The reason for simultaneity could be that the application trades bandwidth with fine grain group access control maintenance, or because the group is so dynamic that the key update

cannot keep up with the request queue due to the limited bandwidth available for key update. For example a stock market multicast application, where the users pay for the duration of time they receive stock quotes, would probably have a large number of simultaneous requests to join or leave the multicast group. A pay-TV multicast application would experience a high number of requests to join the group in the beginning of a program or movie and a high number of requests at the end of the program. Algorithms proposed for the key update [Noub98, WGL98, Noub99, NA99] have a communication complexity $O(\log N)$ for a single join/leave request in a group of size N. This implies a communication complexity of $O(M \log N)$ for M requests. We are interested in algorithms that reduce the communication cost of simultaneous key update requests under variable group dynamic. In addition to processing simultaneous requests, the group controller of a multicast application can also trade bandwidth with strict group structure maintenance. Indeed, the group controller can process group update requests every T units of time (See Figure 2). The simultaneous processing of all requests received during the last T units of time can reduce the average communication complexity of the key update if adequate algorithms are used.

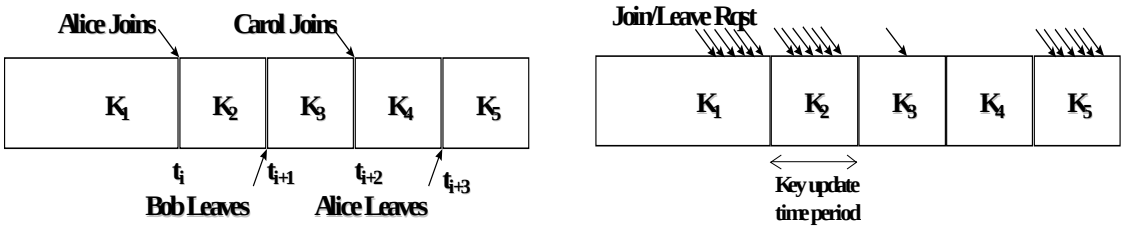

Figure 2. Single join/leave key update versus simultaneous key update with variable period time.

In section 2, we summarize our theoretical results on the structure of the optimal key-tree and provide an algorithm for determining the optimal tree for a given probability of key update. In section 3, we generalize our key update algorithms to the case when simultaneous requests are received. In section 4, we show that the bandwidth required for securing the multicast group can be reduced if traded against a coarser access control.

2 Problem statement and theoretical results

2.1 Optimal tree structure

Given a dynamic multicast group such that simultaneous key update requests are possible. How can we reduce the communication complexity of key update given that, at each step of the algorithm of key update, each user has a probability p to request an update of the group key? In the rest of the paper, we assume that all users have the same probability of leaving or joining the group p. The proposed algorithms can be extended to the case where each user has a specific probability for joining/leaving the group. This can be done using similar techniques as proposed in [Poov99].

In [ZNC01], we have shown that the optimal tree structure that minimizes the communication complexity of key update has a special structure. These results are summarized in the following theorem.

Theorem (optimal key structure): Let G be a multicast group where $|G| = N = 2^k$. Each user has a probability p to request a key update. If we restrict the key-tree to trees where each node can have an arity of 2^i, then the optimal tree that minimizes the communication complexity of the key update (i.e., sum of sizes of all packets used for key update) has the following structure: $2^a\ 2^{2^2}\ 2^{2^2}$

$2^2 \ldots 2^2$ $[2^2|2^1]$. The root node has arity 2^a, the lowest level node has arity 2^2 or 2^1, and all intermediary nodes have arity 2^2. The value of *a* depends on *p*.

The detailed proof of the theorem is described in [ZNC01]. Using dynamic programming we verified the validity of the theorem. The optimal tree structure for $N = 2^{16}$ is given in Table 1. The intuition behind this structure is that for reasonably high values of *p* all the top levels keys have to be changed. In this case a flat structure becomes more efficient.

p	Average number of update requests	Tree structure: $2^a\ 2^2\ 2^2 \ldots 2^2 2^{2\|1}$
0.5	65536	16
0.25	43504	14 2
0.062	18437	12 2 2
0.031250	11201	11 2 2 1
0.007813	6596	10 2 2 2
0.003906	3804	9 2 2 2 1
0.001953	2156	8 2 2 2 2
0.000977	1205	7 2 2 2 2 1
0.000488	666	6 2 2 2 2 2
0.000244	365	5 2 2 2 2 2 1
0.000122	198	4 2 2 2 2 2 2
0.000061	107	3 2 2 2 2 2 2 1
0.000031	57	2 2 2 2 2 2 2 2
0.000015	29	2 2 2 2 2 2 2 2

Table 1. Tree structure as a function of the probability of a user requesting a key update.

2.2 Determining the optimal tree structure

To determine the optimal tree structure one can notice that whenever the probability of key update *p* is below some value p_{thrsh} then a single level tree (of arity $N = 2^k$) is not optimal and has to be broken. If the optimal tree has structure $2^a\ 2^2\ 2^2 \ldots 2^2 2^{2|1}$ then the probability to update the keys on the first level is: $1-(1-p)^{2^{k-a}}$. Combining the threshold probability and looking at the first two levels of the tree we can deduce the tree structure of the optimal tree. We assume that p_{thrsh} is pre-computed and stored in a table $p_{thrsh}[N]$.

***k* ($N = 2^k$)**	3	4	5	6	$k > 7$
p_{thrsh}	0.262647	0.291468	0.292888	0.292893	0.292893

Table 2. Value of p_{thrsh} as a function of *k*.

Theorem: The optimal tree structure is either $2^a\ 2^2\ 2^2 \ldots 2^2 2^{2|1}$ or $2^{a-1}\ 2^2\ 2^2 \ldots 2^2 2^{2|1}$, where $1-(1-p)^{2^{k-(a+1)}} < p_{thresh} \le 1-(1-p)^{2^{k-a}}$. This result is based on the fact that in an optimal tree any sub-tree should also be optimal [ZNC01].

The following algorithm returns the optimal tree structure for a group of *N* users, each requesting a key update with probability *p*.

Algorithm ***Optimum-Tree*** **($N = 2^k$:** number of users**, *p*:** probability of user key update**)**

Let *a* = smallest number *s.t.* $1-(1-p)^{2^{k-(a+1)}} < p_{thresh}[N] \le 1-(1-p)^{2^{k-a}}$;

IF (*Comm.-Complexity*($2^a\ 2^2\ 2^2 \ldots 2^2 2^{2|1}$**) > *Comm.-Complexity*(**$2^{a-1}\ 2^2\ 2^2 \ldots 2^2 2^{2|1}$**))**
THEN *optimal-tree-structure* = $2^a\ 2^2\ 2^2 \ldots 2^2 2^{2|1}$
ELSE *optimal-tree-structure* = $2^{a-1}\ 2^2\ 2^2 \ldots 2^2 2^{2|1}$**;**
/* The arity of the last level in the tree depends on if *k* and *a* have the same parity */
END-ALG

The following algorithm computes the communication complexity of updating the tree keys when each user has probability *p* to request an update and when the tree has structure $2^{a1}\ 2^{a2} \ldots 2^{an}$.
Algorithm *Comm.-Complexity*(*p*, $2^{a1}\ 2^{a2} \ldots 2^{an}$**)**

$$P_{an} = 1-(1-p)^{2^{an}}; C_{an} = 2^{an} P_{an};$$

For (i = *n*-1 to i = 1**)**

$$P_{ai} = 1-(1-P_{a(i+1)})^{2^{ai}}; C_{ai} = 2^{ai}(C_{a(i+1)} + P_{ai});$$

RETURN(C_{a1}**);**
END-ALG

3 Algorithms for simultaneous join/leave

In this section, we first recall the key update algorithm for a single join/leave request. Then we present several algorithms for simultaneous key update. For sake of simplicity we consider the generic key update request (i.e., it could be a join or leave request). Deriving the specific algorithm for the join, or leave is easy. The group controller maintains a tree structure of *l* levels. A key is associated to each leave and node. Each user is assigned to a leave identified by $M_{a1\ldots al}$. Each user is given all the keys on the path from his leave to the root of the tree $\{K, K_{a1}, K_{a1a2}, \ldots, K_{a1\ldots al}\}$. We generalize the key update algorithm from a fixed arity tree to a multilevel tree.

3.1 Processing of requests one at a time

Algorithm Key-Update-Single-Rqst ($M_{a1\ldots al}$**)**
For *i* = *l*-1 .. 0
***Select*(**$K_{a1\ldots ai}$**); /* select a new key for level *i* */**
***Send*({** $E_{Ka1\ldots aiaj}(K_{a1\ldots ai})$ **| for** $0 \le a_j <$ ***arity*(level *i*+1)}**)
/* update all nodes affect by the key update by sending the new level *i* key encrypted by level *i*+1 keys ***/**
END-ALG

If the tree is binary, this algorithm has a communication complexity of $2\mathrm{Log}_2(N)$ (See Figure 1).

3.2 Multiple requests processing

This algorithm processes key updates every *T* units of time. Simultaneous processing of multiple requests allows changing a key only once if it is on the path of several users being processing.

Algorithm Key-Update-Multiple-Rqst (*T*: time update period**)**
Every (*T* units of time)
U = {set of users who requests a key update during the last *T* units of time}
For *i* = *l*-2 **to** 0
For-All ($a_1 \ldots a_i$ ***s.t.*** $\exists\ M a_1 \ldots a_i a'_{i+1\ldots} a'_{l-1} \in$ U**)**
***Select*(**$K_{a1\ldots ai}$**); /* select new keys that have to be updated for level *i* */**

Send({ $E_{Ka1...aiaj}(K_{a1...ai})$ | **for** $0 \le a_j \le$ ***arity*****(level *i*+1)});**

END-ALG

We compared through simulation a fixed 4-ary tree where requests are processed one at a time, fixed 4-ary tree with simultaneous processing of key update requests, and simultaneous processing of requests over an optimal tree. The simultaneous processing of requests over an optimal tree obviously performed better than the other algorithms. Furthermore the identification and construction of the optimal is easy thanks to the algorithm provided in Section 2.2.

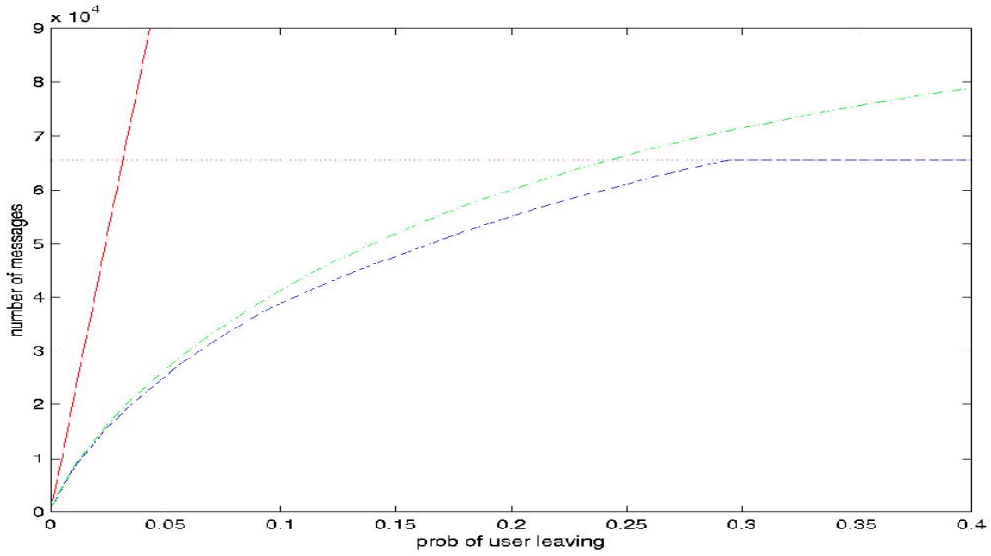

Figure 3. Number of messages for key update as a function of the probability *p* of a user leaving. The red curve corresponds to a fixed 4-ary tree. The green curve corresponds to a fixed 4-ary tree where key update requests are processed simultaneously. The blue line corresponds to simultaneous processing of requests on the optimal tree.

3.3 Adaptive trees

In some multicast sessions the number of join/leave requests can change during the session lifetime. For example the number of requests at the beginning of a session is generally higher than the number of requests in the middle of a session. If this is the case the tree manager has to change the tree structure to adapt to these variations. In order to efficiently adapt the tree structure, we only use the trees of structure $2^a\ 2^2\ 2^2\ \dots\ 2^2$. These trees are almost optimal and have the advantage that switching from $2^a\ 2^2\ 2^2\ \dots\ 2^2$ to $2^{a+2}\ 2^2\ 2^2\ \dots\ 2^2$ can be easily achieved by skipping the transmission of the keys of the second level of the tree. Switching from $2^{a+2}\ 2^2\ 2^2\ \dots\ 2^2$ to $2^a\ 2^2\ 2^2\ \dots\ 2^2$ can be achieved by gradually introducing an additional second level in the tree and transmitting the corresponding keys.

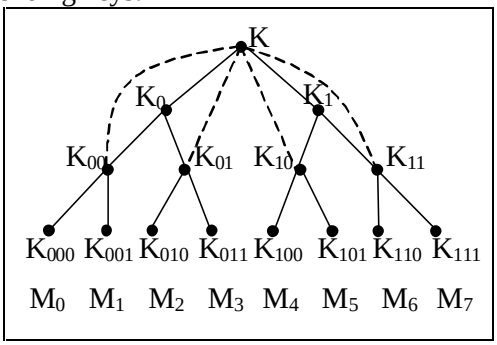

Figure 4. Switching from a 2 2 2 structure to 2^2 2 can be done by skipping the update of the keys of the second level. The reverse can be done by gradually introducing an additional layer.

4 Bandwidth versus access control granularity

The number of messages as a function of the probability of leaving can be used to devise another scheme that reduces the bandwidth requirements of key management. Here, we assume that we have some flexibility for deciding when to update the group keys. If requests are grouped and processed simultaneously, then using the tree structure reduces the total number of messages per unit of time. Let $C(p, T)$ be the communication cost per unit of time of key update when the users have a probability p of leaving and T is the key update period of time. This corresponds to the curves shown in Figure 3, where T is also the unit of time (we only consider the best curve in the figure). If within T the probability of a user leaving is p then the probability of a user leaving within nT is upper-bounded by np. Thus the communication complexity of key update $C(p, nT) \leq C(np, T)/n$ (we divide by n to normalize the cost to a single period T). For example if p = 0.001, then (from the green-curve): $C(p, T)$ = 1254 messages. Taking n = 5 , $C(np, T)/n$ = 958, which is 24% better than $C(p, T)$. Similarly taking n=20, gives $C(np, T)/n$ =726.5, which is 46% better than $C(p, T)$. This is a considerable gain in bandwidth. The drawback of delaying the processing of the key update requests is that some users will remain within the group for 5 or 20 more units of time. It is up to the application designer/operator to trade-off precise access control with bandwidth.

5 Conclusion

In this paper, we have summarized our results on efficient key update for secure multicast. We have shown that when simultaneous key update requests can be processed simultaneously then there exists an optimal tree structure for key management. We provided an efficient algorithm for determining the optimal tree structure as a function of the probability that a user requests a key update. We compared several key update algorithms and shown that it is possible to efficiently adapt the key-tree when the probability of key update changes during the lifetime of a multicast session. Finally, we have shown that it is possible to trade-off the bandwidth requirement of key management with a coarser access control of the group membership.

Reference

[NA99] G. Noubir, and L. von Allmen (1999), "*Security Issues in Internet Protocols over Satellite Links*", In Proceedings of the IEEE Vehicular Technology Conference (VTC'99), Holland.

[Noub98] G. Noubir (1999), "*Optimizing Multicast Security over Satellite Links*", European Space Agency Project, Work package 20 report, version 0.1, April 1998.

[Noub99] G. Noubir (1999). "A Scalable Key Distribution Scheme for Dynamic Multicast Groups." In Proceedings of the Third European Research Seminar on Advances in Distributed Systems, Madeira Island, Portugal.

[Poov99] R. Poovedran (1999), "Key Management for Secure Multicast Communications", Ph.D. Thesis, University of Maryland, College Park, 1999.

[WGL98] Ch.-K. Wong, M. Gouda, and S. S. Lam (1998), "*Secure Group Communications Using Key Graphs*", Proceedings of ACM SIGCOMM, Vancouver, British Columbia, September 1998.

[ZNC01] F. Zhu, G. Noubir, and A. H. Chan (2001), "*Optimal Tree Structure for Key Management of Simultaneous Join/Leave in Secure Multicast*", Technical Report, Wireless Security Laboratory, Northeastern University, Boston, MA, USA, October 11th, 2001.