A FLEXIBLE CATEGORY-BASED COLLUSION-RESISTANT KEY MANAGEMENT SCHEME FOR MULTICAST

Claudiu Duma, Nahid Shahmehri, Patrick Lambrix

Department of Computer and Information Science, Linköpings universitetet, Sweden
{cladu, nahsh, patla}@ida.liu.se

Abstract: Current key management schemes for multicast provide either no resistance to collusion or perfect resistance to collusion. However, resistance to collusion is achieved at the expense of efficiency in terms of the number of transmissions and the number of keys that are used. We argue that applications may have certain assumptions regarding the users and their access to the multicast channel that may be used to provide a larger choice for balancing efficiency against resistance to collusion.

Starting from a user categorization, based on the accessibility to the multicast channel, we formalize the collusion requirement. Different user categorizations give different degrees of collusion resistance and we show that the existing work has focused on special cases of user categorizations. Further, we propose and evaluate a flexible key management strategy for the general case where the accessibility relation defines the order of exclusion of the categories. The theoretical and experimental results show that our scheme has good performance regarding transmissions and keys per controller.

Key words: multicast, security, key management, collusion, efficiency, category-based

1. INTRODUCTION

IP Multicast provides an efficient way for one-to-many communication. Unfortunately, IP Multicast is inherently insecure. Data is transmitted in plain text and is accessible to any hostile party that has access to the communication channel. Assurance on the secrecy of the multicasted data can be provided by

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35691-4_52

D. Gritzalis et al. (eds.), *Security and Privacy in the Age of Uncertainty*

encrypting all transmitted data with a secret key available only to sender and to authorized receivers. We call this secret key, the session key. The group of authorized receivers can be very dynamic, new users are included into the group while others leave due to, for instance, expiration of their subscription. Every time the group of authorized receivers changes, the secret session key shared by that group must be updated accordingly. This process is called re-keying and it must be carried out in a secure and scalable way.

Due to the large number of group members the scalability of the key management scheme is crucial. The requirements are in terms of minimizing the number of transmissions required to re-key as well as the storage requirements (number of keys) for each receiver user and for the group controller.

The security requirements for re-keying are usually in terms of backward and forward access control. Backward access control refers to the impossibility of a newly joined user to gain access to previous session keys. Forward access control refers to the impossibility of a former group member that has been excluded to gain access to future session keys. A third requirement, resistance to collusion, refers to the impossibility of a coalition of two or more excluded members to gain access to future session keys by combining their keying material. This requirement is usually implicitly treated by the previous work in multicast security. Moreover, it usually takes extreme forms. Either no resistance to collusion, or the opposite, perfect resistance to collusion is provided. The resistance to collusion is achieved at the expense of efficiency. However, applications may have certain assumptions regarding members and accessibility to the multicast channel. Based on this we argue that adequate key management schemes can be designed to balance the given collusion constraints against efficiency.

Starting from a user categorization based on the accessibility to the multicast communication channel we formalize the collusion requirement. Different configurations of categories give different constraints on collusion. We show that "no resistance to collusion" and "perfect resistance to collusion" are special cases of our more general collusion requirement and they can be specified by configurations of categories and their relationships. Further, we give a key management strategy for the general case. We assess the efficiency and compare with the referenced work.

2. KEY TREE SCHEMES AND COLLUSION

As shown by the existing work in multicast security the key management schemes based on a logical tree arrangement of the auxiliary keys bring good balance between storage requirements and the number of transmissions [15, 16, 17]. The idea is to recursively partition the multicast group into subgroups, and assign auxiliary keys to these subgroups. Whenever a user leaves the

multicast group the controller uses these auxiliary keys to re-key the remaining users on a subgroup basis by using a minimum number of subgroups covering all non-excluded users. Because the auxiliary keys are actually used to encrypt other keys, they are sometime called Key Encryption Keys [15, 16]. In this paper we use the term auxiliary keys and we refer to the tree arrangement as the auxiliary key tree or just the key tree.

We categorize the key tree schemes based on the properties of their key arrangement. In particular, we make the distinction between the unique keys scheme (UKS) and the reuse keys scheme (RKS):

- In the UKS all keys in the auxiliary key tree are distinct;
- In the RKS, not all the keys are distinct, but for a partition factor of 2 each level of the tree will introduce only two new auxiliary keys.

Logical tree distribution of auxiliary keys reduces the communication overload when re-keying. As shown in [18] the lower bound complexity for re-keying communication when the group key is encrypted using only one key is O(logN). This is a property of UKSs, such as the Logical Key Hierarchy (LKH) [15, 16, 17], One-Way Function Tree (OFT) [11] or One-Way Function Chain (OFC) [3], providing perfect resistance to collusion. More efficient results are achieved by reusing auxiliary keys among different groups of users and encrypting the group key with a combination of these keys (e.g. XOR-ing the keys). This way ad-hoc groups can be created which usually cover more users than the groups pre-established by the key tree. The Boolean Function Minimization (BFM) [4] and the Flat [15] schemes fall in this category. However, the RKS is susceptible to collusion attacks. In [4] it is suggested to artificially increase the key tree such that the probability of collusion would decrease. However, such a solution may not be satisfactory for applications that have certain requirements regarding collusion. The collusion requirement and its formalization are the subject of the next section.

3. COLLUSION REQUIREMENT

As a step towards a definition of collusion we introduce the notion of categories. We say that users belonging to the same category have the same accessibility to the multicast group. A binary accessibility relation is defined on the set of categories, reflecting the fact that users from one category have access to at least all sessions that users of another category have access to, and maybe more. The collusion requirement of a certain application can be defined as a specific configuration of the categories and of the accessibility relation. We introduce the following notation:

- U is the set of users and N is the total number of users.
- C is the set of accessibility categories and M is the total number of categories.

- The function users: $C \rightarrow \wp(U)$, where users(c) gives all the users of category c. We assume that a user can belong to only one category, therefore: $\forall\ c_1, c_2 \in C: c_1 \neq c_2 \rightarrow users(c_1) \cap users(c_2) = \emptyset$. Users with the same accessibility belong to the same accessibility category. All users need to belong to a category, that is $\bigcup_{i=\overline{1,M}} users(c_i) = U$.
- $\prec \subseteq C \times C$, a reflexive partial order, is the accessibility relation on categories reflecting the relative access to the multicast communication the users from one category have compared to users from another category. If $c_1 \prec c_2$, then users from c_2 can access at least all sessions that users from c_1 have access to.
- $E \subseteq C$ the set of excluded categories.

With the above notation we define the collusion requirement as follows: *No coalition of users from a subset I of excluded categories, $I \subseteq E$, can gain access to future session keys unless $\exists c \in C \setminus E, \exists c' \in I: c \prec c'$.*

In other words, a coalition of users from a subset I of excluded categories can gain access to future session keys only if there is a category in I that can access more than a not yet excluded category.

An example scenario that can be modeled by our definition of collusion is the software delivery [7, 8] where users can be categorized according to how much they buy. Some users could buy only the initial software product, while others could subscribe to a certain number of updates and future versions. This would correspond to a "buy more" total order, as observed in [8]. However, software can split in different version branches (e.g. separating features) or merge different tools into one single software product. Also, users could subscribe for more than one product or more than one branch of the same product. Therefore a more general model of this scenario is the partial order, as proposed in this paper. We observe that this order also intuitively relates to the order in which users are excluded from the group of authorized receivers. Although our accessibility relation generalizes the buy more from [8], it is not limited only to that scenario, but it can model any situation where the exclusion order is known or can be inferred.

3.1 Discussion

According to our definition, the collusion requirement for an application depends solely on the configuration of the category set C and of the accessibility relation $\prec$. We discuss a number of possible configurations.

1. $C = \{c\}$ and $\prec = \{<c,c>\}$. All users are in one category. This case corresponds to applications that have no constraints with respect to collusion. Solutions for this case such as BFM and Flat reuse auxiliary keys and achieve best efficiency.
2. $C = \{c_1, c_2, \ldots c_M\}$ and $\prec$ is a total order. Categories are defined. In this case the controller knows exactly the order in which the categories will be excluded. The solution proposed in [8] addresses this case by using BFM with a special arrangement of users such that users from "buy less"

categories can not collude over users from "buy more" categories. Although the number of keys is kept low, the solution imposes too strict constraints regarding the maximum cardinality of each category.

3. $C = \{c_1, c_2, \ldots c_M\}$ and $\prec$ is a reflexive partial order. This is the general case. The controller knows that some order exists among some of categories, where other categories are incomparable. Special solutions do not exist.
4. $C = \{c_1, c_2, \ldots c_M\}$ and $\prec = \{<c_i, c_i>\}_{i=1, M}$. In this case the controller cannot make any assumption regarding the order the categories are going to be excluded. Leaving the multicast group happens in an undeterministic manner. No prior knowledge is available about how much access one category has compared to another. Special solutions do not exist.
5. $C = \{c_1, c_2, \ldots c_M\}$, $M = N$, $users(c_i) = \{u_i\}$ and $\prec = \{<c_i, c_i>\}_{i=1, M}$. This is a special configuration of the previous case where each category directly maps to a user. Again, leaving the multicast group happens in an undeterministic manner. The collusion requirement can be rewritten as *no coalition of excluded users can get access to future session keys.* This is the most common requirement in the existing work for securing multicast. Although solutions for this case provide perfect resistance to collusion, this comes at an expense on efficiency.

In the next section we propose a key management strategy for the general case of collusion, which takes into consideration the existence of different categories of users. We show that our solution is more efficient in terms of transmissions, than the solutions for case 5, which has been the premise for most of the work in securing multicast.

4. SOLUTION STRATEGY

We consider a centralized architecture, where one group controller manages the users' access to the group communication channel. The controller initiates all the sessions, generates and distributes the keys, maintains information about the categories, and accordingly removes and adds users and categories from the multicast channel. Although the central controller can be a potential performance bottleneck, this architecture is more desirable in systems where one party must retain the control such as in multicast software delivery [7, 8]. Methods of replication can be used to unload the controller. For other systems such as GPS that use one-to-many unidirectional satellite communication it is hard if not impossible to use a decentralized architecture.

We assume that *categories are excluded based on the accessibility relation* and we formulate the requirement that is used in the design of our solution:

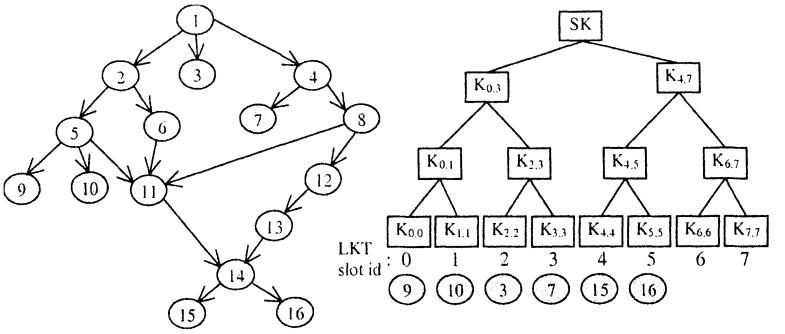
(i) $\forall c, c' \in C : c \in E \wedge c' \prec c \rightarrow c' \in E$; *(ii) No coalition of users from excluded categories can gain access to future session keys.*

This requirement implies the collusion requirement but maps better to operations performed by the controller and is therefore more helpful when approaching the solution. In particular, the $\prec$ relation can be mapped to the knowledge the controller has regarding the order in which the categories must be excluded. Our new problem is then to find a mechanism to exclude a category when the order of exclusion is known, where the exclusion must have perfect resistance to collusion.

To illustrate and exemplify the solution we use the graph representation of the partial order relation $\prec$, e.g. figure 1.a, and call it the categories accessibility graph (CAG). The CAG is obtained from the diagraph of $\prec$ by eliminating all length one cycles, implied by the reflexive property, and all the arcs implied by the transitivity property. The result is a non-transitive and non-reflexive diagraph where the nodes stand for categories and an arc A$\leftarrow$B corresponds to the relation A$\prec$B between the categories A and B. In the remainder we talk about categories and nodes representing them interchangeably.

We start from the observation that if A$\prec$B (or A$\leftarrow$B in CAG) then whenever node A has to be re-keyed node B will also need to be re-keyed. Therefore we introduce the propagation principle: *if A$\prec$B then give node B the keys that are used for re-keying A*. This will assure that whenever A is re-keyed B is also re-keyed, without the need to specifically re-key B and its ancestors.

However this will not solve the case when category A must leave the group while B, together with its ancestors, remain in the group. Therefore: *if A$\prec$B then B must have at least one key that is not known to A*. This key will be used to re-key B when A leaves. Again we can apply the first principle for the ancestors of B, so that the key of B is given to its ancestors too.



a) Categories Accessibility graph (CAG) b) Leafs Key Tree (LKT)

Figure 1. LKT corresponding to the CAG

We also make the observation that at any time any of the leaf nodes can un-deterministically be excluded, independent of the exact branches where they are situated. For the leaf nodes we use a unique key scheme, particularly the LKH key arrangement, and construct the key tree corresponding to the leaf nodes in CAG, naming it the leafs key tree (LKT). We refer to keys in LKT using the notation from [15]. A key $K_{m,n}$ denotes that the key is shared by all nodes with the slot identifier i such that $m \le i \le n$.

Figure 1 shows an example CAG and its corresponding LKT. In this case node 9 will be given the session key SK and the auxiliary keys $K_{0,3}$ $K_{0,1}$ and $K_{0,0}$. Following the propagation principle these keys will also be given to nodes 5, 2, and 1. Moreover, node 5 has a key known only to it and its ancestors.

Each leaf node A in the CAG receives the session key SK and a set of auxiliary keys from LKT, corresponding to the node's slot identifier:

$$keys_{LKT}(A) = SK \cup \{K_{m,n} \mid m \le i \le n \ \wedge i = LKTslotid(A)\}.$$

Each interior node B has a key K_B. Further if $A \prec B$, then all the keys of A are given also to B. In general the key assignment is:

$$keys(X) = \begin{cases} K_X \bigcup_{y \in descendants_{CAG}(X)} keys(Y), if \ X \notin leafs_{CAG} \\ keys_{LKT}(X), if \ X \in leafs_{CAG} \end{cases} \quad (1)$$

where $leafs_{CAG}$ is the set of leaf nodes in CAG and $keys_{LKT}$ as defined above.

The re-keying algorithm to exclude a leaf node X from CAG is:

Step 1. Exclude node X from the LKT following the algorithm for exclusion in the LKH as in [15, 16, 17]. This assures that all leafs, except for the excluded leaf will get the new session key SK^{new}. When removing node 9, for instance, nodes 10, 3, 7, 15 and 16 will be re-keyed. Since their ancestors know the keys of these nodes, the ancestors will also be able to decrypt these messages and learn the new auxiliary keys as well as the new SK^{new}.

Step 2. Further two cases are distinguished:

Step 2.1. No new nodes become leafs in CAG. In this case all the remaining nodes have been re-keyed. Go to step 3.

Step 2.2. New nodes become leafs in CAG. Since these nodes have no other descendants than the one that is excluded they cannot be re-keyed. In figure 1.a, when removing node 14 (assuming that nodes 15 and 16 were removed earlier) nodes 11 and 13 will become leafs. In this case each new leaf in CAG must be included in the LKT and re-keyed. Two subcases follow:

Step 2.2.1. The LKT has enough free slots (unused key leafs) to accommodate the new leafs. Go to step 2.3.

Step 2.2.2. The LKT doesn't have enough free slots. In this case LKT must be expanded. LKT grows with one or more levels such that it accommodates all the newcomers. To grow the LKT with one level, for instance, the controller generates a new root key SK^{new} for the LKT. The old LKT will be the left sub-tree starting from the new SK^{new}. That is, the SK becomes the $K_{0,2^{d-1}}$, where d is the depth of the initial LKT. The SK^{new} is conveyed to the old members of the tree by encrypting it with the old SK and multicasting to the group. Within the same message the old members are informed that the tree increased with one level and the indexes of their keys changed, such that

$K_{2^{d-i}*j.2^{d-i}*(j+1)-1}$ becomes $K_{2^{d-i+1}*j.2^{d-i+1}*(j+1)-1}$. The whole new right sub-tree of the LKT will then be available to newcomers. Observe that LKT can grow more than one level at once making this process more efficient. Go to step 2.3.

Step 2.3. Re-key and include each new leaf Y into the LKT by sending the new session key SK^{new} as well as the auxiliary keys associated with LKT encrypted with its key K_Y. Y and all its ancestors will understand this message so the key propagation is re-established.

Step 3. The algorithm ends.

The number of transmissions used to exclude a category is:

$$2*\log(\text{card}(\text{leafs}_{CAG})) - 1 + \text{newleafs}(sn) + 1 \qquad (2)$$

where *newleafs* is a function, denoting the number of new leafs in CAG at session number *sn*. The last term 1 comes from the worst case assumption that any application of the exclusion algorithm will require expansion of the LKT.

The number of keys per user varies from $\log(\text{card}(\text{leafs}_{CAG})) + 1$ for the leafs in the CAG to $M + \text{card}(\text{leafs}_{CAG}) - 1$ for the root of CAG. The total number of keys is $M + \text{card}(\text{leafs}_{CAG}) - 1$ which is also the number of keys per controller. The solution has good performance for transmissions, but the number of keys per user as well as the number of keys per controller increases due to the propagation of keys.

For a given number of categories M the number of transmissions and the number of keys per controller depends on the number of leafs in CAG. Fewer leafs result in better numbers. On the other hand the number of keys per user depends not only on the number of nodes M and the number of leafs $\text{card}(\text{leafs}_{CAG})$ but also on the mapping between nodes from leafs_{CAG} and the slots in LKT. According to equation 1, the set of keys depends on the union of keys propagating from its descendants, and therefore it has fewer elements in the case where the propagated keys are the same.

5. EVALUATION

We analytically compare the efficiency of our solution strategy with our reference schemes and then present experimental results obtained through simulation.

5.1 Comparison

The comparison is given in Table 1. We observe that the performance of the exclusion algorithm is dependent on the term newleafs(sn), see equation (2). We note that for a given number of categories M each category will sooner or later become a leaf in CAG. Also there will be at least as many sessions as M. There-

fore, on average newleafs(sn) ≤ 1. We also note that card(leafs$_{CAG}$) ≤ M and that M = N denotes the worst case where every category has exactly one user.

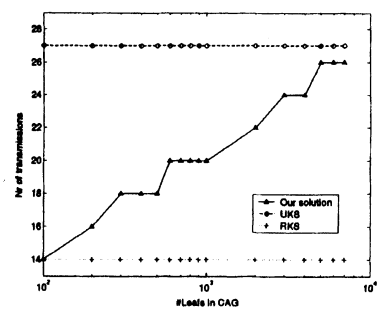
Table 1. Comparison with our reference schemes

		Our scheme	RKS	UKS
Transmissions to exclude	one user	NA	logN	2*logN – 1
	one category	2*log(card(leafs$_{CAG}$)) + newleafs(sn)	NA	NA
Keys/user		log(card(leafs$_{CAG}$)) + 1 .. card(leafs$_{CAG}$) + M –1	logN + 1	logN + 1
Keys/controller		Card(leafs$_{CAG}$) + M – 1	2*logN + 1	2*N – 1
Collusion resistance		Collusion requirement is configurable	Collusion can occur uncontrolled	Perfect resistance

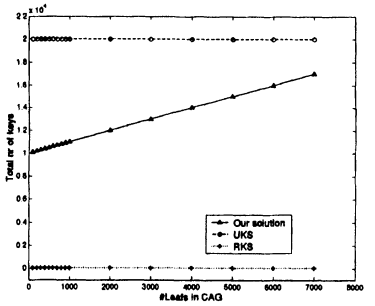
NA = not applicable

5.2 Experimental results

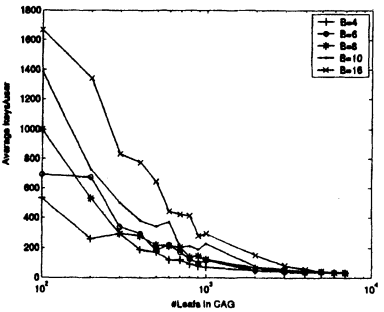
We evaluate the performance of our proposed solution by simulating different configurations of the CAG in the case where the CAG is a tree. We motivate choosing a tree for CAG by the following facts: the tree is a worst case with respect to memory needed for the LKT; further, it is also a worst case with respect to number of keys per user; finally, there already exist random tree generators that have been used in benchmarking multicast protocols.



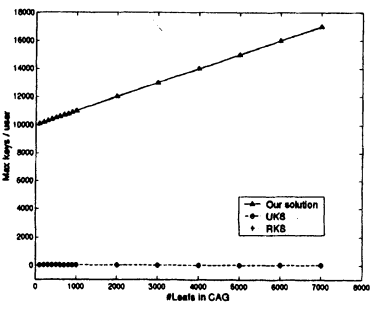
a) Number of transmissions



b) Number of keys/controller



c) Average number of keys/user



d) Maximum number of keys/user

Figure 2. Experimental results

A specific configuration of the CAG models a certain collusion requirement. We randomly generate such configurations by using a random tree generator with three degrees of freedom [14] - the maximum number of nodes, the maximum number of leafs and the maximum branching factor. In the experiments, the number of categories (nodes in CAG) has been kept to 10000, and the number of leafs has been varied from 100 to 7000. Each combination of nodes and leafs has been repeated with a branching factor of 4, 6, 8, 10, and 16.

As shown by the experiments, and as expected from equation (2), the number of transmissions depends solely on the number of leafs, see figure 2.a. Fewer leafs result in fewer transmissions, more leafs in more transmissions. Since the number of leafs is related to the degree of collusion, this result confirms our hypothesis – the more constraints are put on collusion the more transmissions we need. Our solution gives the possibility to balance the collusion requirement and the number of transmissions. The number of keys per controller, figure 2.b, is less than in the UKS, but it increases with the number of leafs in the CAG. Actually the number of transmissions and the number of keys per controller from the UKS are upper bounds for the number of transmissions and the number of keys per controller in our scheme.

Because the keys from child nodes are given to their parents, the fewer leafs we have the more keys per user we get, as shown in figure 2.c (on average) and figure 2.d. Note that we have computed the average number of keys per user in the conditions of only one user per category. If there are more than one user per category the distribution of keys per user can change very much and can greatly influence the average. If categories towards the root have many users, the average would become high. If on the contrary most of the users are concentrated in categories towards the leafs the average goes down.

6. RELATED WORK

An overview of various issues regarding multicast security is given in [1]. The topic of key management for multicast communication has been studied before. The GKMP [9] introduces a centralized controller that distributes session keys by unicasts to each group member. This approach does not scale well for large groups and frequent member changes. To provide scalability, Iouls [12] decentralizes the controller by using a hierarchy of trusted key distribution servers.

The scalability problem associated with frequent key change in large groups has been addressed in [15, 16, 17] by using logical key hierarchy (LKH) of auxiliary keys that balance the number of transmissions and the key storage. OFT [11] halves the communication for re-keying by introducing dependencies in the logical key structure that allows keys to be derived from other certain keys. More precisely, in OFT the key of one node is derived from the two

keys of its child nodes. OFC [3] applies a similar principle to LKH, but only when re-keying, and halves its communication. In OFC, there exists a path from one of the leafs to the root of the tree such that the session key corresponding to the root can be obtained by iterative hashing of that key leaf. The hybrid scheme [2] reduces the storage requirement for the controller at the price of increasing communication and computation overhead. [10] finds the optimal hybrid scheme when constraints on communication and storage are given. More recently, ELK [13] allows trading communication overhead with security and computation. The idea is to avoid the transmission of the whole key when re-keying, by sending only smaller hints of the updates that would enable an authorized receiver to compute the key, partially by brute search.

Logical tree structures with reuse of keys, such as the BFM [4] and Flat [15] improve the efficiency by decreasing the number of keys stored by the controller as well as reducing the number of transmissions for cumulative member removal [4]. However, these schemes are susceptible to collusion. To reduce the probability of collusion the number of leafs of the tree is increased. If collusion still occurs the only way to successfully exclude the colluding group is to collectively remove the corrupted users. The algorithm proposed in [15] for such case suffers from the fact that it also expunges innocent users who happen to share keys with the colluding users. By using the scheme proposed by us the possible colluding users can be securely excluded by expunging the category to which they belong.

More close to our work is the family of broadcast encryption schemes from [6] that allows for flexibility with respect to the collusion requirements as defined there. The schemes allow a controller to broadcast a secret to any set of privileged users so that coalitions of k users not in the privileged set cannot learn the secret. Therefore, the collusion requirement is specified by the parameter k that is the maximum size of a corrupt coalition to which the scheme is still resistant. However, their work has mostly a theoretical value since the overhead of such scheme is very large. Moreover, they use a static distribution scheme with predefined slots.

7. CONCLUSIONS AND FUTURE WORK

This paper adds the collusion constraints as a requirement for designing key management systems. A formalization of the collusion requirement is given in terms of categories of users and constraints among these categories. A solution is proposed for the general case of collusion requirement. A scalability analysis is given in terms of the number of the transmissions for re-keying and the storage requirement (number of keys). We find that in general there is a tight dependency between the exact settings of the collusion requirement shown in the form of CAG and the performance of the solution. In general the solution has good performance for the number of

transmissions needed to re-key in case a category must be excluded. However, the number of keys per user may become large due to the propagation principle.

Future work can be invested in designing smarter key management schemes that bring the best balancing between scalability and security, including collusion as a given requirement. We are currently investigating [5] how to alleviate the explosion of keys due to propagation by using hash dependencies among the keys similar to [3, 11]. This would minimize the storage requirement. However, the hash function would probably amend the computation resources of the end users.

REFERENCES

1. R. Canetti, B. Pinkas, "A Taxonomy of Multicast Security Issues", Internet Draft, 1998.
2. R. Canetti, T. Malkin, K. Nissim, "Efficient Communication-Storage Tradeoffs for Multicast Encryption", *Eurocrypt'99*, 1999.
3. R. Canetti, J. Garey, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, "Multicast security: A taxonomy and efficient constructions", *Infocom '99*, March 1999.
4. I. Chang, R. Engel, D. Kandlur, D. Pendarakis, D. Saha, "Key Management for Secure Internet Multicast using Boolean Function Minimization Techniques", *IEEE Infocom '99*, March 1999.
5. C. Duma, N. Shahmehri, P. Lambrix, "Efficient Storage Requirement for Category-Based Key Management for Multicast", Internal Technical Report, Linköpings universitetet, Sweden, November 2002.
6. A. Fiat, M. Naor, "Broadcast encryption", *Crypto '92*, Springer-Verlag LNCS 839, 1994, pp. 257-270.
7. L. Han, N. Shahmehri, "Secure Multicast Software Delivery", *9th IEEE International Workshop on Enterprise Security (WET-ICE)*, June 2000.
8. L. Han, *Secure and Scalable E-Service Software Delivery*, Licentiate Thesis No. 906, Linköpings universitetet, Sweden, September 2001.
9. H. Harney, C. Muckenhirn, "Group Key Management Protocol (GKMP)", RFC 2093 and RFC 2094, July 1997.
10. M. Y. Li, R. Poovendran, C. Bernstain, "Optimisation of Key Storage for Secure Multicast", *Conference on Information Science and Systems 2001*, March 2001.
11. D. A. McGrew, A. T. Sherman, "Key Establishment in Large Dynamic Groups Using One-Way Function Trees", Technical Report No. 0755, TIS Labs at Network Associates, May 1998.
12. S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting", *ACM SIGCOMM'97*, 1997.
13. A. Peering, D. Song, J. D. Tyger, "ELK, a New Protocol for Efficient Large-Group Key Distribution", *IEEE Security and Privacy Symposium 2001*, May 2001.
14. S. Ratnasamy, S. McCanne, "Inference of Multicast Routing Trees and Bottleneck Bandwidths using End-to-End Measurements", *Infocom '99*, March 1999.
15. M. Waldvogel, G. Caronni, D. Sun, N. Weiler, B Plattner, "The VersaKey Framework: Versatile Group Key Management", *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 8, August 1999.
16. D. Wallner, E. Harder, R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, June 1999.
17. C. K. Wong, M. Gouda, S. S. Lam, "Secure Group Communications Using Key Graphs", *ACM SIGCOMM '98*, 1998.
18. Y. R. Yang, S. S. Lam, "A Secure Group Key Management Protocol Communication Lower Bound", The University of Texas at Austin, Department of Computer Sciences, Technical Report TR-00-24, September 2000.