Optimization of Key Storage for Secure Multicast

Mingyan Li and R. Poovendran
Department of Electrical Engineering
University of Washington
Seattle, WA 98195, USA
e-mail:radha@ee.washington.edu

C. Berenstein
Institute of System Research
and Department of Mathematics
University of Maryland, College Park, MD 20742, USA

Abstract — **This paper addresses the problem of distributing cryptographic keys to a secure multicast group with single sender and multiple receivers. The group is assumed to be dynamic and hence the members are allowed to join and leave during the session. We show that the problem of distributing keys to a dynamic group can be formulated as a constraint optimization problem. From the formulation, we also show how to minimize the amount of keys to be stored by the sender.**

Keywords: Security, Optimization, Multicast Communications

I. Introduction

Recent progress in communications, networking and multimedia related research has led to new standards and applications. This convergence of three areas has also created potential for new applications involving group communications. Many of these applications have a single sender or service provider who transmits identical data to multiple end users. For such applications, use of multicast communication model will reduce the sender and the network overhead. However, the success of these commercial models depends on the ability of the sender in securing the communications so that only the intended receivers have access to the data. The use of cryptography is one approach to control access to communications [2].

Since identical data is delivered to all the group members, the owner of the group or the Group Controller (GC) can minimize the computation by using a single cryptographic key and symmetric key encryption to encrypt and transmit the data [1]. Every intended receiver is given access to the Session Encryption Key (SEK). This key is also called Traffic Encryption Key (TEK). Anyone possessing the SEK used during the session will have access to the group communications.

The SEK has to be updated whenever its intended lifetime expires and also whenever there is a change in group membership. When a new member is added to the group, the SEK is changed to ensure that a new member does not have access to the past encrypted data. When a member is deleted, the SEK is changed to ensure that the deleted member does not have access to the future data.

The problem of SEK management is to ensure that only the valid members have access to it during the session.

Since the SEK is known to every valid member of the group, when SEK needs to be updated, there needs to be additional keys called Key Encrypting Keys (KEKs) to distribute the new SEK [2, 7, 8]. [1] Hence, the problem of securely distributing SEK is reduced to the problem of distributing the KEKs to the members such that only the valid members have access to the KEKs and the SEK.

Since the KEKs are used to update the SEK, assignment of KEK to the group members should be (a) scalable in key update communication and/or key storage requirements of sender and receivers with respect to group size, (b) resistant to illegal collaboration (*collusion*) and (c) able to guarantee that deletion of one or more members does not compromise *all* the keys of valid members.

Wallner *et al.* [7] and Wong *et al.* [8] proposed a scalable solution based on rooted trees that requires every user of a group of size N to store $\log N$ number of keys. Their scheme also requires $\mathcal{O}(\log N)$ key update communications when a member is deleted. However, the GC has to store $\mathcal{O}(N)$ number of keys in their model. Although several variations of the tree based schemes have been presented in the literature [2, 6], none of them allow the designer to choose the amount of communications and compute the model parameters. In [2] an approach to do this was attempted with asymptotic analysis.

In this paper, we present a formulation of the key distribution that considers the tradeoff between the amount of key update communications and the key storage requirements of GC. We show that this problem can be posed as a constraint optimization problem. We then convert this problem to a fixed point equation and show how to solve for the optimal design parameters.

The paper is organized as follows: Section II presents the basic tree based key distribution schemes [7, 8] and a variation of this model, which is of interest to us [2]. Section III shows our formulation of the problem and the derivation of the optimal cluster size M, and the required proof of optimality of the cluster size. Section V presents numerical illustrations. Contributions of the paper are discussed in Section VI.

II. Virtual Tree Based Key Distribution Protocols

The tree based KEK distribution approach [7, 8] builds a logical tree of a given degree for a given group. The Figure 1 illustrates a rooted binary key distribution tree for a group with eight members.

The logical tree is constructed such that each group member is assigned to a unique leaf node of the tree. Every node of the logical tree is assigned a KEK. The set of keys assigned to the nodes along the path from a leaf node to the root are assigned to the member associated with that particular leaf node. For example, member M_1 in Figure 1 is assigned key encrypting keys $\{K_0, K_{1,1}, K_{2,1}, K_{3,1}\}$. Since the root key K_0 is also shared by all the members, if there is no change in group membership, K_0 can be used to update the SEK for all the members.

[1] The KEKs are used for encryption and transportation of the cryptographic keys including the SEK.

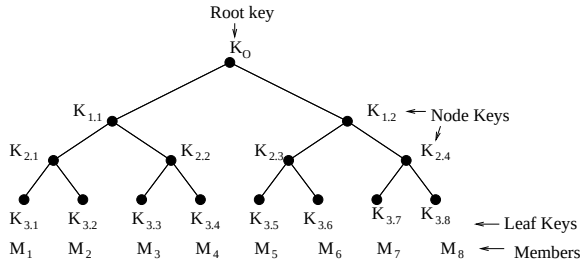

Figure 1: The Logical or Virtual Key Tree of [7, 8].

The tree based structure also induces a natural hierarchical grouping among the members. By assigning the members to appropriate nodes, the group controller can form desired hierarchical clusters of members and selectively update, if needed, the keys of the group. For example, in Figure 1, members M_5, M_6, M_7, and M_8 exclusively share the key $K_{1.2}$. The GC can use the key $K_{1.2}$ to selectively communicate with members M_5, M_6, M_7, and M_8. Such clustering of the members on the tree may be decided by the GC based on application specific needs. In order to be able to selectively disseminate information to a subset of group members, the GC has to ensure that the common key assigned to a subset is not assigned to any member not belonging to that subset.

Using the notation $\{m\}_K$ to denote the encryption of message m with key K, and the notation

$$\mathrm{A} \longrightarrow \mathrm{B} : \{m\}_K$$

to denote the secure exchange of message m from A to B, GC can selectively send a message m to members $M_5, \cdots M_8$ by the following transmission:

$\mathrm{GC} \longrightarrow M_5, M_6, M_7, M_8 : \{m\}_{K_{1.2}}$

If, however the key $K_{1.2}$ is invalidated for any reason, GC needs to update the key $K_{1.2}$ before being able to use a common key for members M_5, M_6, M_7, and M_8. It can do so by first generating a new version of $K_{1.2}$, denoted $\hat{K}_{1.2}$, and then performing two encryptions, one with $K_{2.3}$ and the other with $K_{2.4}$. The following two messages are needed to update key $\hat{K}_{1.2}$ to the relevant members of the group.

$\mathrm{GC} \longrightarrow M_5, M_6 : \{\hat{K}_{1.2}\}_{K_{2.3}}$

$\mathrm{GC} \longrightarrow M_7, M_8 : \{\hat{K}_{1.2}\}_{K_{2.4}}$

A Member Deletion on Trees

Since the session key and the root key encrypting key K_O are common to all the members in the group, they have to be invalidated each time a member is deleted. Apart from these two keys, all the intermediate key encrypting keys assigned to the deleted member need to be invalidated. In the event there is bulk member deletion, the GC has to (a) identify *all* the invalid keys, (b) find the minimal number of valid keys that need to be used to encrypt and transmit the updated keys, and (c) update the valid members with the new keys.

The general principle behind the member deletion is discussed below using member M_1 as example. Member M_1 in Figure 1 is indexed by the set of four keys $\{K_O, K_{1.1}, K_{2.1}, K_{3.1}\}$. Deleting member M_1 leads to invalidating these four keys and the session key, generating new keys, and updating these keys of the appropriate valid members who shared the invalidated keys with member M_1. When M_1 is deleted, the following updates are necessary: (a) all member need new root key K_O and new session key SK, (b) members $M_2 - M_4$ need to update $\{K_{1.1}\}$, (c) members $M_3 - M_4$ need to update $\{K_{2.2}\}$, and (d) member M_2 needs to update $\{K_{2.1}\}$.

Hence, the user storage as well as the update communications scale as $\mathcal{O}(\log N)$ for the tree based schemes in [7, 8]. The group controller has to store *all* the keys corresponding to the nodes of the entire tree. It can be shown that for any rooted tree, the storage requirement of the group controller is $\mathcal{O}(N)$[2]. Hence, the key storage requirements of the GC is a bottleneck in this model. For many applications with limited storage, it is desirable to reduce the group controller storage requirements as well. We now present a variation of the tree based model presented in [2] and show how to use that model to reduce key storage requirement of GC.

B Model for Minimizing the Key Storage of the Group Controller

In the tree scheme discussed earlier, every leaf node is assigned a unique member. Hence, the number of leaves in the tree is N. Once this parameter is fixed, the depth of the tree and hence the total number of nodes in the tree are fixed. Since every node of the tree is assigned a KEK, the number of keys are fixed. If there is a mechanism to group members and assigning multiple members to a leaf, then by controlling the number of members assigned to a leaf node, we can vary the total number of nodes in the tree.

Let a be the degree of the tree. If the group is divided into clusters of size M with every cluster assigned to a unique leaf node, then we will have $\lceil \frac{N}{M} \rceil$ number of clusters, then we need to build a tree of depth $\log_a \lceil \frac{N}{M} \rceil$. The Figure 2 illustrates this for a binary tree with cluster size M.

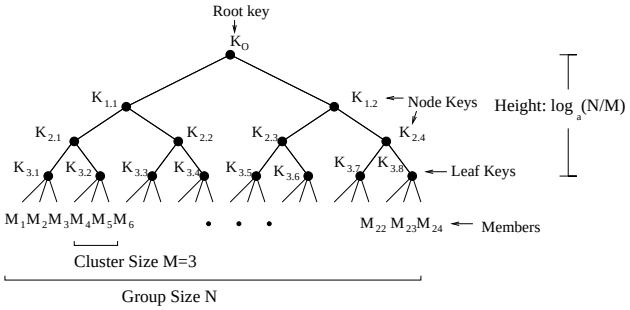

Figure 2: Logical Key Tree with Clusters

In order to do this, the following questions need to be addressed:

- How to manage the keys within each of the clusters?
- How to find the optimal cluster size so that the update communications will be within a factor of the original tree scheme in [7]?

C Managing Keys within a Cluster

Within each cluster with M members, all the members are assigned a cluster KEK, which is also called *common cluster key*. The common cluster key is used to update SEK within a cluster with a single encryption and decryption. Every member of the cluster is also assigned a unique key K_u which is

[2] Exact value of the storage for an $a-ary$ tree is $\frac{aN-1}{a-1}$ and scales as N.

shared only with the GC. The GC uses a random seed r as an index for pseudorandom function [2] f_r to generate the key K_u for member u as $K_u = f_r(u)$. Hence, for each cluster, only the seed and the cluster KEK need to be stored. When a member leaves a cluster, a new cluster KEK is generated and encrypted with the individual KEKs. When a member is deleted within the cluster, the group controller has to perform $(M-1)$ individual encryptions within the cluster alone to update the common cluster key.

D Update Communication of the Group

Since each cluster is assigned a unique leaf node, the set of keys assigned to a cluster is the set of keys along the path from the root to the leaf node. Since there are $\log_a \lceil \frac{N}{M} \rceil$ number of nodes along the path from the root to a leaf node and each node is assigned a KEK, there are $\log_a \lceil \frac{N}{M} \rceil$ KEKs that are assigned to a cluster. When a member is deleted, the total number of key update messages is $M-1$ within the cluster plus $(a-1)\log_a \frac{N}{M}$ within the group, leading to $M-1+(a-1)\log_a \frac{N}{M}$ update messages. That is, for an $a-ary$ tree-based key distribution, when a single member is deleted, the total number of update communications C is given as:

$$C = M - 1 + (a-1)\log_a \frac{N}{M} \tag{1}$$

III. Minimization of Key Storage with Communication Constraint

The number of keys to be stored by the GC is computed as the keys on the tree plus seeds for $\frac{N}{M}$ clusters, leading to

$$S = \sum_{i=0}^{\log_a \frac{N}{M}} a^i + \frac{N}{M} = \left(1 + \frac{a}{a-1}\right)\frac{N}{M} - \frac{1}{a-1} \tag{2}$$

Since the last term $\frac{1}{a-1}$ is at most 1, it can be ignored with estimate error of S within 1 key.

The storage and the update communications are functions of the cluster size M which needs to be optimized. Since the tree based schemes in [6, 7, 8] have logarithmic update communications, in the cluster based model, we would like to keep the update communications as logarithmic of the group size as well. Hence the optimization problem is posed as

$$\min\left(1 + \frac{a}{a-1}\right)\frac{N}{M} \quad \text{w.r.t.} \quad M \tag{3}$$

subject to

$$M - 1 + (a-1)\log_a \frac{N}{M} \leq \beta \log_a N, \tag{4}$$

where $\beta \geq 1$.

The storage can be noted as a monotonically decreasing function of the cluster size M. The following theorem summarizes the result.

Theorem 1: Optimal cluster size M that minimizes the storage function $S = \frac{(2a-1)N}{(a-1)M}$ while satisfying $C = M - 1 + (a-1)\log_a \frac{N}{M} \leq \beta \log N$ is obtained by the largest root of the equation $M - \lambda \log_e M = \mu$, where $\lambda = \frac{(a-1)}{\log_e a}$ and $\mu > \lambda(1 - \log_e \lambda)$.

Proof: Since the storage decreases monotonically with respect to the cluster size M, it is sufficient to find the values of M that satisfies the update communication constraint. Moreover, since the storage is a monotonically decreasing function, the largest value of M satisfying the update communication constraint will be the solution of this constraint optimization. Hence, the optimal value of the cluster size is computed by the equation:

$$M - \lambda \log_e M + \lambda \log_e N - 1 = \beta \log_e N \tag{5}$$

This can be rewritten as

$$\begin{aligned} M - \lambda \log_e M &= 1 + (\beta - \lambda)\log_e N \qquad (6)\\ \Rightarrow M - \lambda \log_e M &= \mu \end{aligned}$$

where $\mu = 1 + (\beta - \lambda)\log_e N$.

A Computing Cluster Size M

Since the function $M - \lambda \log_e M + \lambda \log_e N - 1$ is convex with the minimum value $\lambda(1 + \log_e \frac{N}{\lambda}) - 1$, if $\beta > \frac{\lambda(1+\log_e \frac{N}{\lambda})-1}{\log_e N}$, then the equation (5) has two solutions. Since $M - \lambda \log_e M$ has the minimum at $M = \lambda$ and $\lambda > 0$, the roots of the equation $M - \lambda \log_e M = \mu$ lie below and above $M = \lambda$. Moreover, if $M > \lambda$, the gradient of $M - \lambda \log_e M$ is $1 - \frac{\lambda}{M} > 0$. Hence the fixed point equation

$$M - \lambda \log_e M = \mu \tag{7}$$

is a contraction mapping with the largest root as the fixed point solution if we start the iteration with an initial value $M_0 > \lambda$. Since $\lambda, \mu > 0$, $M = (\mu + \lambda \log_e M) > \mu$, if we set the initial value of M to be $M_0 = \mu$, after some algebra, a series approximation to M is given by

$$M = \mu \prod_{i=1}^{\infty} (1 + \left(\frac{\lambda}{\mu}\right)^i \log_e \mu), \tag{8}$$

Since $\lambda > 0$ is fixed, and $\log_e \mu < \mu$ as $\mu \to \infty$, if we denote the asymptotic value of M by M_∞, the limiting value is given by

$$\begin{aligned} M_\infty &= \lim_{\mu \to \infty} \mu \prod_{i=1}^{\infty} (1 + \left(\frac{\lambda}{\mu}\right)^i \log_e \mu) \qquad (9)\\ &= \mu + \lambda \log_e \mu \\ &\approx \mu \qquad (10) \end{aligned}$$

But $M_0 = \mu \approx (\beta - \lambda)\log_e N$. Hence, the asymptotic value of the largest root of the equation $M - \lambda \log_e M = \mu$ is $M_\infty = (\beta - \lambda)\log_e N$.

We now show that the same results can be derived using the first order Taylor series approximation with Newton's method. Setting the first approximate solution to the equation $M - \lambda \log_e M = \mu$ for $M > \lambda$ as $M_0 = \mu$, the first approximation is $M_1 = \mu + \lambda \log_e \mu$. This is indeed the asymptotic solution we obtained using the fixed point iterations earlier. Letting $N \to \infty$ leads to $M_\infty \to \mu + \log_e \mu \approx \mu$. It can be shown that even if the series is computed for higher order terms, for large values of N, the largest root M of the equation $M - \lambda \log_e M = \mu$ converges to $M_\infty = \mu + \lambda \log_e \mu$ and grows as $\mathcal{O}(\log N)$.

B Computing Minimal Storage

We showed that the asymptotic value of the largest root of the equation $M - \lambda \log_e M = (\beta - \lambda)\log_e N$ is $M = \mu =$

$(\beta - \lambda)\log_e N$ by two different approaches. The corresponding value of the storage denoted S_∞ is

$$\lim_{N\to\infty} S_N = \frac{(2a-1)N}{(a-1)M_\infty}$$
$$S_\infty = \frac{2a-1}{a-1}\frac{N}{(\beta-\lambda)\log_e N} \quad (11)$$

Hence, the constraint optimization leads to optimal growth of storage as $\mathcal{O}(\frac{N}{\log N})$ when the update communication is constrained to grow as $\mathcal{O}(\log N)$.

We now formally state it as the proof of *Theorem 1.*

Proof of Theorem 1: The storage $S = \frac{(2a-1)N}{(a-1)M}$ is a monotonically decreasing function of M. We showed that the minimal storage is obtained by the *largest cluster size* M that satisfies the constraint

$$M - 1 + (a-1)\log_a \frac{N}{M} = \beta \log_a N \text{ for } \beta \geq 1$$

can be converted to the form $M - \lambda \log_e M = \mu$. The asymptotic value of the largest root of this equation is shown as $M_\infty = (\beta - \lambda)\log_e N$. Hence the asymptotic optimal storage under communication constraint is $S = \frac{(2a-1)N}{(a-1)(\beta-\lambda)\log_e N}$.

We note that the optimal cluster size M, and hence the optimal storage are functions of the parameter β. From (11), it is seen that β can be used as a design parameter to control the key storage at the GC.

IV. Related Work

Many schemes have been proposed for key mangement problems. One early work along this line can be found in [3]. In [4], each member share a KEK with GC, and there is a group KEK shared by all group members. Under member deletion, GC has to individually contact every member to update the SEK and the group KEK. Therefore, the communications is linear in goup size. Virtual tree based distribution scheme is independenly proposed by Wallner *et al* in [7] and Wong *et al* in [8]. A rooted binary tree was used in [7] and key grphs were used in [8]. The storage of GC is $\mathcal{O}(N)$ for virtual key tree. In [2], a hybrid tree scheme is proposed. Group members are divided into clusters, by combining virtual tree and clusters, the storage of GC is reduced to sublinear in group size. The tree schemes are analyzed by using entropy in [6], and thus prove minimum communication for a tree based scheme is $\mathcal{O}(\log N)$. A review of current research in secure multicast is presented in [1]. A protocol called ELK for key distribution is proposed in [5]. ELK provides reliability of key update messages without relying on reliable multicast protocols.

V. Numerical Comparisons

In this section, we present numerical illustration to show that the optimal cluster size M_∞ can lead to significant improvements in storage over values in [7]. For simplicity, we set $K - \lambda = 1$ which leads to $M_\infty = \log_e N$ and $S_\infty = \frac{(2a-1)N}{(a-1)\log_e N}$. Table 1 presents the comparison in storage and communication for several pairs of (degree a, group size N). Compared to the increase in update communications, the reduction in storage of GC is much more significant.

VI. Conclusions

(Degree a, Group size N)	# of keys in GC by virtual tree	# of keys in GC as Eq (11)	Storage reduction in %	Update Comm.[3] increase in %
$(2, 2^{10})$	2047	444	78.3	31.4
$(2, 2^{20})$	2097151	226920	89.2	45.4
$(3, 2^{10})$	1535	370	75.9	19.0
$(3, 2^{32})$	6.44×10^9	4.84×10^8	92.5	38.5
$(4, 2^{10})$	1365	345	74.7	11.6
$(4, 2^{20})$	1398101	176490	87.4	23.9

Table 1: Comparison of Storage and Key Update Communication.

In this paper, we showed that the key storage requirement in [7] can be reduced from $\mathcal{O}(N)$ to $\mathcal{O}(\frac{N}{\log N})$ by formulating it as a constraint optimization problem. We also gave a proof of optimality. Our approach can be viewed as an explicit design methodology.

Acknowledgments

References

[1] R. Canetti, and B. Pinkas, "A Taxonomy of Multicast Security Issues", *Internet draft*, April, 1999.

[2] R. Canetti, T. Malkin, and K. Nissim, "Efficient Communication-Storage Tradeoffs for Multicast Encryption", *Eurocrypt'99*, pp. 456 - 470.

[3] A. Fiat and M. Naor, "Broadcast Encryption", *Advances in Cryptology-CRYPTO'92*, vol. 773, pp.481-491, 1993.

[4] H. Harney, C. Muckenhirn, "Group Key Management Protocol Architecture", *RFC 2094*, July 1997

[5] A. Perrig, D. Song, and J.D. Tygar, "ELK, a New Protocol for Efficient Large-Group Key Distribution", *IEEE Proceedings of Security and Privacy*, May 2001, Oakland, California

[6] R. Poovendran, J. S. Baras, "An Information Theoretic Approach for Design and Analysis of Rooted Tree-Based Multicast Key Management Schemes", *Advances in Cryptology-CRYPTO'99*, pp.624-638, 1999.

[7] D. M. Wallner, E. C. Harder, and R. C. Agee, "Key Management for Multicast: Issues and Architectures", *Internet Draft*, September 1998.

[8] C. K. Wong, M. Gouda, S. S. Lam, "Secure Group Communications Using Key Graphs", *IEEE/ACM Trans. on Networking*, Vol.8, No.1, pp.16-31, Feb. 2000.