PH.D. THESIS

Key Management for Secure Multicast Communications

by R. Poovendran
Advisor: J.S. Baras

CSHCN Ph.D. 99-2
(ISR Ph.D. 99-4)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

Sponsored by NASA, DoD, Army Federal Research Lab under ATIRP

ABSTRACT

Title of Dissertation: KEY MANAGEMENT FOR

SECURE MULTICAST COMMUNICATIONS

Raadhakrishnan Poovendran, Doctor of Philosophy, 1999

Dissertation directed by: Professor John S. Baras
Department of Electrical and Computer Engineering

Providing key management schemes for large scale multicast groups has become an important problem due to many potential commercial applications such as stock quote and software distribution on the Internet. For secure multicast communication, all the group members have to share a common session key. Since the member dynamics such as *join* or *deletion* do not necessarily terminate the multicast session, it is important to update the session key to all the valid members, so that the non-members do not have access to the future keys. Finding efficient ways for key generation and distribution in the presence of member dynamics is an actively researched problem.

This dissertation considers the single sender, multiple receiver model of secure multicast communication. The goal is to develop schemes that have reduced

computational overhead at the time of key generation, minimize the amount of message units required at the time of key updates, and minimize the number of keys to be stored by the sender as well as receivers. In order to achieve this goal, a key generation and distribution architecture based on rooted trees and control panels is proposed. A control panel is assumed to consist of mutually suspicious members who jointly generate the keys that are distributed to the rest of the members. Based on the assumption about the control panel, we provide a distributed key generation mechanism which allows a set of mutually suspicious members to contribute to the generation of a joint secret without revealing their individual contributions.

The key distribution scheme presented considers the *member revocation event* and relates it to the key assignment of individual users. We define and show that the *entropy of the member revocation event* plays an important role in determining the number of keys assigned to a member. We claim that the number of keys allocated to a member based on the elementary concepts from information theory will also correspond to the minimum number of keys that need to be assigned to a member unless additional functional relationship among keys exists, since it *completely captures* the uncertainty of the member revocation event. We also identify some weaknesses in the recent schemes in [17, 15], and solve an open problem posed at Eurocrypt'99 [16].

KEY MANAGEMENT FOR SECURE MULTICAST COMMUNICATIONS

by

Raadhakrishnan Poovendran

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1999

Advisory Committee:

Professor John S. Baras, Chairman
Professor Carlos Berenstein
Professor Anthony Ephremides
Professor Virgil D. Gligor
Professor Lawrance Washington

©Copyright by

Raadhakrishnan Poovendran

1999

DEDICATION

To my parents and teachers

ACKNOWLEDGEMENTS

I am deeply grateful to professor J. S. Baras for the technical, moral and financial support that he provided throughout my doctoral study. He has given me a unique intellectual environment and vision within which I have been able to develop this thesis. I am also very grateful to my dissertation committee - professors C. Berenstein, A. Ephremides, V. D. Gligor, and L. Washington - for their involvement and feedback. My work relating the key distribution and information theory is due to suggestions from professor Ephremides. Professor Gligor's Network Security course exposed me to the area of Network Security. Among the non-committee members, I would like to acknowledge the influence of professor Prakash Narayan's teaching on my research work. I would also like to acknowledge my discussions with Dr. M. S. Corson from Institute for Systems Research, and Dr. Eric Harder from the DoD. I also thank Professor Jim Massey for providing needed papers and other relevant ETH publications.

I would like to thank S. Khudanpur, M. Karir, V. Bharadwaj, A. Arora, H. Khurana, D. Lee, H. Lo, J. Thomas, S. Goel, R. Annamalai, S. Ahmed, A. Newman, T. Kuga, Praveen, and H. Rais. I am thankful to Tina Vigil, Diane Hicks, Suzanne M. Kirchhoff, and Towanda Jones for all their technical help. Tina has taken care of many details and saved me from the "system" time and again. I am also grateful to my high school teacher Ratnasabapathy for his help at a time when there was none.

Several agencies generously provided financial support for this work. In particular, the work was supported in part by NASA contracts under the cooperative agreement NASA-NCC5227, Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0002, DRIF 01-3-93607, Development of Networking Technology for a Prototype contract no. 19YFJH99V, and Development of a Robust Scalable Multicast Security contract no. CG9813. I am also thankful to NASA for allowing me to use their library facilities for my research work. I feel that the National Security Agency was generous in providing me with their first Rising Star Award of LUCITE.

I would like to thank Drs. John E. Dorband, Jan. M. Hollis and Laveen Kanal from my prior work environment. I would like to acknowledge the generous support of Scott Speigle from ARMY Missile Command, Huntsville, Alabama on landmark navigation project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Internet applications, such as online games, newscast, stock quotes, multiparty conferences, and military communications, can benefit from secure multicast communications. In most of these applications, users typically receive identical information from a single or multiple senders. Hence, grouping these users into a single multicast group and providing a common session encryption key to all of them will reduce the number of message units to be encrypted by the senders. Securing group communications or computations leads to challenging problems such as maintaining communication integrity in the presence of group membership changes, establishing source authentication, and minimizing key storage size and number of update messages at the senders as well as the receivers.

This dissertation addresses the key generation and distribution problems associated with maintaining communication integrity in the presence of membership changes. We consider the single sender - multiple receiver model of the multicast communications. Source authentication is not addressed in this dissertation.

1.1 Key Distribution

An important problem in preserving *communication integrity* is that of controlling membership to the multicast group such that only the valid or the authorized, legitimate members can have access to group communications. In this context, secure communication and computation is achieved by providing a *common group key* to all the valid members for session encryption, and updating the key when there is a need. This common key is often called by various names such as *session key*, *traffic encrypting key* or the *net key* [10, 11].

The key distribution and update scheme should be able to prevent any set of members from collaborating or *colluding* and obtaining future keys or keys assigned to other members. In particular, revoked members of the group should not be able to collaborate and obtain the future keys of the multicast group.

The problem of controlling membership to the multicast group is reduced to the problem of maintaining the following *condition/invariant*: at any time all the valid group members and only the valid group members have access to the current session key with no possible collusion from non-members.

A modified version of the key assignment problem is to *find key distribution methods that satisfy the invariant and also require the minimum number of keys to be stored and updated with membership changes.*

This modified problem is addressed using information theory concepts.

1.2 Key Generation

Another important problem in preserving communication integrity is to find methods for a set of members to jointly generate keys *without* having to expose

their individual contributions to the generation of group keys.

Most of the currently available group key generation schemes fall into two categories. They either generate group public keys with the help of a trusted third party or are based on the generalization of the Diffie-Hellman (DH) technique for group keying. (The DH keys are common shared keys). The generalized Diffie-Hellman scheme, which has been extensively used in the recent group communication protocols, involves several exponentiations and the computations scale linearly as a function of the group size N. Both of these methods rely on assumed cryptographic hard problems - performing discrete logarithm or factoring an integer.

We present a key generation scheme that does not depend on the computational difficulties of the integer factoring or on discrete logarithm, but instead makes use of the concept of a one-time pad. We provide a key generation scheme that allows the key generating members to locally compute the one-time pads and use it to securely exchange their individual shares without exposing them. The padded shares are combined to generate the padded common secret. Our scheme also provides a method to compute the group parameter that is a combination of all the pads, and can be used to remove the *combined padding effect* and extract the common secret. If the key generation mechanism can provide uniformly distributed variables over an interval of interest, this scheme will be resistant to attacks from any individual member or up to $(N-2)$ collaborating members. An advantage of our scheme is that it can be used for generation of group shared keys as well as public keys.

1.3 Contributions

This dissertation makes the following technical contributions:

1. It defines a distributed key generation scheme that allows a set of members to contribute to the common key generation without exposing individual secrets. The scheme allows members to locally compute the one-time pads and a group binding parameter that removes the combined effects of the padding at the time of common secret construction. This has the advantage that the future one-time pads need not be pre-distributed and stored. The scheme can be used for generating group shared keys, or public keys with suitable modifications.

2. It explains how to use basic concepts of information theory to: (a) determine the optimal number of keys allocated to a member, (b) find the average key length that can be supported, (c) interpret the weaknesses of current schemes, and (d) determine optimal cluster size with maximum amount of uncertainty as to which cluster will be compromised next.

3. It defines a key management architecture that makes use of the key generation and distribution schemes mentioned above.

1.4 Dissertation Organization

The second chapter reviews the background work and identifies the requirements of the key management architecture from the viewpoint of key generation, distribution, and rekeying. The third chapter presents the new key generation scheme based on the requirements identified in the second chapter and its analysis. The

fourth chapter presents the notion of optimal schemes for key distribution. The fifth chapter presents a clustering strategy based on maximum uncertainty of cluster revocation to provide an optimal key allocation problem which was posed as an open problem at Eurocrypt'99. The last chapter presents some of the possible future research directions.

Chapter 2

Related Work

In this chapter, we present the key distribution and generation approaches proposed to date and analyze their properties. We define the factors that influence the design of a key management scheme and then identify desirable properties of a multicast key management scheme. This enables us to evaluate the strengths and weaknesses of the schemes discussed and the different optimality claims made by these schemes. Since the terminology used by these schemes varies, we introduce a common terminology and the associated notation to provide a basis for scheme comparison.

2.1 Factors influencing the design of a Key Management Scheme

The design of a key management scheme is influenced by the following factors:

- Heterogeneous nature of the group membership affects the possible type of encryption algorithm to be used, and the length of the key that can be

supported by an end user.

- The cost of setting up and initializing the entire system parameters, such as, selection of the Group Controller (GC), group announcement, member join and initial key distribution.
- Administrative policies, such as those defining which members have the authorization to generate keys.
- Required level of performance of parameters, such as session sustainability, and key generation rates.
- Required additional external support mechanisms, such as the availability of a Certificate Authority (CA).

2.1.1 Desirable Properties of a Multicast Key Management Scheme

In addition to the facts mentioned above, a multicast key management scheme needs to exhibit the following desirable properties:

1. Ability to handle membership changes. This is important since the whole group must share a single session encryption key. The communication integrity in the presence of membership changes implies the ability of the group to update the session key and distribute it to the valid members with possible back traffic protection.
2. Ability to prevent user collusion. This is important since a subset of members or the deleted members should not be able to collaborate and construct the keys of other members or the future group keys.

3. The ability to provide scalability in terms of security related administration such as member admission, deletion, key revocation and updates. This is important so that the security administration does not become the performance bottleneck.

4. Ability to provide inter-domain issues related to security such as security parameters and cryptographic schemes of various clusters. This is important since some of the members may belong to more than one groups and may need to communicate across the groups.

2.1.2 Abstract Parameters and the Terminology Used

We now present the abstract parameters and common terminology used to explain the properties of various published secure multicast schemes.

1. The set of N receivers (users) in the single sender multiple receiver multicast group is denoted by $M = \{M_1, M_2, \cdots, M_N\}$.

2. The Group Controller or the Group Center (GC) is assumed to be the sender of the group and $GC \notin M$. Hence, the GC is not considered as a receiver.

3. The Session Key is denoted by SK. It is also called the Traffic Encrypting Key or TEK in some of the recent work.

4. The Key Encrypting Key is denoted by KEK.

5. Each user $M_i \in M$ holds a set of keys denoted by $K(M_i)$.

6. The set of all keys used by the group is denoted by $\mathcal{K}$. We note SK $\in \mathcal{K}$. Also $\mathcal{K} = \cup_{M_i \in M} K(M_i)$.

7. The probability of member M_i being revoked is denoted by p_i.

8. The Algebraic expression $H_d = -\sum_{i=1}^{N} p_i \log_d p_i$ denotes the $d-ary$ entropy of the member revocation event.

9. We say members M_i and M_j can collude and *compromise* member M_l if $K(M_l) \subseteq K(M_i) \cup K(M_j)$. Hence, a key assignment free of compromise due to user collusion requires $K(M_l) \not\subseteq \bigcup_{M_i \in M; M_i \neq M_l} K(M_i)$.

10. Encryption of a message m by key K is denoted by $\{m\}_K$

11. Transmission of an encrypted message m from member A to member B, using key K, is denoted by: A $\longrightarrow$ B : $\{m\}_K$.

12. We will denote by K^{-1} the private key corresponding to the public key K. Hence the public key pair is denoted as (K, K^{-1}).

2.2 Summary of Multicast Key Management Schemes

We first summarize the multicast key management schemes that provide key distribution functionality. These schemes do not address key generation or group initialization. We then review schemes that describe key generation.

2.2.1 Group Key Management Protocol

The Group Key Management Protocol (GKMP) in [7, 8], proposes a single GC that is allowed to perform all the security related administrative tasks including member join, deletion, maintenance of ACL, and key generation and distribution. The GKMP has the following features:

1. The group is managed by a single group controller (GC).

2. The group uses Group Traffic Encrypting Key (GTEK), and a future Group Key Encrypting Key (GKEK).

3. The Group Key Packet (GKP) generated by the GC at update step n is given by $GKP_n = \{GTEK_n, GKEK_{n+1}\}$.

The following are the advantages of this method:

1. A single encryption can update the keys for the whole group.

2. The GC has information about the entire membership.

3. The number of keys to be stored is two, the possible minimal value for any scheme without authentication requirements.

The following are the disadvantages of the method:

1. Since all the members share a single key encrypting key (and session key), failure or compromise of a single member compromises the group key packet, and hence the entire future group communication. This forces the entire group to be reinitialized.

2. Single GC becomes a performance bottleneck in terms of security related operations since every join, and deletion has to be done by the GC.

3. Since GC is the only entity permitted to generate the keys, failure or compromise of GC will prevent proper key updates as needed.

2.2.2 Scalable Key Management with Core Based Trees

In an attempt to support scalability in terms of security administration, the Scalable Multicast Key Distribution scheme associated with the Core Based Trees (CBT) was proposed by Ballardie [36]. The following are the properties of the CBT based key management scheme:

1. The CBT uses a single GC to generate the session key and the key encrypting key.

2. The CBT attains scalability in terms of security administration by explicitly allowing any valid member of the group to admit new members and distribute the keys.

The CBT has the following drawbacks:

1. Since the CBT proposes to distribute a single key encrypting key and a single session encrypting key, compromise of a single member will compromise these two keys, and compromise the future group communication.

2. The scalability of CBT comes at the expense of having to assume that *all* the group members can be unconditionally (without any verification mechanism) trusted to distribute the keys only to the valid future members.

2.2.3 Cluster Base Protocols

Since the GKMP and CBT use a single Group Key packet for the entire group, deletion or compromise of a single member invalidates the entire group keys. In [35, 9], a solution to this problem was suggested by proposing to partition the group into clusters. The following are the features of these schemes:

1. A given group is partitioned into clusters.

2. Each cluster is assigned a cluster controller.

3. Each cluster has its own cluster session encrypting key and the cluster key encrypting key.

4. Cluster controller generates the cluster keys, and performs security administration for the cluster.

An advantage of this scheme is that the impact of rekeying is limited to the individual cluster.

The following are the problems associated with these schemes:

1. Finding intermediate single nodes/members that "can be unconditionally trusted" to perform security related operations for cluster control may be difficult. Since the remote monitoring and verification of a single node is difficult, there have been several recent attempts to develop "collective entity" or threshold based administration.

2. Each cluster uses a single key encrypting key. Hence, a single member compromise inside the cluster will compromise the future keys of the cluster as in the case of GKMP, and CBT.

The schemes summarized so far use a common future key encrypting key. Though this approach minimizes the storage requirements, compromise of a single member leads to compromise of all the future keys. Instead, if each member is given a unique key encrypting key, revocation or compromise of a member does not affect the key encrypting keys of other members. However, the key update will require $\mathcal{O}(N)$ encryption in this case. A desired solution is a tradeoff between

these two extremes. One such solution is based on the rooted trees based key distribution, in which more than one key encrypting key is distributed to every member.

2.2.4 Tree Based Schemes for Key Distribution

The first attempt at using a rooted tree based key distribution approach for efficient member revocation was independently proposed in [10] and [11]. Modifications to reduce the computational and key storage requirements at the time of key updates for these two methods were later presented in [17, 15, 14, 13]. We will briefly review the basic concept behind the rooted tree based key distribution below.

2.2.5 Distribution of Keys on the Tree

As a concrete illustration, figure 2.1 presents a KEK distribution based on a binary rooted tree for eight members. In this approach, each leaf of the tree represents a unique member of the group; i.e. the leafs are in one-to-one correspondence with members. Each node of the tree represents a key. The set of keys along the path from the root to a particular leaf node are assigned to the member represented by that leaf node. For example, member M_1 in figure 2.1 is assigned KEKs $\{K_O, K_{2.1}, K_{1.1}, K_{0.1}\}$.

If there is no member deletion/revocation or compromise, the common KEK denoted by K_O can be used to update the session key for all the members. The tree based structure also induces a natural hierarchical grouping among the members. By logically placing the members appropriately, the GC can choose the appropriate keys and hence selectively update, if needed, the keys of the group.

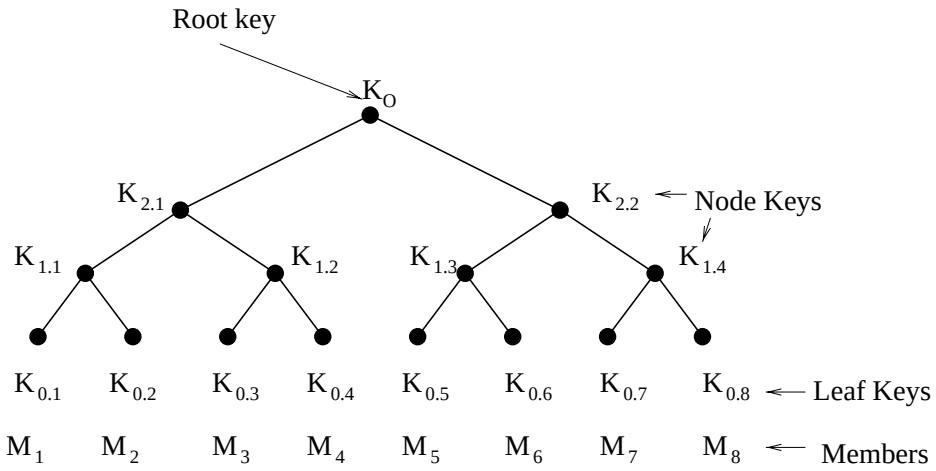

Figure 2.1: The Logical Key Tree of [10, 11, 13, 15, 17]

For example, in figure 2.1, members M_5, M_6, M_7, and M_8 exclusively share the key $K_{2.2}$. The GC can use the key $K_{2.2}$ to selectively communicate with members M_5, M_6, M_7, and M_8. Hence, the local grouping of the members and the keys shared 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. In figure 2.1, if the group controller needs to update the key $K_{2.2}$, it can do so by first generating a new version of $K_{2.2}$, and then performing two encryptions, one with $K_{1.3}$ and the other with $K_{1.4}$. Using the notation $\{m\}_K$ to denote the encryption of 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, we note that the following two messages are needed to update key $K_{2.2}$ to the relevant members of the group.

$\mathrm{GC} \longrightarrow M_5, M_6 : \{K_{2.2}\}_{K_{1.3}}$

$\mathrm{GC} \longrightarrow M_7, M_8 : \{K_{2.2}\}_{K_{1.4}}$

Although we used a tree with uniform depth to all its leaf nodes, this is not

necessary in general. The selection of uniform depth relates to the *maximum entropy* of member revocation event as explained in chapter 4.

2.2.6 Member Revocation in Rooted Trees

Since the SK and the root key are common to all the members in the multicast group, they have to be invalidated at each time a member is revoked. Apart from these two keys, all the intermediate KEKs of the revoked member need to be invalidated. In the event there is bulk member revocation, the GC has to:

- Identify *all* the invalid keys,
- Find the minimal number of valid keys that need to be used to transmit the updated keys.

Member M_1 in figure 2.1 is indexed by the set of five keys $\{K_O, K_{2.1}, K_{1.1}, K_{0.1}\}$. Revoking M_1 is equivalent to invalidating these four keys, generating four new keys, and updating the keys of the appropriate valid members. When M_1 is revoked, the following key updates need to be performed: (a) members $M_5 - M_8$ need to update $\{K_O\}$, (b) members $M_3 - M_4$ need to update $\{K_O, K_{2.1}\}$, and (c) member M_2 needs to update $\{K_O, K_{2.1}, K_{1.1}\}$.

Revocation of a single member involves $\mathcal{O}(\log_d N)$ messages to update $\log_d N$ keys on the rooted $d - ary$ tree. If the number of members to be revoked is two or more, the number of messages may be further minimized depending on the location of the revoked members. In order to further reduce the number of message updates under bulk member removal, the index of the valid members has to be considered and grouped so that the maximum number of valid members share a common key. Direct computations are based on Boolean function minimization

techniques and are not computationally efficient.

2.3 Generation of the Group Keys

We noted that a single node failure at GC can lead to termination of the security related operations of the group. We also noted from the CBT, and IoLus that finding "trusted" intermediate nodes may be a problem in a network. Even in the case of the rooted trees, single node GC is used for key generation which may compromise the whole system if the GC were to be compromised. In our early work [22, 24], we presented a cluster panel based key generation scheme. The approach presented can be used by a set of distributed members to generate a common key. In the context of multicast key management, the cluster control panel jointly generates the common keys and uses them directly or as seeds for generating the keys on the trees. We present the group Diffie-Hellman(DH) approach presented by [2] which does not require a trusted third party for key generation. The Group RSA is also possible [26] but requires a trusted third party. Moreover, the group RSA involves additional testings that are not required in the group Diffie-Hellman Scheme.

2.3.1 Group Diffie-Hellman Scheme

The joint group key generation can lead to a group shared key or a group public key. In the event that the generated key is a shared key, a generalized Diffie-Hellman (DH) approach proposed in [2] can be used. If the desired group key is to be a public key, depending on whether the key belongs to the ElGamal or RSA type, there are two different recent proposals that allow a set of members

N	number of key generating members
i, j, l, m	key generating member indices
M_i	key generating member i
q	order of the algebraic group
g	generator in G
K_i	secret key of member M_i
r_i	random exponent $\in \mathcal{Z}_p^*$, chosen by member M_i.
K	joint key/secret generated by N members.
$K_{i,j}$	shared key between members M_i, and M_j.

Table 2.1: Notations for Group DH problem

to generate the joint public keys with the *assistance of an information theoretic helper.* In many applications such as the secure multicast protocols proposed in [9, 22], it is not always possible to find an "information theoretic helper". Another distributed joint secret generation scheme not based on any hardness problems was proposed in [37, 38]. Although the scheme is computationally efficient, it has some security weakness that will allow two or more members to collaborate and obtain the contribution of individual members. We briefly describe each of these methods below.

2.3.2 Group Diffie-Hellman Extension Keys

The following notations and assumptions are used in describing the group DH problem. These notations are from [2], and are used here to maintain consistency with [2].

All arithmetic operations are performed in the group G which is a cyclic

subgroup of prime order q in $\mathcal{Z}_p^*$, with $p = lq + 1$ for some $l \in \mathcal{N}$. We first describe the simple two-party case and then describe the multiparty case.

2.3.3 Two Party DH Problem

In the simplest setup consisting of two members, denoted M_i and M_j, a solution to the problem is to perform the Diffie Hellman secret generation as follows: Members M_i and M_j agree upon the group and a generator g. Members M_i and M_j choose random integers a, b, such that $1 \leq a, b \leq q - 1$, and compute g^a and g^b. Members then exchange (message exchange is kept to the simplest possible to illustrate the concept)

$M_i \longrightarrow M_j : g^a$

$M_j \longrightarrow M_i : g^b$

After the exchange, each member can independently compute g^{ab} from the message received from the other member. Assuming that performing discrete logarithm is difficult, if a, b are the private keys of members M_i and M_j then the group public key is given by g^{ab}. The group, however, *does not share a common private key*, and the authors in [2] proposed a group key without allowing the members to share the common private key. Every member performs $\mathcal{O}(\log ab)$ "squaring" to compute g^{ab}. The minimum number of message exchanges needed for key establishment is two.

2.3.4 Multiparty Group DH Problem

There are three versions of the multiparty group DH key extensions available in [2]. We will describe the basic one denoted Generalized DH.1 (GDH.1). All three algorithms consist of two steps called *up-flow* and *down-flow*. In the up-flow

stage members collect the contributions from other members and propagate to the next highest indexed member with modification in message sequence. The message exchange for the up-flow is given by: $M_i \longrightarrow M_{i+1} : \{g^{\prod(a_l|l\in[1,j])|j\in[1,i]}\}$. For example, member M_5 receives $\{g^{a_1}, g^{a_1a_2}, g^{a_1a_2a_3}, g^{a_1a_2a_3a_4}\}$ and forwards $\{g^{a_1}, g^{a_1a_2}, \cdots, g^{a_1a_2a_3a_4a_5}\}$ to member M_6. In the up-flow procedure, each member needs to perform one exponentiation. From the indices of the message, member M_i sends i messages to member M_{i+1}. The last member of the group, M_N computes the group key as $K = g^{a_1a_2a_3\cdots a_N}$.

At this stage, member M_N can broadcast the session key value to all the members. Instead of broadcasting the K to all the members, in order to provide the authentication part, the key scheme in [2] has the down-flow part as follows:

$M_{N-i} \longrightarrow M_{N-i+1} : \{g^{\pi(a_l|l\notin[i,j])|j\in[1,i]}\}$.

In the down-flow stage i exponentiations are performed by M_i. One of these enables M_i to compute K, and the rest of the exponentiations ensure that the rest of the group members eventually receive appropriate shares. In order to illustrate this case, we assume that the group size $N = 6$. In this example, the last member M_6 sends M_5 the message $\{g^{a_6}, g^{a_1a_6}, g^{a_1a_2a_6} \cdots, g^{a_1a_2a_3a_4a_6}\}$. Using $g^{a_1a_2a_3a_4a_6}$, it computes $(g^{a_1a_2a_3a_4a_5})^{a_6} = g^{a_1a_2a_3a_4a_5a_6}$. Member M_5 raises the rest of the terms to the power a_5 and distributes to M_4. This process is repeated by each member $M_i(1 \leq i \leq N)$ with appropriate modifications until M_1 computes the session key. There are $\mathcal{O}(N^2)$ messages and exponentiations for such a process.

From the computational steps we note that the group DH is useful in cases where the group symmetric keys are generated or when the group public keys, with no member having the *whole* of the group private keys, are to be generated. It was noted in [2] that the generation of group ElGamal keys with all the mem-

bers contributing without hiding the secrets is not feasible in this setup. In order to provide a common private key for the generated public key, members have to add the individual private keys.

2.3.5 Generating Group ElGamal Keys

In Yung [27], each key generating member was associated with an individual ElGamal public key. The private keys of all the members were added to generate the group private key. The group public key is then the product of the individual public keys. Computational steps are summarized below.

1. The M_i randomly chooses $1 \leq a_i \leq q-1$ and computes the public key g_i^a.
2. Member M_i sends a_i to other members as $M_i \longrightarrow M_j(1 \leq j \leq N; j \neq i) : a_i$.
3. Every member computes the group private key as

$$a = \sum_{i=1}^{N} a_i \text{mod p.} \tag{2.1}$$

4. The group public key is the product of the individual public keys modulo p. If we denote the group public key by K, it is given by

$$K = \prod_{i=1}^{N} g^{a_i} = g^{\sum_{i=1}^{N} a_i}. \tag{2.2}$$

Although this method has less computations, *it exposes the individual private keys of the key generating members.*

2.4 A Distributed Scheme without a Trusted Third Party

In [37, 38], the following solution for contributed joint secret generation was proposed. We present the approach in an algorithmic manner.

1. Members are indexed and every member knows the left or the right neighbor of it.

2. Every member has generated its own secret random integer using an appropriate method.

3. The member M_1 generates a random number α within the range, and adds his/her secret $\gamma_1$to it, and securely communicates $\delta_1 = (\alpha + \gamma_1)$ to the second member. $M_1 \longrightarrow M_2 : \{\alpha + \delta_1\}$.

4. For $i = 2, \cdots, n-1$

   member M_i adds its secret γ_i to the quantity $\delta_{i-1} = \alpha + \sum_{j=1}^{i-1} \gamma_j$, received from member M_{i-1} and securely communicates $\delta_i = \alpha + \sum_{j=1}^{i} \gamma_j$ to member M_{i+1}.

5. Member n receives the quantity δ_i securely communicated by member M_{n-1}, adds its secret γ_n and securely communicates the result $\delta_n = \alpha + \sum_{j=1}^{n} \gamma_j$ to the first member.

6. First member removes the value α and extracts the common secret

   $\theta = \sum_{j=1}^{n} \gamma_j$.

Figure 2.2 presents a view of the distributed computations for group secret generation. The GI (assumed to be a member and denoted here by the index 1

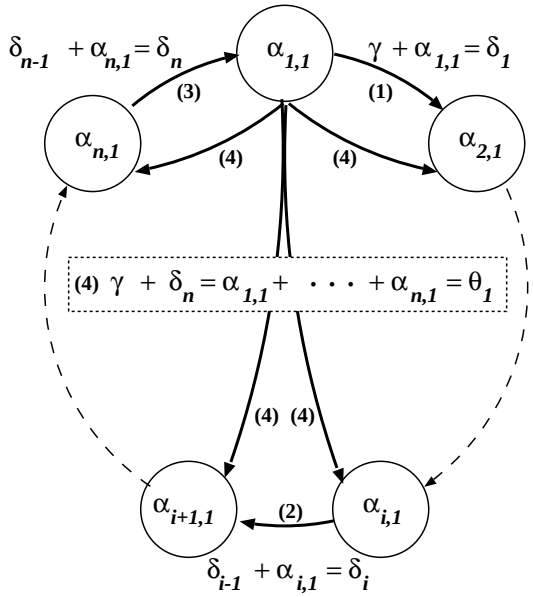

Figure 2.2: Distributed secret generation algorithm

shown in Figure 2.2) can perform the following steps to generate the joint secret of the group:

1. Generate two random integers γ, $\alpha_{1,1}$ and compute $\delta_1 = (\gamma + \alpha_{1,1})$, and send the result to member M_2 (the "next" member in the group) as $M_1 \longrightarrow M_2$: $\{\{T_1, I, 1, \delta_1\}_{K_1^{-1}}\}_{K_2}$.

2. The following steps are repeated for $i = 2, \ldots, n-1$:

   (a) Member M_i generates a random integer $\alpha_{i,1}$.

   (b) Member i then operates on the quantity it received from member M_{i-1} as $\delta_i = (\delta_{i-1} + \alpha_{i,1})$.

   (c) Member M_i then sends the result to member M_{i+1} as $M_i \longrightarrow M_{i+1}$: $\{\{T_i, I, 1, \delta_i\}_{K_i^{-1}}\}_{K_{i+1}}$.

3. Eventually, the group member M_n receives δ_{n-1} and then generates a uniformly-distributed random number $\alpha_{n,1}$, performs $\delta_n \equiv (\delta_{n-1} + \alpha_{n,1})$, and then securely sends it to the initiating member M_1 as
   $M_n \longrightarrow M_1$: $\{\{T_n, I, 1, \delta_n\}_{K_n^{-1}}\}_{K_1}$.

4. The initiator (M_1) then decrypts it and performs

$$\theta_1 = (-\gamma + \delta_n) \tag{2.3}$$

   and sends θ_1 to each member i, for $i = 2, \ldots n$, as
   $M_1 \longrightarrow M_i : \{\{T_1, I, 1, \theta_1\}_{K_1^{-1}}\}_{K_i}$.

If there is no member collaboration, this approach prevents a member from knowing the individual secret of any other member. The computations scale as $\mathcal{O}(N)$ with N being the group size. We now show that this scheme reveals the secret of any one member under the following collaboration of any two members. The proposed attack works even for the boundary indices if we consider the member indices as forming a physical ring, thus providing a right and left neighbor for any given member.

2.4.1 A user Collusion Problem of the Scheme

Let M_i and M_{i+2} be collaborators. After computing the quantity $\delta_i = \alpha + \sum_{j=1}^{i} \gamma_j$, member M_i securely communicates it to member M_{i+1}. It also securely communicates the quantity (without the knowledge of member M_{i+1}) to member M_{i+2}.

The member M_{i+1} computes the secret $\delta_{i+1} = \alpha + \sum_{j=1}^{i+1} \gamma_j$ and securely communicates it to member M_{i+2}. Using these two quantities, member M_{i+2} can

extract the secret of member M_{i+1} in a straight forward manner. Hence, the individual secret is not guarded against collusion in this approach.

2.4.2 Summary of the Distributed Key Generation Schemes

From the key generation schemes, we note that the Group Diffie-Hellman method provides resistance to user collusion but can not provide public keys. On the other hand, the scheme in [37, 38] can be used for shared keys or public keys. The scheme in [37, 38] however suffers from user collusion.

In the next chapter, we provide a new key generation scheme that can be used to generate a joint secret while resisting user collusion. After initialization, our approach can be combined with the scheme by Koblitz [37, 38] to improve the performance. We do not require exponentiation as in the group Diffie-Hellman method.

Chapter 3

Joint Key Generation

We present a set of possible distributed key generation schemes that can be used by the control panels introduced in the previous section. In doing so, we first present the existing feasible schemes and then present an approach that can be thought of as generalization of the *one-time* pad techniques. We note that the scheme proposed doesn't depend on any property of secure multicast and hence can be used in other applications which require joint secret generation as well.

As a reminder, we note that the key management proposals in [9, 36] lacked a mechanism to select, and allow intermediate nodes to perform key generation and distribution. This chapter addresses the issue of selecting a set of intermediate nodes to jointly perform the key generation for the group. Our procedure admits both shared key and public key generation.

The key generating group consists of N members who are assumed to be mutually suspicious. Each member is provided with group initialization parameters such as individual pad and group binding parameter. A member generates its share of the secret which we call *Fractional Key* (FK), hides it using the pad, securely exchanges it with the rest of the members and combines shares of all

the members to generate the *hidden joint secret*. The group is also parameterized using a group binding parameter that can be used to remove the combined effect of all the pads, thus revealing the joint secret to the members possessing the hidden joint secret. In our approach, we assume that there is a trusted third party, such as the group initiator, that will select and initialize the group key generation procedure. We now present our scheme for allowing a set of specific members to generate the joint keys.

3.1 Assumptions

The following is a list of the underlying assumptions of our proposed scheme:

- There exists a binary operation $\oplus$ that operates on the set S of *elements* generating the secret such that $S \oplus S :\rightarrow S$.
- The shared keys are generated by a fixed number of participants n.
- A mechanism exists for certifying the members participating in the key generation procedure, for securely exchanging the quantities required in the algorithm and for authenticating the source of these quantities.
- Every member can generate uniformly distributed, independent random numbers in a given range.

3.2 Notations Used

The following notations are used to describe the different quantities used in the proposed method:

$\alpha_{i,j}$: The one-time pad of the ith member at the jth secret update iteration.

θ_j: The pad binding parameter at the jth secret update iteration.

$FK_{i,j}$: The *fractional key* of the ith member at the jth secret update iteration.

$HFK_{i,j}$: The *hidden* $FK_{i,j}$ of the ith member at the jth secret update iteration.

SK_j: The group shared key at the jth key update instance.

$A \longrightarrow B : \mathcal{X}$: Principal A sends principal B a message $\mathcal{X}$.

Given the above listed assumptions, the joint secret generation scheme consists of the following major parts:

1. Initialization: distribution of initial pad and binding parameters.
2. Generation of the common shared secret using the hidden fractional keys of individual members.

3.3 Initialization

The group initiator chooses n uniformly distributed, mutually independent random numbers (initial pads of members) $\{\alpha_{i,0}\}_{i=0}^{n}$. It also chooses $\theta_0 = \sum_{i=0}^{n} \alpha_{i,0}$. A member i is distributed a unique initial pad $\alpha_{i,0}$, and the initial *group binding parameter* θ_0.

3.4 Description of the Computational Steps

Figure 3.2 represents the symbolic computation of the proposed method at secret update step j.

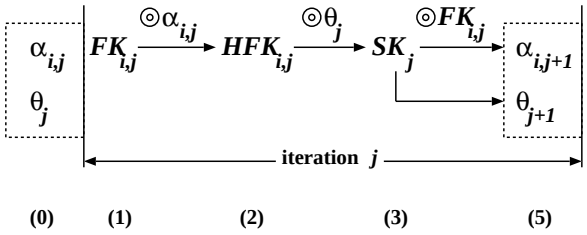

Figure 3.1: Iteration and mappings of the key generation algorithm

1. At the time of initialization, n members are selected (depending on an application specific procedure) and given initial pads denoted by $\alpha_{i,1}; 1 \leq i \leq n$ such that

$$\alpha_{1,1} \oplus \alpha_{2,1} \oplus \cdots \oplus \alpha_{n,1} = \theta_1. \tag{3.1}$$

   Here, $\oplus$ is the binary operation defined over the set of valid keys. For example, one possible selection is to set $\oplus$ to be modulo p with a large prime p.

2. In the first iteration step, every member i uniformly picks a value $FK_{i,1}$ from the set S of valid individual shares, and generates its hidden share as $HFK_{i,1} = FK_{i,1} \oplus \alpha_{i,1}$. These shares are then securely communicated to all other members.

3. Every member i locally computes

$$\sum_{i=1}^{n} \oplus HFK_{i,1} = \sum_{i=1}^{n} \oplus (\alpha_{i,1} \oplus FK_{i,1}) \tag{3.2}$$

$$= \lambda_1 \theta_1 \oplus \theta_2 \tag{3.3}$$

   with

$$\theta_2 = \sum_{i=1}^{n} \oplus FK_{i,1} \tag{3.4}$$

and $\lambda_1\theta_1$ is the result of operation $\oplus$ performed on θ_1, λ times. Especially, we note that λ_1 need not be a scalar (i.e. need not belong to the same field or ring as $\alpha_{i,j}$ or θ. This is essential in applying our method to Elliptic curves).

4. Every member then locally computes the new value of the group shared secret θ_2 by removing the effect of the initial shared secret value

$$\theta_2 = \lambda_1\theta_1 \oplus \theta_2 \oplus \mu_1\theta_1 \tag{3.5}$$

(where μ is the appropriate *inverse* of λ. For example, if $\oplus$ is addition operation under modulo p where p is a large prime, then $\mu = p - \lambda$.)

5. Every member i locally computes its new pad as

$$\alpha_{i,2} = \theta_2 \oplus \gamma_2 FK_{i,2} \tag{3.6}$$

essentially removing the effect of its own share.

6. At the share update step j , the procedure is:

   - generate new individual shares $FK_{i,j}$
   - combine it with the individual dynamic pad $\alpha_{i,j}$ to generate $HFK_{i,j}$
   - exchange HFK's of all the members securely
   - compute the new shared secret $\theta_{j+1} = \lambda_j\theta_j \oplus \theta_{j+1} \oplus \mu_j\theta_j$ with μ is the appropriate inverse of λ.
   - compute the new individual pad $\alpha_{i,j+1} = \theta_{j+1} \oplus \gamma_{j+1} FK_{i,j+1}$.

This summarizes the computational steps of the proposed scheme. We now identify possible structures that are relevant to this computational scheme.

3.5 Necessary Algebraic Structures

From the brief description of our proposed scheme, we note that the following properties are needed for the operator $\oplus$ along with the set S:

- Combination of the pads along with the individual shares should lead to HFKs that are contained in the set S, else the shared key SK need not belong to set S. Hence, the operation $\oplus$ on set S is algebraically *closed*.

- Even if the members combine the HFK's in a specified order, independent of which HFKs in the order are processed first, the result of the combination should be same. Hence, the operator $\oplus$ needs to be associative. For example

$$(HFK_{1,j} \oplus HFK_{2,j}) \oplus HFK_{3,j} = HFK_{1,j} \oplus (HFK_{2,j} \oplus HFK_{3,j}) \quad (3.7)$$

- In order to guarantee freshness, members need to be able to separate the values of the *previous* and *new* shared secret, and operator $\oplus$ also needs to be commutative. This means that the expression

$$(\alpha_{1,j} \oplus FK_{1,j}) \oplus (\alpha_{2,j} \oplus FK_{2,j}) \oplus \cdots (\alpha_{n,j} \oplus FS_{n,j}) \quad (3.8)$$

  can be separated and written as

$$(\alpha_{1,j} \oplus \alpha_{2,j} \cdots \oplus \alpha_{n,j}) \oplus (FS_{1,j} \oplus FS_{2,j} \cdots FS_{n,j}) = \lambda_j \theta_j \oplus \theta_{j+1}. \quad (3.9)$$

- Since combining HFKs leads to the hidden shared key value $\lambda_j \theta_j \oplus \theta_{j+1}$, we need to find the inverse of $\lambda \theta_j$ to remove the effect of the *previous key* θ_j. If elements $FK_{i,j}$ are uniformly picked from set S, then the value of θ_j will be uniformly distributed and hence θ_j can take any value from the set S. As a result, every element in the set S has an inverse with respect to the operation $\oplus$.

It is important to note that if the removal of the previous θ_j is not required, then the inverse is not needed. If the elements do not possess (have) the associative property under $\oplus$, the minimal necessary structure is that the set S along with the operator $\oplus$ is groupoid. If associativity is allowed, the semigroup structure is enough.

The minimal necessary algebraic structure that allows associativity, commutativity and existence of an inverse for all given elements is a commutative group. One immediate example is a group of prime order which is widely used in cryptosystems.

Using the approach presented above, we present a group shared key generation scheme, and a group ElGamal public key scheme.

3.6 Generation of Group Shared Key

The computational grid diagram for this method is the same as before, and is shown below with slight modifications.

The key generation algorithm is an iterative process depicted in Figure 3.2. Each iteration j requires as input (indicated as step (0) in the figure) a set of one-time pads $\alpha_{i,j},\ i = 1, \ldots, n$, and the binding parameter θ_j, which are obtained from the initialization algorithm for iteration $j = 1$, and from the preceding iterations for $j > 1$.

The iterative key generation algorithm consists of the following steps (1)-(5):

1. For $i = 1, \ldots, n$, a member i generates a *cryptographically-secure* random number $FK_{i,j}$.

2. For $i = 1, \ldots, n$, a member i generates a quantity $HFK_{i,j} = \alpha_{i,j} \oplus FK_{i,j}$,

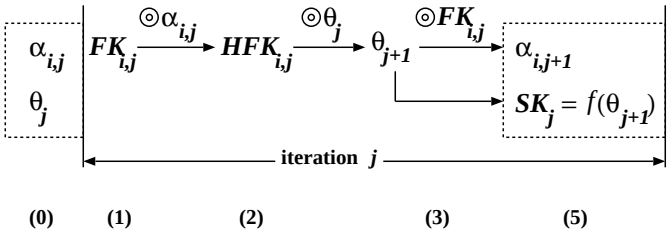

Figure 3.2: Iteration and mappings of the key generation algorithm

and all the members securely exchange the HFKs as $\forall\ 1 \leq l, m \leq n, l \neq m$, $l \longrightarrow m$: $\{\{HFK_{l,j}\}_{K_l^{-1}}\}_{K_m}$.

3. Once the exchange is complete, each member computes the new group parameter θ_{j+1} as
   $\theta_{j+1} = \lambda\theta_j \oplus HFK_{1,j} \oplus HFK_{2,j} \oplus \cdots \oplus HFK_{n,j}$.
   $\Rightarrow \theta_{j+1} = FK_{1,j} \oplus FK_{2,j} \oplus \cdots FK_{n,j}$.

4. If the resulting group parameter θ_{j+1} is *cryptographically-insecure* for a particular application, all members can repeat steps (1) - (3) creating a new high quality group parameter θ_{j+1}.

5. For $i = 1, \ldots, n$, a member i computes $\alpha_{i,j+1} = \theta_{j+1} \oplus FK_{i,j}$, and $SK_j = f(\theta_{j+1})$ where $f(\cdot)$ is a pseudo random function.

3.7 Generation of the Group Elgamal Keys over $\mathcal{Z}_p^*$

We assume that all the members agree on prime p, and generator g.

The iterative key generation algorithm consists of the following steps (1)-(5):

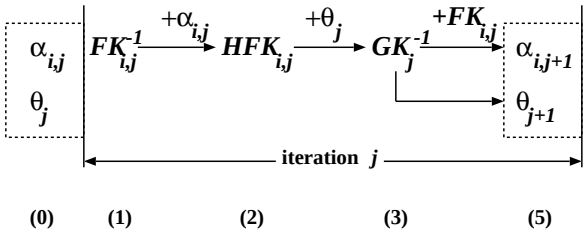

Figure 3.3: Iteration and mappings of the key generation algorithm

1. For $i = 1, \ldots, n$, a member i randomly picks up a number $FK_{i,j}$ with $0 \leq FK_{i,j} \leq p-2$ and generates $g^{FK_{i,j}}$. The public key is $(p, g, g^{FK_{i,j}})$. The private key is $FK_{i,j}$.

2. All the members publish their public key $(p, g, g^{FK_{i,j}})$. *This prevents members from introducing any bias the group private key later.*

3. For $i = 1, \ldots, n$, a member i generates a quantity
   $HFK_{i,j} = (\alpha_{i,j} + FK_{i,j}) \bmod p$, and then all members securely exchange the HFKs as
   $\forall\ 1 \leq l, m \leq n,\ l \neq m,$
   $l \longrightarrow m$: $\{\{HFK_{l,j}\}_{FK_{l,j-1}^{-1}}\}_{FK_{m,j-1}}$.

4. Once the exchange is complete, each member computes the θ_{j+1} as
   $\theta_{j+1} = ((p-2)\theta_j + \sum_{i=1}^{i=n} HFK_{i,j}) \bmod (p-1)$. The group public key is $g^{\theta_j} = \prod_{i=1}^{i=n} g^{FK_{i,j}}$.

5. If the resulting group key pair is *cryptographically-insecure* for a particular application, all members can repeat steps (1) - (3) creating a new high quality key pair.

6. For $i = 1, \ldots, n$, a member i computes the iteration update as
   $\alpha_{i,j+1} = (\theta_{j+1} + FK_{i,j}) \bmod p$.

The steps (1) - (5) present the computational steps for generating the keys at each update. At the end of step (1), a member i generates the jth update of its ElGamal public key pair. Member i then *hides* the key in step (2) by generating a $HFK_{i,j}$ and sends it to the other participating members. The public/private key pairs used in the exchanges of iteration j are the individual ElGamal fractional key pairs of iteration $j-1$ At this stage, every member can independently combine the shares of all the key generating members and derive the group private key θ_j. If the members decide not to generate any public key pair, then they can use this as a group secret shared key. However, if the group decides to generate a group ElGamal public key pair, the members then obtain the value of $\theta_j \bmod (p-1)$ as the private key. The corresponding public key is given by $g^{\theta j}$.

We note that the key generation procedure described above combines the following different features:

- For a single member, generation of the public key pair uses the standard ElGamal method which is based on the assumption that it is difficult to perform the discrete logarithm function.

- Generation of the HFK's is based on the result that if two numbers are generated uniformly and independently, identically distributed (iid), then given a non-trivial function of them, it is difficult to derive the individual components.

- Generation of the group public key is a generalized ElGamal public key system. The main result here is that although every member can individually generate the same group public key pair, they don't have the direct access each other's private keys.

- Even if an attacker breaks the group private key (via traffic analysis) and, hence, the group parameter θ_j for the next iteration, the attacker still has to break another $(n-1)$ ElGamal keys to obtain the messages exchanged in the next key update. From the computational point of view, this implies the group key length can be made smaller if the message is relevant for only a limited time frame.

- The time-varying pad $\alpha_{i,j+1}$ is computed such that, for an outsider, obtaining $\alpha_{i,j+1}$ is as hard as obtaining the actual key FK_j at any given time.

- Although all the members each have a $HFK_{i,j}$, obtaining the $FK_{i,j}$ involves brute force search. Hence, even if a fellow member becomes an attacker, that rogue member has the same amount of computational burden in obtaining the FK as a crypto analyst; i.e. trust is not unconditional.

- By the same argument above, we note that only the member i can compute $\alpha_{i,j+1}$. Everyone else has to perform a brute force search before finding $\alpha_{i,j+1}$, which is time-varying.

- Even if an outsider captures and decrypts a packet and obtains the HFK of a single participating member, the attacker is faced with the following challenges:

    1. Having a HFK does not give any advantage to the attacker in decrypting any message encrypted with a g^{θ_j}.

    2. The outside attacker has to find the corresponding remaining $(n-1)$ HFKs. Such is the case since the keys are transported in a secure manner. Hence, only the participating members have the direct access

to the HFKs. For an outsider, it may be much harder to simultaneously attack and obtain these $(n-1)$ parts that have limited lifetime.

- FK's, as well as the GK, are checked for standard weaknesses before being used.

3.8 Recovering the Fractional Key of a Single Node

The following steps are involved in recovery of the $FK_{\tilde{\imath},j}$ and $\alpha_{\tilde{\imath},j}$ of the node failed $\tilde{\imath}$, where j represents the iteration number in which the node was compromised or failed.

1. Any one FK-generating member—called the Recovery Initiator (RI)—must initiate recovery and give the HFK of the failed node $\tilde{\imath}$ to the newly-elected node i as
   $RI \longrightarrow i : \{\{HFK_{\tilde{\imath},j}\}_{FK^{-1}_{RI,j}}\}_{FK_{i,j}}$.

2. The RI must also give the newly-elected node i the current θ_j as
   $RI \longrightarrow i : \{\{\theta_j\}_{FK^{-1}_{RI,j}}\}_{FK_{i,j}}$.

3. Using the same algorithm as is used for distributed initialization only with $\alpha_{l,j}$ replaced with β_l, the RI initiates a distributed process whereby member l is given two random numbers (γ, β_l) as in the initialization with $\gamma \equiv \sum_{i=1}^{i=n} \beta_i \bmod p$.

4. For $l = 1, \ldots, n-1$, each node l then computes a *modified* hidden fractional key $\widehat{HFK}_{l,j} = (\beta_l + FK_{l,j}) \bmod p$ and hands it to the newly-elected member

i as $l \longrightarrow i : \{\{\widehat{HFK}_{l,j}\}_{FK_{l,j}^{-1}}\}_{FK_{i,j}}$.

5. Node i then combines all of the modified HFKs and recovers the private key $FK_{\tilde{i},j}$ using the operation $FK_{\tilde{i},j} = \{\theta_j + (p-1)(\gamma + \sum_{i=1}^{i=n} \widehat{HFK}_{i,j})\} \bmod p$.

6. Node i then extracts the pad $\alpha_{\tilde{i},j}$ using the operation $\alpha_{\tilde{i},j} = (\theta_j + (p-1)(\gamma FK_{\tilde{i},j})) \bmod p$.

We note that the recovered values of $FK_{\tilde{i},j}^{-1}$ and $\alpha_{\tilde{i},j}$ are unique. Once the new node recovers the fractional key of the compromised node, it can inform the other contributing members to update the iteration number j to $j+1$, and then all members can execute the key generation algorithm. Note that even though the newly-elected member recovers the compromised fractional key and pad, the next key generation operation of the new node does not use the compromised key or pad. Hence, even if the attacker possesses the fractional key or pad at iteration j, it does not allow the attacker to obtain the future fractional keys or pads without any computation.

Although $n = 3$ can generate the keys, if a single member exposes its secret, the remaining two members can compute each others pads as follows, thus breaking the system. Hence, four is the minimum necessary member size of this procedure. This is summarized as

Lemma 3.1: Independent of how non-trivial the bit-length of the key is, operating with $n = 3$, a FA can invalidate the system's *zero knowledge proof* capability within the group.

Proof: Assume that the time instant at which one member i ($i = 1$ or 2 or 3) becomes a *rogue* is j. At this time the members have values of $\alpha_{1,j} = (HFK_{2,j} + HFK_{3,j}) \bmod p$, $\alpha_{2,j} = (HFK_{3,j} + HFK_{1,j}) \bmod p$, $\alpha_{3,j} = (HFK_{1,j} +$

$HFK_{2,j}) \bmod p$. Every member also has access to the current θ_j and their own $FK_{l,j}$ ($l = 1, 2, 3$). At this stage, obtaining the α component of any other member is as computationally intensive as an outside attacker trying to obtain θ_j. However, if a member, say $i = 1$, is compromised and releases its secret $\alpha_{1,j}$, then each of the other members can use this and compute $FK_{1,j} = (\alpha_{1,j} + \theta_j) \bmod p$. Since the $\theta_j = (FK_{1,j} + FK_{2,j} + FK_{3,j}) \bmod p$, each member can now compute the other member's FK as well.

3.9 Proofs of Computational Security

In this section, we will show that the given scheme is protected from the external threat of traffic analysis. In doing the analysis we will assume that the number of key generating members is even. We also show that given the HFK of all the members involved in key generation, the best thing a crypto analyst can do is to *guess* the random number arbitrarily (if the random variables are chosen uniformly). We also present a measure of departure from the ideal case in terms of mutual information.

3.10 Mutual Information between the FK and HFK

In the scheme described above, we note that at each time instance j, a member i first generates a $FK_{i,j}$ and performs modulo addition of it with the time varying pad $\alpha_{i,j}$ to generate a $HFK_{i,j}$. We note that every member generates its $FK_{i,j}$ using a random number generator that outputs iid random variables. Hence:

- By assumption $FK_{i,j}$'s are mutually independent. Moreover, the $FK_{i,j}$'s at different time updates are also independent. i.e.

$$I(FK_{i,j} \wedge FK_{l,m}) = H(FK_{i,j}) - H(FK_{i,j}|FK_{l,m}) = 0 \tag{3.10}$$

  where $(i,j) \neq (l,m)$.

- The time varying pad $\alpha_{i,j}$ is given by

$$\alpha_{i,j} = SK_{j-1} \oplus FK_{i,j-1} = \sum_{i=1}^{n} \oplus FK_{i,j-1}. \tag{3.11}$$

  Hence, $\alpha_{i,j}$ is independent of $FK_{i,j-1}$, and $FK_{i,j}$ i.e. the pad of member i at time update j is independent of all FK's of that member which implies $I(\alpha_{i,j} \wedge FK_{i,l}) = H(\alpha_{i,j}) - H(\alpha_{i,j}|FK_{i,l}) = 0$, $\forall l$.

- Since the pad $\alpha_{i,j}$ of a member j is a function of the FKs of the other $n-1$ key generating members at time update instance $i-1$, and since all the FKs are iid, we have that $\alpha_{i,j}$ is independent of all the FKs at any other time instance, i.e. $I(\alpha_{i,j} \wedge R_{l,m}) = H(\alpha_{i,j}) - H(\alpha_{i,j}|R_{l,m}) = 0$, $\forall m \neq j-1$.

Using these observations, at key update time instance j, for member i, the mutual information between FK and HFK can be computed as:

$$\begin{aligned} I(HFK_{i,j} \wedge FK_{i,j}) &= I(\alpha_{i,j} \oplus FK_{i,j} \wedge FK_{i,j}) && (3.12)\\ &= H(\alpha_{i,j} \oplus FK_{i,j}) - H(\alpha_{i,j} \oplus FK_{i,j}|FK_{i,j}). && (3.13)\\ &= H(\alpha_{i,j} \oplus FK_{i,j}) - H(\alpha_{i,j}). && (3.14) \end{aligned}$$

From this equation we note that if the FKs are uniform as well as iid, then $\alpha_{i,j} \oplus FK_{i,j}$ is also uniform. For this case we have that

$$I(\alpha_{i,j} \oplus FK_{i,j} \wedge FK_{i,j}) = H(\alpha_{i,j} \oplus FK_{i,j}) - H(\alpha_{i,j}) \quad (3.15)$$

$$= \log L - \log L = 0. \quad (3.16)$$

Hence, if the random variable generator gives "uniformly" distributed iid quantities, then the mutual information between the HFK and the FK is zero; i.e. given the HFK, the best thing the attacker can do is to guess the FK. We note that this statement is true independent of whether the attacker is another key generating member or not. Hence, another single group member cannot extract the FK of a member i by obtaining the HFK of other members; i.e. the pad $\alpha_{i,j}$ does provide the desired randomization for the FK $FK_{i,j}$ for the member i at key update time j.

We also note that in generating the HFKs, we are performing modulo addition of two uniformly distributed rvs. None of them have any language structure. Hence, the attacker cannot use word frequency analysis or any other language constructs to reduce the *search space*.

In summary, if the r.v's are generated as uniform and iid, then there is perfect secrecy between the HFKs and the FKs. We also need mutual independence with respect to the initial parameters. Under these conditions of distributions no one, including the other key generating members, can make use of $HFK_{i,j}$ to extract $FK_{i,j}$.

3.11 Proofs Based on Conditional Probabilities

In this sections we show how to use probabilistic arguments to derive the secrecy conditions satisfied by the model. In order to do that we assumed that the FK's,

HFK's and the $\alpha_{i,j}$, all have the space of same size. We first note that the proposed scheme is analogous to a two step procedure as shown below

$$\begin{aligned} f_1(FK_{i,j};\, \alpha_{i,j}) &= FK_{i,j} \oplus \alpha_{i,j} = HFK_{i,j} && (3.17) \\ f_2(HFK_{1,j};\, \cdots HKF_{n,j};\, \theta_j) &= HFK_{1,j} \oplus \cdots \oplus HFK_{n,j} \oplus \theta_j \\ &= SK_j && (3.18) \end{aligned}$$

$$f_3(FK_{i,j};\, SK_j) = \alpha_{i,j+1}$$

$$\theta_{j+1} = SK_j$$

Theorem 3.1: The fractional key based shared key generation scheme provides perfect secrecy for map $f_l, l = 1, 2, 3$ iff FK's are chosen uniform and the map f_l is injective.

Proof: We will give the proofs for maps f_1 and f_2. Note that the proof does not really depend on the type of operation $\oplus$ is except for the fact that the operation $\oplus$ needs to be invertible.

We can consider the function $f_1(.)$ as the encryption of "message" $\alpha_{i,j}$ using the key $FK_{i,j}$ to get the cipher text $HFK_{i,j}$. Let's denote $FK_{i,j} \in \mathcal{FK}$, $HFK_{i,j} \in \mathcal{HFK}$, $\alpha_{i,j} \in \mathcal{A}$. Hence, for each $\alpha_{i,j} \in \mathcal{A}$ and $HFK_{i,j} \in \mathcal{HFK}$ there is at least one $FK_{i,j} \in \mathcal{FK}$ such that $f_1(FK_{i,j}, \alpha_{i,j}) = FK_{i,j} \oplus \alpha_{i,j} = HFK_{i,j}$ and

$$|\mathcal{A}| = |\{f_1(FK_{i,j}, \alpha_{i,j}) : FK_{i,j} \in \mathcal{FK}\}| \leq |FK|. \tag{3.19}$$

However, the fractional key scheme is such that $|\mathcal{FK}| = |\mathcal{HFK}| = |\alpha|$. Hence,

$$|\{f_1(FK_{i,j}, \alpha_{i,j}) : FK_{i,j} \in \mathcal{FK}\}| = |FK| \tag{3.20}$$

Therefore, we have shown that if $FK_{i,j} \neq \hat{FK}_{i,j}$ then $f_1(FK_{i,j}, \alpha_{i,j}) \neq f_1(\hat{FK}_{i,j}, \alpha_{i,j})$. Hence, for a given $\alpha_{i,j} \in \mathcal{A}$ and $HFK_{i,j} \in \mathcal{HFK}$, we have only

one $FK_{i,j}$ such that $f_1(FK_{i,j}, \alpha_{i,j}) = HFK_{i,j}$. There are $|FK|$ possible "keys". Hence, given an HFK, there are $|FK|$ possible unique mapping of each $\alpha_{i,j}$ to HFK, i.e., $f_1(FK_{l,i,j}, \alpha_{l,i,j}) = HFK_{i,j}$, $1 \leq l \leq |FK|$. This leads to

$$\begin{aligned} P(\alpha_{l,i,j}|HFK_{i,j}) &= \frac{P(HFK_{i,j}|\alpha_{l,i,j})P(\alpha_{l,i,j})}{P(HFK_{i,j})} \\ &= \frac{P(FK_{l,i,j})P(\alpha_{l,i,j})}{P(HFK_{i,j})} \end{aligned} \tag{3.21}$$

Perfect secrecy condition yields $P(\alpha_{l,i,j}|HFK_{i,j}) = P(\alpha_{l,i,j})$. Using this we have $P(FK_{l,i,j}) = P(HFK_{i,j}), 1 \leq l \leq |FK|$. This means that the "keys" are drawn with equal probability. But since the number of "keys" is $|FK|$, we must have that the $FK_{i,j}$ are chosen uniformly. Hence, if we have perfect secrecy for $f_1(.)$ then we must choose the keys FK's uniform.

Proof of the converse part: If the conditions are satisfied - namely that the FK's are chosen uniformly and only one set of HFK and $\alpha_{i,j}$ is mapped to a unique $FK_{i,j}$ then the proof of perfect secrecy is given by

$$P(HFK) = \sum_{FK_{i,j} \in FK} P(FK_{i,j})P(f^{-1}(HFK)) \tag{3.22}$$

$$= |FK|^{-1} \sum_{FK_{i,j} \in FK} P(f^{-1}(HFK)) \tag{3.23}$$

$$= |FK|^{-1} \tag{3.24}$$

$$\tag{3.25}$$

The last step follows from the fact that for a given key, only one message is assumed to be mapped to the cipher. Hence, independent of the distribution of $f^{-1}(HFK)$ the summation should be equal to 1. Using the result above we have

$$P(\alpha_{i,j}|HFK) = \frac{P(HFK|\alpha_{i,j})P(\alpha_{i,j})}{P(HFK)} \tag{3.26}$$

Hence, the system defined by function f_1 has perfect secrecy. The proofs for the function f_3 is identical to that of f_1. To prove that the function f_2 also implies perfect secrecy, we note that

$$
\begin{aligned}
f_2(HFK_{1,j};\cdots HKF_{n,j};\,\theta_j) & \qquad (3.27)\\
&= HFK_{1,j} \oplus HFK_{2,j} \cdots \oplus HFK_{n,j} \oplus \theta_j \\
&= FK_{i,j} \oplus \alpha_{i,j} \oplus HFK_{2,j} \cdots \oplus HFK_{n,j} \oplus \theta_j \\
&= f_1(FK_{i,j};\,\alpha_{i,j}) \oplus HFK_{2,j} \cdots \oplus HFK_{n,j} \oplus \theta_j \\
&= FK_{i,j} \oplus HFK_{2,j} \cdots \oplus HFK_{n,j} \oplus \theta_j \\
&= \theta_j \oplus \alpha_{i,j}
\end{aligned}
$$

This can be written as

$$FK_{i,j} \oplus \lambda_{i,j} = f_1(FK_{i,j};\lambda_{i,j}) = \gamma_{i,j} \qquad (3.28)$$

Now, for f_1 we have already shown that perfect secrecy is achieved iff $FK_{i,j}$ is uniformly chosen as long as only one key relates the "plain text" and the "cipher". Hence SK_j is uniformly distributed if $FK_{i,j}$ is uniformly chosen and every member makes sure that no fractional key is repeatedly used by them.

In summary, if the fractional keys are chosen uniformly and the function $\oplus$ is such that only one fractional key relates a given HFK and the pad, then we have perfect secrecy if the bit length of these quantities are same. Clearly, modulo addition operation satisfies these requirements.

Chapter 4

Key Revocation

As noted in chapter two, there are many variations of the rooted tree based key distributions proposed to minimize the storage at the group controller and the members while providing a reduction in the amount of encryptions required to update the session key [36, 10, 11, 13, 15, 17, 14]. Many of these tree based schemes seem to present different optimal values for the required keys to be stored at the GC and the user node.

We show that these methods can be analyzed in a systematic manner. We also show that the design of an optimal tree is closely related to Huffman trees and the *entropy of member revocation event*. We then show that this entropy provides a bound on the providable key length if all the keys are of the same length. We perform weakness analysis of some of the recent rooted tree based schemes using entropy and show that these schemes do not scale well.

We then show how to generate a key management scheme with specific amount of user collusion, thus generating a family of key management schemes.

4.1 Distribution of Keys on the Tree

We reproduce the tree structure from chapter two in this section. The figure 4.1 presents a KEK distribution based on a binary rooted tree for 8 members. As noted earlier, the leafs are in one-to-one correspondence with members. Each node of the tree represents a key. The set of keys along the path from the root to a particular leaf node are assigned to the member represented by that leaf node. For example, member M_1 in figure 4.1 is assigned KEKs $\{K_O, K_{2.1}, K_{1.1}, K_{0.1}\}$.

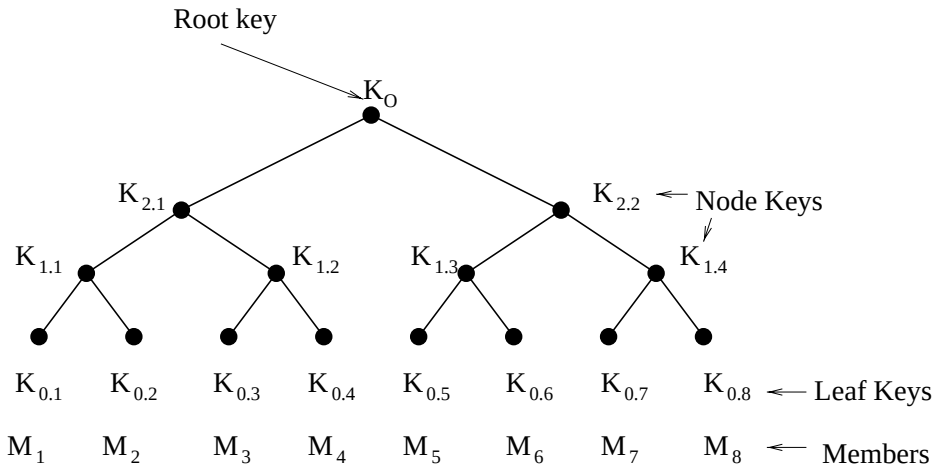

Figure 4.1: The Logical Key Tree of [10, 11, 13, 15, 17]

Member revocation details were presented in chapter two and are not reproduced here.

The following observations can be made towards the rooted tree based key distributions:

- Since each member is assigned $\log_d Nd^2$ keys, deletion of a single member requires $\log_d Nd^2$ keys to be invalidated.
- Since there are $\log_d Nd$ nodes between the root and a leaf and $\log_d N$ nodes are shared with other members, and for each common node one encryption

is required, the GC needs to perform a total of $\log_d N$ encryptions.

- For a $d-ary$ tree with depth $h = \log_d N$, the GC has to store $1+1+d+d^2+\cdots+d^h = \frac{d(N+1)-2}{(d-1)}$ number of keys. Setting $d = 2$ leads to the binary tree for which the required amount of storage works out to be $\frac{2(N+1)-2}{2-1} = 2N$. This result can be independently checked by noting that a binary tree with N leafs has $2N-1$ nodes. Hence the GC has to store the SK and $(2N-1)$ KEKs, leading to $2N$ keys that need to be stored.

In [15, 17], binary rooted tree based key distributions which require GC to store a total of $2\log_2 N$ keys were proposed. The generalized version of this result requires $d\log_d N$ keys to be stored at the GC. Each member needs to store only $\log_d Nd^2$ keys in this scheme. However, the number of keys to be updated remain at $\log_d N$ as in [10, 11]. Hence, at first glance, the results in [17] seem to reduce the storage requirements for the GC by

$$\frac{d(N+1)-2}{d-1} - d\log_d N = \frac{d(N+1-(d-1)\log_d N)-2}{(d-1)} \tag{4.1}$$

number of keys without increasing the key storage requirements at the end user node.

4.2 Preliminary Observations

We first show the need to optimize the rooted-tree using a worst case example. Let us consider the binary rooted-tree shown in figure 4.2.

Since the SK and the root key are common to all the members, they will be invalidated each time a member is revoked. In this tree, if all the members have equal probability of being revoked, the average number of keys to be invalidated

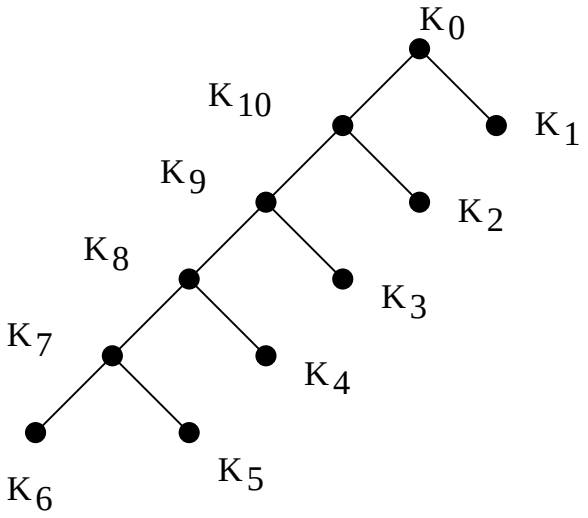

Figure 4.2: An Unbalanced Key Distribution

when a member is revoked is given by

$$\begin{aligned} \frac{3+4+\cdots+(N+2)+(N+2)}{N} &= \frac{N+1+N(N+1)/2}{N} \\ &= \frac{(N+1)(N+2)}{2N}. \end{aligned} \tag{4.2}$$

Hence, the average number of keys to be invalidated grows as $\mathcal{O}(N)$ in this model. In rooted trees from [10, 11, 17] the number of keys to be invalidated is of order $\mathcal{O}(\log_d N)$.

The key assignment problem in [10, 11, 17, 15, 13, 14] has been related to the number of members alone. The number of keys per member was assigned based on the observation that for N members $\log_d N$ keys are enough for a rooted tree.

We however will show that the problem of key assignment can be related to the physical process of member revocation and that it can be intimately related to a suitably defined "entropy" of member revocation event. We further demonstrate some interesting capabilities of this approach, including security weakness analysis. In order to do develop our formulation, we first define the terminology

and show that the well known Kraft inequality plays a critical role in compromise recovery.

4.2.1 Member Indexing

Let $X_{n-1}X_{n-2}\cdots X_1X_0$ be the binary index sequence representing N users. Following the conventional network terminology, we call this indexing the User Index (UID). In order for the GC to be able to revoke each member and invalidate the keys, the GC has to store a member index and the corresponding set of keys assigned to that member. Hence, UID for a member has to be in one-to-one correspondence with the set of keys assigned to that member. This requirement implies that each member should be indexed using the set of keys assigned to that member. We now define the Key Index (KID) in the following manner.

Definition: Key Index (KID) of a member i is defined as the string generated by concatenation of the keys assigned to the member i, taken in any order. If the number of keys assigned to member i is denoted by L_i, then there are $L_i!$ possible different sequences that can be generated using these L_i keys. All these KIDs are equivalent. Hence, the KID of a member is an equivalence class with $L_i!$ elements in it, where L_i is the number of keys assigned to member i.

M_1 in figure 4.1 has five KEKs and is represented by the string $K_O K_{2.1} K_{1.1} K_{0.1}$. Since there are 120 different ways to concatenate these keys, there are 119 additional strings generated by rearranging and concatenating the keys assigned to M_1.

Use of UID alone as in [10, 11, 15, 17] doesn't provide insights into the problems due to user collusion. The discussions on user collusion are presented in a later section due to its separate significance.

4.2.2 Unique Key Set Assignment and Kraft Inequality

At the time of member revocation, the GC has to be able to uniquely identify the set of keys assigned to the revoked member and invalidate the keys. After revoking a member, securely reaching the rest of the group requires that the valid member has one or more keys that are not in the set of keys assigned to the revoked member. We will call the ability of the GC to reach the valid members under some user(s) revocation as the *reachability* condition. Unlike other works that emphasize UID, we note that the KID plays a major role since it is the keys that need to be invalidated and (re)generated.

One important necessary condition for reachability to hold in the rooted tree based key assignment is that the KID of any member should not be a prefix *of the KID of any other member. On the rooted-tree, this leads to the well known Kraft inequality given below.*

Theorem 4.1. *Kraft Inequality for KID*

For a $d-ary$ rooted key tree with N members and KIDs satisfying the prefix condition, if we denote the number of keys for member i by l_i, the sequence $\{l_1, l_2, \cdots l_N\}$ satisfies the Kraft inequality given by

$$\sum_{i=1}^{i=N} d^{-l_i} \leq 1. \tag{4.3}$$

Conversely, given a set of numbers $\{l_1, l_2, \cdots l_N\}$ satisfying this inequality, there is a rooted tree that can be constructed such that each member has a unique KID with no-prefixing.

Proof: Presented in [1], and is not included here.

4.2.3 Limitations of Kraft Inequality

We now show why the Kraft inequality is only a necessary condition for reachability. Let A, B, and C be three members who have been assigned keys $\{K_1, K_2, K_3\}$, $\{K_1, K_2, K_3, K_5, K_6\}$, and $\{K_4, K_5, K_6\}$ respectively. For a binary tree these lengths satisfy the Kraft inequality since $(2^{-3} + 2^{-5} + 2^{-3}) = \frac{9}{32} < 1$. We note that if the member B is revoked, *all* the keys of member A are completely invalidated whereas the keys of member B will be only partially invalidated. If, on the other hand, member A is revoked, the GC can securely reach members B and C using any one of the keys from the set $\{K_5, K_6\}$. However, if the GC has to revoke members A and C simultaneously, all the keys of member B will be compromised. Although the set of keys assigned to member B is not a concatenation of keys of members A and C, all the keys assigned to B are contained in the set of keys assigned to members A and C. Hence, the condition that the KID of a member should not be a prefix for the KID of another member is not a sufficient condition for reachability. Moreover, this example shows that the choice of KIDs satisfying the Kraft inequality does not imply that the KID system is collusion resistant.

On the other hand, the KIDs satisfying the Kraft inequality do help to solve another important problem, namely the optimal key allocation per member. We present the needed formulation in the next section. This optimal assignment is very closely tied to the underlying physical process of member revocation as shown in the next section.

4.3 Probabilistic Modeling of Member Revocation

Since the key updates are performed in response to member revocation, statistics of *member revocation event*, are useful data for system design and performance characterization. Hence, the statistics of member revocation should be linked to the assignment of KID to a member. It may be noted that we are not making any claim about the specific selection of any key at this stage. We denote by p_i the probability of revocation of member i.

4.3.1 Relating the Probability of Member Revocation to the Keys on the Rooted Tree

The physical process of member revocation is related to the rooted trees via the leaf nodes using the following observations.

- Since each member in the rooted tree is assigned to a unique leaf, the probability of revocation of a member is equal to the probability of revocation of the corresponding leaf node.
- Since all the nodes of the rooted tree are assigned a unique key,the probability of revocation of leaf node is also the probability of revocation of the key represented by the leaf node.
- Hence, we note that the probability p_i of revoking member i is equivalent to having the probability p_i of revoking the key at the leaf i.

We can also derive additional properties that are more useful on the trees. For example, although the probability of revocation of any intermediate node key

is a composition of the probabilities of all the children nodes, the KIDs are sets uniquely associated with each member. Hence, the probability of revocation of a member is identical to not only the revocation of the leaf node key, but also the revocation of the set of keys assigned to that member, taken together as a set. The individual revocation probabilities of the keys may be different, and can be computed using basic formulae with some realistic assumptions.

The following assumptions are implicit in the models presented in [10, 17, 15] and are useful in the derivation of the optimal number of keys to be assigned to each member.

- *Assumption 1:* Revocation of members are mutually independent events.
- *Assumption 2:* The number of members N is a fixed quantity.

This assumption is restrictive and can at best satisfy only one temporal "snap shot" of the real world requirement. Implicit in this assumption is the property that the tree structure is fixed over the entire session. One way to remove this constraint is to set N as the maximal allowed number of members. In deriving the optimal number of keys to be assigned per member, we will assume that N represents the number of members in the group.

The assumption that the member revocation events are independent allows a simple computation of the probabilities of revocation of all the intermediate node keys on the tree. Let the branch k of an intermediate node i have the probability of revocation p_{ik}. If the individual member revocations are statistically independent, the following equation presents the probability of revocation p_i of the intermediate node i of a $d-ary$ rooted tree.

$$p_i = \sum_{k=1}^{k=d} p_{ik} \tag{4.4}$$

Hence, starting from the revocation probabilities of the leaf nodes, one can compute the probabilities of revocation of all the intermediate nodes. Using the recursive nature of the rooted tree structure, every probability of revocation of any key corresponding to an internal node can be expressed in terms of the probabilities of the member revocation. In particular, the the root key and the session key are revoked every time a member is revoked.

4.3.2 Defining the Entropy of Member Revocation Event

In physical processes that involve probabilistic modeling, one can often define the uncertainty of the occurrence of an event using a suitably defined entropy of the process. We will use Shannon entropy [1] to express the amount of uncertainty as to which member will be revoked. We first define the *entropy* of member revocation event.

Definition: We define the $d-ary$ entropy H_d of the member revocation event by

$$H_d = -\sum_{i=1}^{i=N} p_i \log_d p_i \tag{4.5}$$

where p_i is the probability of revocation of member i. As mentioned earlier, the entropy expresses the uncertainty as to which member will be revoked in $d-ary$ digits.

Since the member revocation event and the leaf node key revocation event are probabilistically identical, the entropy of the member revocation event is the same as the entropy of the leaf key revocation event.

Leaf Key Revocation Entropy: is the entropy or uncertainty as to which of the leaf keys will be revoked. Since the leaf key revocation probability is in one-to-one correspondence with the member revocation probabilities, the leaf key revocation

entropy is identical to the entropy of the member revocation event.

Hereafter we will use the term entropy of member revocation event instead of leaf key revocation entropy since they are equivalent. Another very useful observation is that since the member revocation event is also probabilistically equivalent to the KID revocation event, the entropy of member revocation event is identical to the entropy of the KID revocation event.

A main outcome of these observations is that the entropy of the KID revocation event is identical to, and can be completely characterized by the entropy of the leaf key revocation event (which is equivalent to the member revocation event).

With this probabilistic model, we show below that we can:

- Derive optimal number of keys per member.
- Analyze the collusion properties of some schemes.
- Derive a bound on the length of the keys.
- Determine if a given rooted key scheme can sustain its key generation rates

4.3.3 Assigning Optimal Number of Keys per Member

Since the SK and the root key are common to all the members, these two keys do not contribute to the optimization. We now show that the optimal key assignment on the rooted tree can be posed as an optimization problem.

Theorem 4.2. For a key assignment satisfying the Kraft inequality, the optimal average number of keys, excluding the root key and the SK, held by a member is given by the $d-ary$ entropy $H_d = -\sum_{i=1}^{i=N} p_i \log_d p_i$ of the *member revocation event*. For a member i with probability of revocation p_i, satisfying this

optimality, the optimal number of keys l_i, excluding the root key and the TEK, is given by

$$l_i^* = -\log_d p_i. \tag{4.6}$$

Proof:

The average number of keys held by the members, denoted by $\hat{l}$, given by:

$$\hat{l} = \sum_{i=1}^{i=N} p_i l_i. \tag{4.7}$$

Then the constrained problem is to minimize $\hat{l}$, subject to the constraint of the Kraft inequality given by

$$\sum_{i=1}^{N} d^{-l_i} \leq 1. \tag{4.8}$$

Using Lagrangian multipliers leads to the following convex cost function

$$C = \sum_{i=1}^{N} p_i l_i + \lambda(\sum_{i=1}^{N} d^{-l_i}) \tag{4.9}$$

where λ is Lagrange multiplier, and $\sum_{i=1}^{N} d^{-l_i} \leq 1$. Differentiating C by l_i leads to

$$\frac{\partial C}{\partial l_i} = p_i - \lambda d^{-l_i} \log d \tag{4.10}$$

where "log" with no base denotes the natural logarithm. Setting the derivative to zero, yields the optimal value of l_i as

$$p_i = d^{-l_i^*} \lambda \log d \tag{4.11}$$

Using the constraints on the probability and the Kraft inequality leads to

$$p_i \;\; = \;\; d^{-l_i^*} \tag{4.12}$$

$$l_i^* = -\log_d p_i \tag{4.13}$$

$$= \log_d(1/p_i) \tag{4.14}$$

We note

$$\frac{\partial^2 C}{\partial l_i \partial l_j} = \lambda d^{-l_i}(\log d)^2 = \delta_{i,j} p_i \log d \geq 0; \tag{4.15}$$

where, $\delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$

Since the second partials are positive, the optimal values of l_i's correspond to a global minimum value of the cost function C. Using the convexity of the cost function, this minima is indeed the global minimum of the cost function C. Hence, optimization of LKT provides the *minimal* number of average keys per member (excluding the SK and the root key), and this optimum value is given by

$$\begin{aligned} \hat{l}^* &= \sum_{i=1}^{i=N} p_i l_i^* \\ &= -\sum_{i=1}^{i=N} p_i \log_d p_i \\ &= H_d \end{aligned} \tag{4.16}$$

where $H_d = -\sum_{i=1}^{i=N} p_i \log_d p_i$ is the entropy of the member revocation event. Since the SK and the root key are common to all the members, optimal average number of keys per member is given by $H_d + 2$, and the number of keys assigned to member i with revocation probability p_i, including the SK and the root key is given by

$$l_i^* + 2 = -\log_d p_i + 2 = \log_d \frac{d^2}{p_i}. \tag{4.17}$$

The following properties that are very useful in identification of the minimal number of keys that can be used after member revocation are summarized in the form of the lemma below; the are also satisfied by the optimal number of keys held by a member.

Lemma 4.1.

1. If $p_i > p_j$, then $l_i(= -\log_d p_i) > l_j(= -\log_d p_j)$.

2. There must be at least two members with the largest number of keys.

3. Since the number of keys assigned per member needs to be integer, *true* average number of keys per member is more than the optimal value, and is not more than d additional keys.

In order to derive the last part of the lemma, we need the following definition from information theory [1].

Definition: The *relative entropy* or the *Kullback Leibler* distance between two probability mass functions $p(x)$ and $q(x)$ is defined as

$$D(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)}. \tag{4.18}$$

Sketch of the Proofs:

1. The logarithm being a monotone function, if $p_i > p_j$, then $\log_d p_i > \log_d p_j$. Hence $-\log_d p_i < -\log_d p_j$, leading to $l_i(= -\log_d p_i) < l_j(= -\log_d p_j)$.

2. If there are no two members with the largest number of keys, then we can reduce the largest number of keys held by at least one and still ensure that all members have unique sets of keys assigned. However this reduction will violate the proof of optimality of the individual codeword lengths. Hence, at least two members should be assigned largest number of keys.

3. In the earlier derivation we showed that the entropy is the point of optimality, and is indeed a global minimum of the average number of keys held by a member. Since the number of keys need to be integer, the average number of extra keys is given by

$$\begin{aligned}
\sum_{i=1}^{i=N} p_i l_i - H_d &= \sum_{i=1}^{i=N} p_i l_i + \sum_{i=1}^{i=N} p_i \log_d pi && (4.19)\\
&= \sum_{i=1}^{i=N} p_i \log_d \frac{p_i}{d^{-l_i}} && (4.20)\\
&= \sum_{i=1}^{i=N} p_i \log_d (\frac{p_i}{q_i}) + \log_d (\frac{1}{q_i}) && (4.21)\\
&= D(p||q) + \log_d (\frac{1}{q_i}) && (4.22)
\end{aligned}$$

where $q_i = \frac{d^{-l_i}}{\sum_{j=1}^{j=N} d^{-l_j}}$.

To show that difference is non-negative, we need to show that

$D(p||q) = \sum_{i=1}^{i=N} p_i \log_d(\frac{p_i}{q_i})$ and $\log_d(\frac{1}{q_i})$ are non-negative. From the Kraft inequality, $\sum_{i=1}^{i=N} d^{-l_i} \leq 1$, we have $\frac{1}{\sum_{i=1}^{i=N} d^{-l_i}} \geq 1$. Hence, $\log_d \frac{1}{\sum_{i=1}^{i=N} d^{-l_i}} \geq 0$.

To show that the term $D(p||q) \geq 0$, we need the following known lemma.

Lemma 4.2. Let $\{a_i\}_{i=1}^{i=N}$, and $\{b_i\}_{i=1}^{i=N}$ be two sets of non-negative quantities. Then

$$\sum_{i=1}^{i=N} a_i \log_d \frac{a_i}{b_i} \geq (\sum_{i=1}^{i=N} a_i) \log \frac{\sum_{i=1}^{i=N} a_i}{\sum_{i=1}^{i=N} b_i} \qquad (4.23)$$

with equality being achieved *iff* $\frac{a_i}{b_i} = \text{constant}$.

Proof: Let $y = \log t$. Then the tangent to the curve y at point $(t', \log t')$ is given by the equation $y - \log_d t' = \frac{t-t'}{t'}$. Except at the point of tangent, the curve $y = \log t$ is below the line $y = \frac{t-t'}{t'} + \log t'$. At the point of tangent,

we have $t = t'$. Setting $t = \frac{b_i}{a_i}$, and $t' = \frac{\sum_{i=1}^{i=N} b_i}{\sum_{i=1}^{i=N} a_i}$, leads to

$$\log \frac{b_i}{a_i} \leq \frac{(\frac{b_i}{a_i} - \frac{\sum_{i=1}^{i=N} b_i}{\sum_{i=1}^{i=N} a_i})}{\frac{\sum_{i=1}^{i=N} b_i}{\sum_{i=1}^{i=N} a_i}} + \log \frac{\sum_{i=1}^{i=N} b_i}{\sum_{i=1}^{i=N} a_i} \tag{4.24}$$

$$a_i \log \frac{b_i}{a_i} \leq (b_i \sum_{i=1}^{i=N} a_i - a_i \sum_{i=1}^{i=N} b_i) + a_i \log \frac{\sum_{i=1}^{i=N} b_i}{\sum_{i=1}^{i=N} a_i} \tag{4.25}$$

$$\sum_{i=1}^{i=N} a_i \frac{b_i}{a_i} \leq \sum_{i=1}^{i=N} a_i \log \frac{\sum_{i=1}^{i=N} b_i}{\sum_{i=1}^{i=N} a_i} \tag{4.26}$$

$$\sum_{i=1}^{i=N} a_i \frac{a_i}{b_i} \geq \sum_{i=1}^{i=N} a_i \log \frac{\sum_{i=1}^{i=N} b_i}{\sum_{i=1}^{i=N} a_i} \tag{4.27}$$

observing that at the point of equality $t = t'$ leads to, $\frac{a_i}{b_i} = \frac{\sum_{i=1}^{i=N} a_i}{\sum_{i=1}^{i=N} b_i}$.

Using the above given lemma and noting that $q_i = \frac{d^{-l_i}}{\sum_{i=1}^{i=N} d^{-l_i}}$, we check that

$$D(p||q) = \sum_{i=1}^{i=N} p_i \log \frac{p_i}{q_i} \geq (\sum_{i=1}^{i=N} p_i) \log \frac{\sum_{i=1}^{i=N} p_i}{\sum_{i=1}^{i=N} q_i} = 1 \log \frac{1}{1} = 0 \tag{4.28}$$

Equality holds *iff* $\frac{p_i}{q_i} = \frac{\sum_{i=1}^{i=N} p_i}{\sum_{i=1}^{i=N} q_i} = 1$. Hence, $q_i = \frac{d^{-l_i}}{\sum_{i=1}^{i=N} d^{-l_i}}$ is the probability which optimizes the average number of keys assigned per member.

In summary, the optimal key allocation strategy requires that the member with revocation probability p_i be assigned $\boxed{\log_d \frac{d^2}{p_i}}$ number of keys. In the case of binary rooted trees, the optimal number of keys for a member with probability of revocation p_i needs to be assigned $\log \frac{4}{p_i}$ keys.

The results indicating that there are at least two members with the largest KID also indicate that the tree is *packed*. i.e., if a member is revoked, all the complementary keys of that member are needed to securely reach the rest of the members. If there are bulk member removals, the set of keys that are in the complementary set of the revoked members can be used to securely update the

valid members. Depending on the nature of the specific key choices it is possible to develop a fast algorithm for key updates.

4.3.4 Maximum Entropy and the Key Assignment

The results reported in [10, 11, 13, 17, 15] present rooted trees with all member having the same number of keys. Since the optimal number of keys for a member i with probability of revocation p_i is $\log_d \frac{d^2}{p_i}$,this assignment is equivalent to treating $\log_d \frac{d^2}{p_i}$ = constant for all values of i. Hence, the results in [10, 11, 17, 15, 13] assume that the probability of revocation is uniform for the entire group. Since the uniform distribution *maximizes* the entropy and entropy is the average number of keys per member under the optimal strategy, the schemes in [10, 11, 13] assign maximal set of keys per member. We summarize these results by the following theorem.

Theorem 4.3. Since the entropy of member revocation $H_d = -\sum_{i=1}^{i=N} p_i \log_d p_i$ is maximized when the revocation probabilities are uniform and the schemes in [10, 11, 17, 15, 13] implicitly assume uniform member revocation probabilities, these schemes correspond to the worst case key assignments for individual members. These schemes assign $\log_d Nd^2$ keys per member, where N is the group size.

We note here that the use of maximal number of keys per member does not imply that the key distribution scheme is free of any possible member collusion or even secure. We elaborate on this point later. Next we derive the explicit bounds on the optimal number of keys per member.

4.3.5 Upper Bounds on the Integer Values of keys

The optimal number of keys for a member with probability of revocation p_i in a $d-ary$ rooted tree key management scheme was shown to be

$$\log_d \frac{d^2}{p_i}. \tag{4.29}$$

Since this quantity corresponds to the number of physical keys, it has to be an integer value. The following theorem summarizes the bound on the optimal number of keys to be held by a member. If we denote the integer value of the average number of keys, excluding the SK and the root key, held by members by $\hat{l}^*$, the bounds on the optimal number of keys per member are given by the following inequalities

Theorem 4.4. The optimal average number of keys held by a member satisfies

$$H_d + 2 \leq \hat{l}^* + 2 < H_d + 3. \tag{4.30}$$

Proof: Using the notation $\lceil -\log_d p_i \rceil$ to represent the smallest integer greater than or equal to $-\log_d p_i$, we have the integer value of l_i^* as

$$l_i^* = \lceil -\log_d p_i \rceil. \tag{4.31}$$

For this choice of l_i, we have

$$-\log_d p_i \leq l_i < -\log_d p_i + 1 = \log_d \frac{d}{p_i} \tag{4.32}$$

For this value of l_i^*, the Kraft inequality is still satisfied since

$$\sum_{i=1}^{i=N} d^{(-l_i^* = -\lceil -\log_d p_i \rceil)} \leq \sum_{i=1}^{i=N} d^{-\log_d p_i} = \sum_{i=1}^{i=N} p_i = 1. \tag{4.33}$$

Multiplying equation (4.33) by p_i and summing over i leads to

$$-\sum_{i=1}^{i=N} p_i \log_d p_i \leq \sum_{i=1}^{i=N} p_i l_i \leq -\sum_{i=1}^{i=N} p_i \log_d p_i + \sum_{i=1}^{i=N} p_i \tag{4.34}$$

$$\Rightarrow H_d \leq \hat{l}^* < H_d + 1 \tag{4.35}$$

$$\Rightarrow H_d + 2 \leq \hat{l}^* + 2 < H_d + 3 \tag{4.36}$$

Since the average number of keys per member is $(\hat{l}^* + 2)$, we note that the *optimal* number of average keys per member is at most 3 $d-ary$ digits more, and is at least 2 $d-ary$ digits more than the *entropy of the member revocation event*.

4.3.6 Effect of Using Incorrect Entropy on Key Length

In figure 4.2 we presented the effect of an unbalanced rooted tree on the number of keys to be assigned and to be invalidated. We note that this quantity can be completely characterized using basic results from information theory as well.

Lets us assume that the true revocation probability of member i is p_i and the used probability of revocation for member i is q_i. Hence, the optimal number of keys to be assigned to that member is given by

$$l_i^* = \lceil -\log_d q_i \rceil \tag{4.37}$$

Using an incorrect distribution introduces redundancy in the number of keys that are assigned to the members. This redundancy is given by the following theorem.

Theorem 5. The average number of keys per member under the true distribution p with the number of key selection based on $l = -\log_d q_i$ satisfies the following bounds

$$H_d(p) + D(p||q) \leq L < H_d(q) + D(p||q) + 1. \tag{4.38}$$

Proof: Upper bound is derived as:

$$\begin{aligned} L - H_d(p) &= \sum_{i=1}^{i=N} p_i(\lceil -\log_d q_i \rceil + \log_d p_i) \\ &< \sum_{i=1}^{i=N} p_i(-\log_d q_i + \log_d p_i + 1) \\ &= \sum_{i=1}^{i=N} p_i(\log_d \frac{p_i}{q_i} + 1) \\ &= D(p||q) + 1 \\ &= L < H_d(p) + D(p||q) + 1. \end{aligned} \tag{4.39}$$

The lower bound is derived as:

$$\begin{aligned} L - H_d(p) &= \sum_{i=1}^{i=N} p_i(\lceil -\log_d q_i \rceil + \log_d p_i) \\ &\geq \sum_{i=1}^{i=N} p_i(-\log_d q_i + \log_d p_i +) \\ &= \sum_{i=1}^{i=N} p_i(\log_d \frac{p_i}{q_i}) \\ &= D(p||q) \\ &= L < H_d(p) + D(p||q). \end{aligned} \tag{4.40}$$

Hence, on average the number of redundant keys assigned to a member due to the use of an incorrect distribution is given by the inequalities (4.38).

Apart from being closely related to the optimal key assignments, the entropy of member revocation event is also related to the sustainable key length of the secure multicast group, as shown next. To our knowledge, result of this kind is not available in the literature of key length selection.

4.3.7 Bounds on Average Key Length

When there is a member revocation, the average number of keys to be invalidated is given by $(2 + H_d)$. If each key is L digits long, then in order to update these keys, the total number of digits that need to be generated by the GC after member revocation is $L(2 + H_d)$ digits. Since $H_d \leq \log_d N$ with equality attained *iff* all the members have equal revocation probabilities, the hardware need to be able to generate an average of $L \log_d(Nd^2)$ digits within the next unit of time of update to let the session continue. The following theorem summarizes this result.

Theorem 4.6. For a $d-ary$ rooted tree key distribution scheme in which each key is of length L digits, if the hardware digit generation rate is given by B, then the key length is bounded above by $\boxed{L \leq \frac{B}{H_d+2}}$ on average and the maximally allowable key length is bounded by $\boxed{L \leq \frac{B}{\log_d(Nd^2)}}$. Considering individual members, the key length is bounded below and above by $\boxed{\frac{B}{\log_d \frac{d^2}{p_{max}}} \leq L \leq \frac{B}{\log_d \frac{d^2}{p_{min}}}}$.

Proof: As shown earlier, the average number of keys to be generated in the event of member revocation is given by $\boxed{(2 + H_d) = 2 + \sum_{i=1}^{i=N} p_i l_i}$. Hence, the hardware should be able to generate a total of $L(H_d+2)$ digits of *suitable quality*[1] in unit of time to let the session continue without delays in the average sense. Hence the hardware digit generation rate B must satisfy $B \geq L(H_d+2)$. Observing that the entropy is maximal under uniform distribution and the maximal value is given by $H_d \leq H_d(U) = \log_d N$ leads to the bound $L \leq \frac{B}{\log_d(Nd^2)}$ with equality *iff* all the members have the same revocation probabilities. The minimal and maximal allowed key lengths are decided by the maximal and the minimal member revocation probabilities p_{max} and p_{min} and satisfy $\boxed{\frac{B}{\log_d \frac{d^2}{p_{max}}} \leq L \leq \frac{B}{\log_d \frac{d^2}{p_{min}}}}$.

Since it is of interest to make sure that the system sustains the secure com-

[1]Based on the application specific use of the key.

munication mode, one strategy is to design the system so that it satisfies the worst case scenario. Hence the hardware digit generation rate B needs to satisfy $\boxed{B \geq L \log_d \frac{d^2}{p_{min}}}$.

In summary, it was shown that the entropy of member revocation event plays an important role in deciding the key length if the system were to update the keys each time a member is revoked.

4.4 Security Analysis of Recent Results Using Member Revocation Entropy

The authors in [15] noted that given the binary index of a member, each bit in the index takes two values, namely 0 or 1. To follow the example given in [15], when $N = 16$, $\log_2 16 = 4$ bits are needed to uniquely index *all* 16 members. The authors then proceeded to claim that since each bit takes two values, it can be *symbolically* mapped to a distinct pairs of keys. The table below reproduces the mapping between the ID bit # and the key mapping for the case in [15] for $N = 8$:

ID Bit #0	$K_{0.0}$	$K_{0.1}$
ID Bit #1	$K_{1.0}$	$K_{1.1}$
ID Bit #2	$K_{2.0}$	$K_{2.1}$

where, the key pair $(K_{i,0}, K_{i,1})$ symbolically represents the two possible values of the *ith* bit of the member index. Although this table does provides a one-to-one mapping between the set of keys and the member index using only eight keys, the problem with this approach becomes clear if we map the table to the rooted tree structure. Figure 2 shows the mapping of the keys on the tree. (For the

sake of clarity, not all the keys corresponding to the leafs are shown in figure 2). Adjacent leafs have K_{20}, K_{21} as the keys and this pair is repeated across the level. In fact, at any depth only two specific keys have been used and duplicated across the depth.

In approaches such as [17, 15] that use UID to optimal Huffman coding, a special case of member revocation brings these key management schemes to halt, by collusion. This happens if the members M_0 and M_7 need to be revoked. The corresponding keys to be revoked are shown in figure 4.3. These two members have only the session key in common. However, if these two members need to be simultaneously revoked, the group controller is left with *no key* to securely communicate with the rest of the valid members. This reduces the rooted tree to the GKMP [7]. The compromise recovery for this case requires that the entire group re-key itself by contacting one member at a time.

The key assignments in [17, 15] and their variations also allow the members to collaborate or collude and break the system. We now discuss user collusion on the rooted tree in [17, 15].

4.4.1 Impact of Member Collusion on Rooted Trees

We showed that if more than one member were to be revoked, the whole key scheme may be compromised. There are three different ways to interpret the collusion problems with approaches in [15, 17] based on rooted trees. We present them in the order of generality:

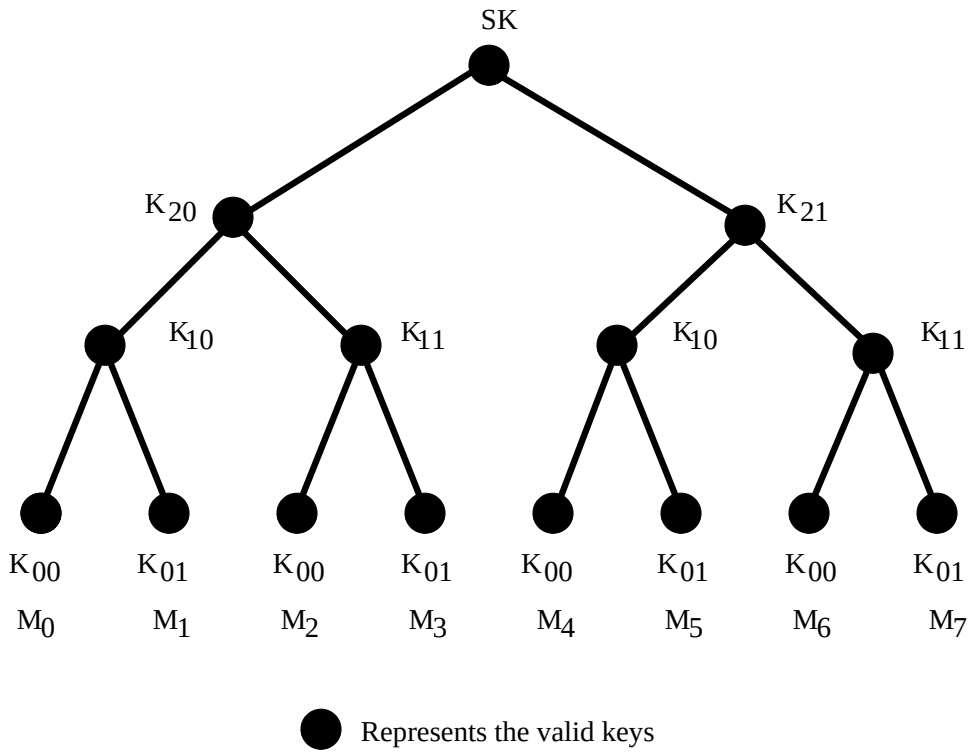

Figure 4.3: The Key Distribution in [17, 15]

Interpretation based on Minimal number of Key Requirements

A simple way to interpret the shortcomings of the results in [15, 17] is to note that $2\log_2 N < N$, if $N > 4$. In order to prevent member collusion from being able to break the rest of the system, there must be at least N keys so that each member has a unique key and can be contacted at the time of member revocation. Since $2\log_2 N < N$ (for $N > 4$) is the number of distinct keys used by the variation of rooted tree presented in [15, 17], such a scheme can be completely or partially compromised depending on the colluding members. However, when $N = 4$, $2\log_2 N = 4$. Hence, in order to be able to reach any valid members securely, the key distribution has to be a degenerate multicast as shown in figure 4.4.

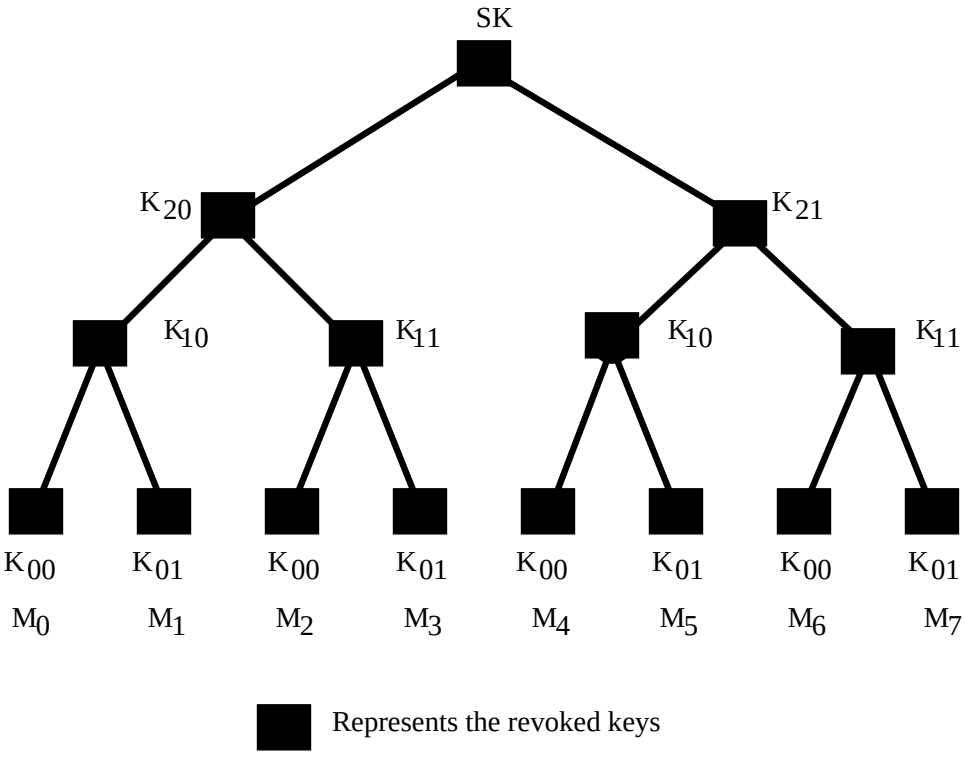

Figure 4.4: Revocation of Members M_0, M_7 in [15, 17].

Interpretation Based on Complementary variables

The third interpretation is based on the notion of sets and includes a larger definition of collusion discussed under the category of complementary variables in [10]. The approach in [15, 17] is a special case of the complementary variable approach. If the secure group membership is a set such that every member is denoted by a unique key and that key is given to all other members but the member itself, at the time the member is to be revoked, all other members can use the key denoting the revoked member as the new key. For a set of N members, all the members will have $(N-1)$ keys that correspond to other members and no member will have the key denoting itself. Clearly, if two members collude, between them they will have *all* the future keys of the group. Hence, this kind of

key assignment does not scale beyond 2 members.

Interpretation based on Huffman Coding

We showed that the average number of keys per member is given by entropy. We also showed that if the distribution is uniform, the average number of keys per member attains its maximum value. When the member revocation probabilities are equal, the number of keys assigned to a member is same as the average number of keys per member. We also showed that this strategy is used in [10, 11, 17, 15].

The schemes in [15, 17] mapped the UIDs to KIDs directly. Since the number of bits needed for N members is $\log_2 N$, the schemes in [15, 17] used a unique pair of keys to symbolically map each bit position of the the member index. Hence, a total of $2\log_2 N$ keys are used to uniquely represent each member index. This selection of keys can create a set of N unique indices and the codewords generated by concatenating $\log_2 N$ keys satisfy the Kraft inequality. Hence, this mapping of a unique pair of keys to each bit location corresponds to performing a Huffman coding with $2H_2(U)$ distinct keys, where $H_2(U) = \log_2 N$. However, the problem with Huffman coding is that it is uniquely decodable!. Hence, a key assignment based on direct mapping of bit location to keys will lead to serious security exposure. In fact, an attacker can break the whole system by breaking the members whose indices are *all* ones and *all* zeros. These two members represent all possible bit patterns and hence have all the $2\log_2 N$ keys among themselves.

If we use the notation $(k_j, \hat{k}_j)$ to denote the unique key pair representing the two possible binary values taken by the jth bit, we note that the collusion or compromise of two members holding keys k_j and $\hat{k}_j$ respectively will compromise the integrity of the key pair $(k_j, \hat{k}_j)$. The following lemmas summarize our

observations:

Lemma 4.3. If the binary rooted key tree uses Optimal Huffman Coding for assigning members a set of keys based on $2\log_2 N$ $(N > 4)$ (here N is dyadic) distinct keys as in [15, 17], the whole system can be broken if any two members whose "codewords" (and hence indices) are one's complement of each other collude or are compromised. Hence, the integrity systems in [15, 17] do not scale beyond 4 members in the presence of colluding members.

In a $D-ary$ tree, each digit takes D values and the sum of these values is given by $\frac{D(D-1)}{2}$. Hence, if a set of k ($k \geq D$) members whose ith bit values when summed lead to $\frac{D(D-1)}{2}$ collude, they will be able to fully compromise the ith bit location. This result is summarized by:

Lemma 4.4. For a $D-ary$ tree with N members, the key corresponding to bit location b will be compromised by a subset of k $(k \geq D)$ members whose symbolic value of the bit location b denoted by the set $\{b_1, b_2, \cdots, b_k\}$ satisfy $\boxed{b_1 + b_2 \cdots b_k \equiv 0 \bmod \frac{D(D-1)}{2}}$.

4.4.2 On Generating a Large Class of Key Management Schemes with Varying Degree of Collusion

From our analysis of the tree based schemes, we note that many different key management schemes with different levels of protection against the user collusion can be made. On one extreme, the keys representing the rooted tree have no relationship, leading to a very high degree of integrity but also higher storage requirements. On the other extreme, all members share the same keys as in GKMP [7] leading to the system failure in the event of a single member failure. The schemes in [15, 17] fail with the collusion of two members or can fail at

different bit level depending on the index of the colluding members. Depending on how many digit locations are represented as $k - ary$ digits. The figure 4.5 shows the comparison between various schemes.

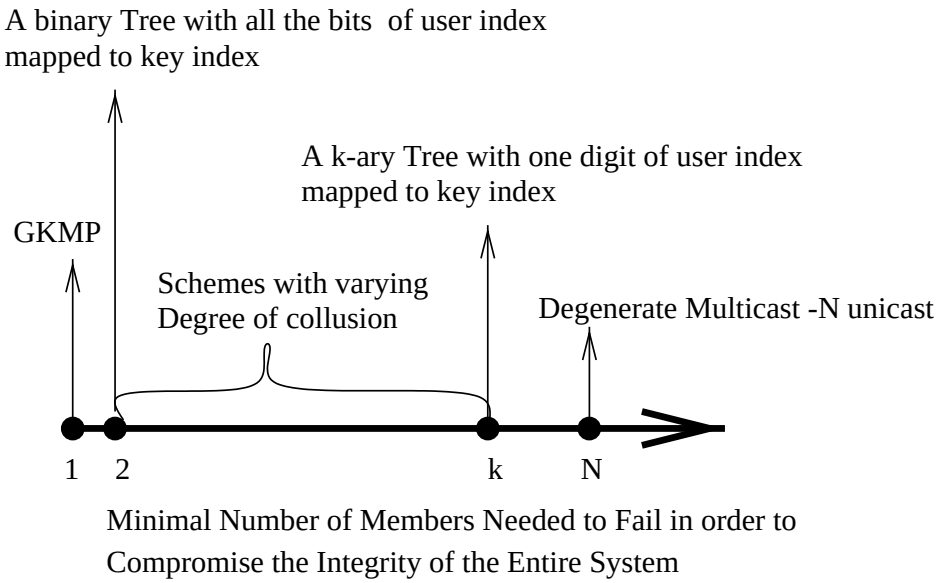

Figure 4.5: Effect of User failure of different schemes

Chapter 5

Oneway Functions for Keys

The previous chapter dealt with the design and analysis of rooted tree based key distribution schemes that will optimize the key storage requirements and communication overhead. In doing so, no assumption was made about the mechanism by which the keys are regenerated or distributed to the group.

In the recent past, there have been attempts making use of *cryptographically strong* functions to further reduce the amount of computations and the key storage requirements at the GC. In particular, McGrew and Sherman [14] proposed a rooted tree based key distribution approach that can deal with member join or removal based on properties of *oneway* functions. Canetti coauthored two papers which relied on the *cryptographic* strength of *pseudo random* functions and provided (a) efficient key update [13] under member removal, and (b) efficient communication key storage requirement [16] with *sub-linear* storage requirements.

In [16] Canetti et al. stated that the optimality or the lack of it for their scheme was not provable in the paper and posed it as an open problem. We resolve it by showing that the storage provided by them is not optimal and that it is a specific point along a cost function that is a hyperbola. We also show

that the worst case optimal strategy is related to *maximum entropy* even under their setup. This result, though obvious from the elementary information theory point of view, shows that while clustering members into groups, the worst case optimal strategy is to group members of the clusters so that the uncertainty as to which cluster will need key update should be roughly equal. We now present the current approaches and analyze them

5.1 Oneway function tree of McGrew and Sherman

McGrew and Sherman modified the rooted tree of Wallner et al [10] using oneway functions for explicitly computing the updated keys of the members. In order to illustrate their approach, we consider a rooted binary tree of depth three. This tree supports eight members and is shown in figure 5.1. The member M_3 is being revoked and the key update for member M_4 using oneway functions approach of McGrew and Sherman [14] is demonstrated here. In this scheme, each node n is associated with two keys, a node key k_n and a blinded node key k'_n. The blinded node key is computed from the node key using *one-way function* $g(.)$ as $k'_n = g(n)$.

In order to generate the blinded node keys, the GC chooses fresh random keys and assigns a unique key to each *leaf node.* Since each leaf node is assigned to a unique member, the leaf key is also the individual member key. From the previous chapter, we note that the *entropy of member revocation is same as the entropy of the leaf key revocation.* The internal node key for node n is computed

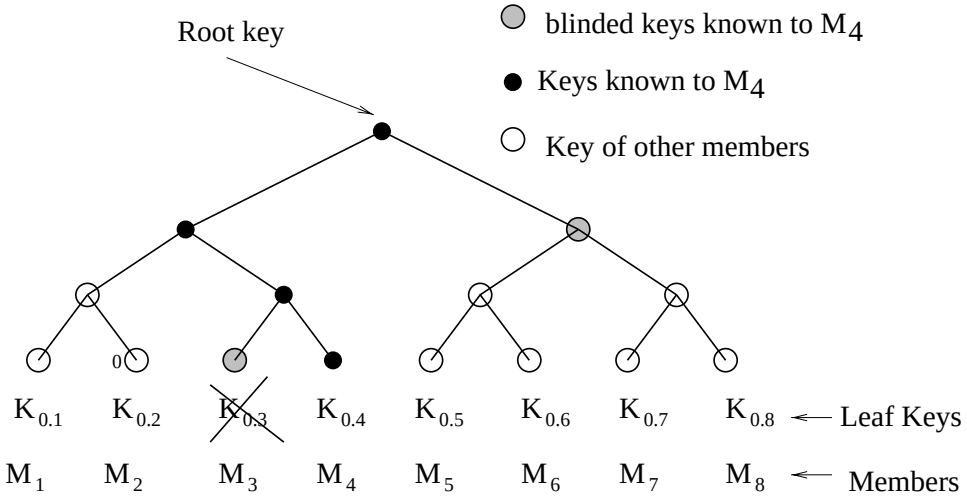

Figure 5.1: Key Update Process in [14].

using the formula

$$k_n = f(g(k_{left(n)}), g(k_{right(n)})) \tag{5.1}$$

where f is a *mixing* function, g is a one-way function, $k_{left(n)}$ and $k_{right(n)}$ are the keys of the left and the right children of node n. For example, f can be an *XOR* function.

McGrew and Sherman's construction requires that the following invariant is preserved by the key generation procedure

System Invariant. *Every member knows the unblinded node keys of all the nodes on the path from its leaf to the root, blinded node keys that are siblings to its path to the root, and no other blinded or unblinded keys.*

The intermediate keys are computed using the following steps.

- Every member is given its leaf key, and all the relevant sibling blinded node keys.

- Every member independently computes its blinded keys of the nodes along

the path to the root.

- Every member computes the relevant internal node keys.

If any of the blinded key changes, then the corresponding siblings should be updated with the new blinded key.

5.1.1 Computations under Addition or Deletion of Members

When a member is deleted or revoked, the node keys and the blinded node keys along the path from the leaf assigned to the member to the root are invalidated. If n is a node for which the blinded node key and the node key were revoked, and m is the sibling(s) of n, all the descendants of m need to be given the *new* blinded node key of n. If the member deletion or revocation leaves only one child for a parent, the child member is moved up to the parent node.

If a new member is added to a node n, node n is split, and two children are generated. In order to prevent any other member from colluding and compromising the keys, the new children leafs are given new set of leaf keys. All the relevant blinded keys are given to the members whose path from their leafs to the root are sibling to the newly created node path.

5.1.2 Summary of McGrew-Sherman Approach

Although the approach proposed by McGrew and Sherman can reduce the amount of communication overhead at the GC, the security of the new key generation scheme can't be proven rigorously or reduced to the security of any primitive.

However, we do note that at this stage there doesn't seem to be any obvious weakness in the scheme either.

An alternate method proposed by Canetti et al can be used to update the keys when a member is revoked and is described below.

5.2 Worst Case Member Revocation

We again consider a binary tree of depth three. This tree can support a group of eight members. In order to illustrate the method, we consider the case of revoking member M_1. Figure 5.2 illustrates the revocation of M_1. The keys $\{K_O, K_{2.1}, K_{1.1}, K_{0.1}\}$ are to be invalidated while revoking member M_1, and the keys $\{K_O, K_{2.1}, K_{1.1}\}$ need to be updated for the relevant members. From the figure 5.2, a member M_i needs to update the keys corresponding to the internal nodes that are common to M_i and M_1. For example, members M_5, M_6, M_7, M_8 share the root key with M_1, and need to update it after revocation of member M_1.

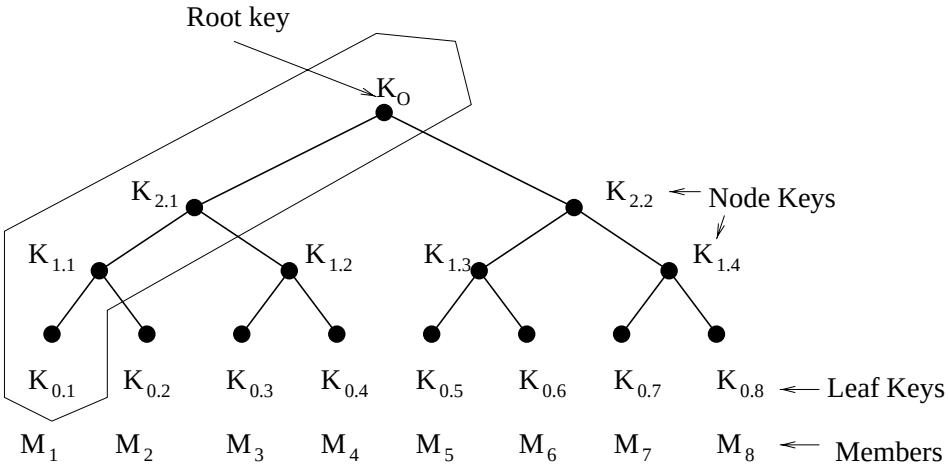

Figure 5.2: Deletion of M_1 in Rooted-Tree of [10].

As noted earlier, the tree structure allows the GC to update multiple members simultaneously. In figure 5.3, members that can be grouped together are marked. In [10], update of the keys is shown along the tree. As before, we use the notation $\{m\}_K$ to denote the encryption of message m using the key K. In removing M_1, the following messages are encrypted and transmitted: $\{K'_{11}\}_{K_{02}}$, $\{K'_{21}\}_{K'_{11}}$, $\{K'_{21}\}_{K'_{12}}$, $\{K'_{O}\}_{K'_{21}}$, and $\{K'_{O}\}_{K_{22}}$.

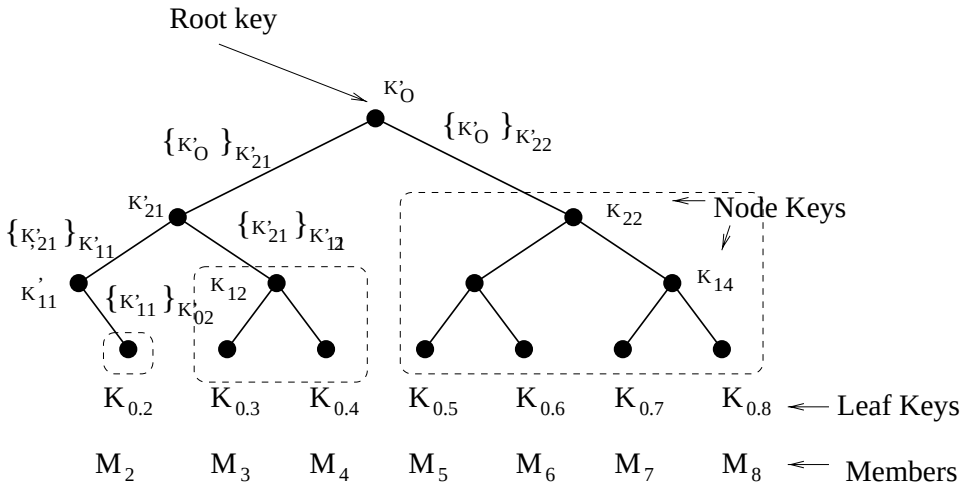

Figure 5.3: Key Update After revocation of M_1 [10].

The number of encryption needed is given by $2\log N$ for the binary tree with depth $\log N$. We now present the method by Canetti et al [13], that *reduces the communication overhead* to $\log N$ instead of $2\log N$.

5.3 A Scheme for Reducing Communication Overhead

In [13], Canetti et al presented an efficient key update method based on pseudo-random functions. The security of their scheme can be reduced to the strength or

the security of the pseudo-random function used in the computation. We describe their method below. Description of the method can also be found in [13, 16].

Figure 5.4 shows the type of update performed using the pseudo-random function. In order to prevent any obvious security weakness, the pseudo-random function $G(.)$ is chosen so that it doubles the size of the input. The output string is then split into two strings of equal length, denoted as $L(.)$, and $R(.)$. If the input is denoted by x, then $G(x) = L(x)R(x)$, where $L(x)$ and $R(x)$ are the *left* and *right* strings of the output.

When a member is to be revoked, the GC chooses a *secret key* or a *random seed* r, and assigns it to the parent node of the leaf node to which the revoked member was assigned. Every internal node n along the path from the root to the leaf assigned to the revoked member is assigned a unique fresh value r_n. The values of r_n, and the relevant node keys are computed in a recursive manner from the bottom of the tree to the top as described below.

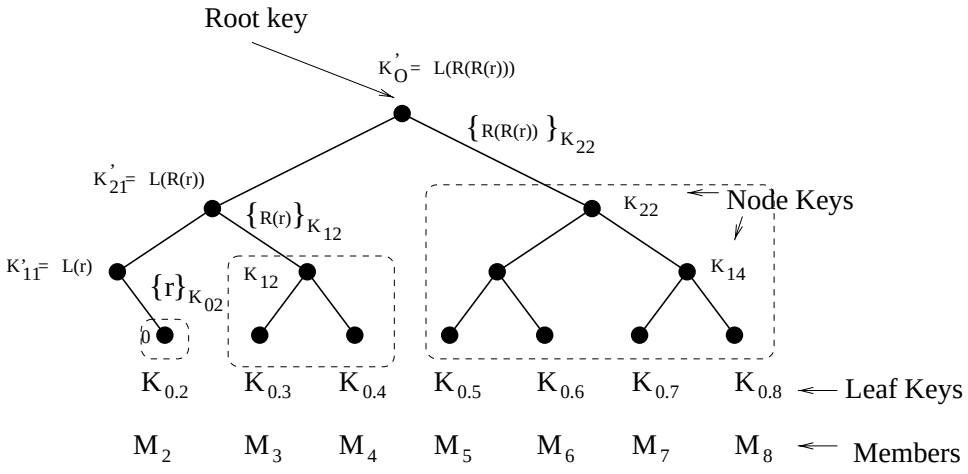

Figure 5.4: Key update process using pseudo-random functions [13].

Without loss of generality, we illustrate the update procedure for revocation of M_1.

1. The GC chooses a fresh random quantity r, encrypts, and sends $\{r\}_{K_{02}}$ to M_2.

2. The member M_2 computes $G(r) = L(r)R(r)$, and locally computes the parent node key $K^{'}{}_{11} = L(r)$.

3. The GC then encrypts $R(r)$, and send $\{R(r)\}_{K_{12}}$ to M_3 and M_4.

4. The members M_2, and M_3 independently compute the next level internal node key $K^{'}{}_{21}$ as $K^{'}{}_{21} = L(R(r))$.

5. The GC then encrypts and distributes $R(R(r))$ to M_5- M_8 as $\{R(R(r))\}_{K_{22}}$.

6. All the members can now compute the root key as $K^{'}{}_{O} = L(R(R(r)))$.

We note that the algorithm can be generalized to a $d - ary$ tree without difficulty.

The general procedure for revoking M_1 is summarized (any encryption and distribution is assumed to be done by the GC only)

1. Choose a fresh random quantity r and assign it to the leaf node that is revoked.

2. Encrypt and distribute the value of r to the siblings of the revoked member.

3. For $i = 0 \cdots h$, (h is the depth of the tree) repeat all the steps below.

4. Compute $G(R^i(r)) = L(R^i)(R^{i+1})$.

5. Compute the key of the internal node i (excluding the revoked leaf) $K^{'}{}_{i1} = L(R^{i-1}(r))$.

6. Encrypt and distribute $\{R^i(r)\}_{K_{i2}}$ to members who need that value, and have not been provided with any value earlier.

Invoking the properties in [33], it was noted that $G(.)$ being a pseudo-random function, repeated applications of it to the input r will make it difficult for the GC to set the root key to be from the weak key space. i.e., if $G(.)$ is pseudo-random, output of $G^l(r)$ is difficult to predict. Hence, iff $G(x) = L(x)R(x)$ is a cryptographically strong pseudo-random function generator, it will be hard to choose k such that $k = L(R \cdots (r) \cdots)$. Canetti et al. claimed that this property of the pseudo-random functions will prevent the GC from choosing weak keys. *We note that the argument assumes that the initial key assignment doesn't include any weak keys.* Else, the computation of the keys along the tree is somewhat complicated than that is presented by Canetti et al.

5.3.1 Use of Pseudo-random Functions in Storage Reduction

The GC can minimize the key storage requirements by generating the member specific keys as outputs of a pseudo-random function with indexing. In this scheme [16], the GC holds a single secret key r, an index to a pseudo-random function f_r [16]. The leaf key of member M_i is generated by $k_i = f_r(i)$. When a user is compromised, the GC computes the new session key SK, encrypts the new SK with the individual keys of each valid member and distributes. The security of this key generation scheme is based on the security of the pseudo-random function and the encryption scheme used. For security reasons, if a single member is compromised, the whole membership has to be updated with communication overhead of $\mathcal{O}(N)$.

On the other hand, the rooted trees need only $\mathcal{O}(\log N)$ communication messages to be used to update the keys. Hence, a mix model of key distribution was proposed in [16] to minimize the key storage requirements at GC. In this model, the group is divided into clusters of size M. Each cluster is assigned to a unique leaf of a $d-ary$ tree. Within the clusters, a cluster specific pseudo-random function is used to generate member specific leaf keys.

The model can be summarized in the following steps: Given a group of size N,

1. form clusters with fixed size M.

2. build a $a-ary$ rooted-tree with depth

$$b = \lceil \log_a \lceil \frac{N}{M} \rceil \rceil \tag{5.2}$$

3. assign each cluster to a unique leaf node of the $a-ary$ rooted-tree of depth, denoted by b.

4. use a pseudo-random function generator with cluster specific seed and generate the member specific keys.

Once the $a-ary$ rooted-tree of depth b is constructed, the key distribution is done using the technique in [10].

Communication-Storage Parameters Using the minimal storage scheme in conjunction with the rooted-tree, the user storage, the GC storage, and the number of encryptions needed can be computed analytically. We present the results with two specific examples in the table below.

	general M, a	Example 1	Example 2
user storage	$\log_a(\frac{N}{M})$	$\mathcal{O}(\log n)$	2
GC storage	$\frac{N}{M}\frac{a}{(a-1)}$	$\mathcal{O}(\frac{n}{\log n})$	$n^{0.5}+1$
Encryptions	$(M-1)+(a-1)\log_a(\frac{N}{M})$	$\mathcal{O}(\log n)$	$2n^{0.5}-2$

Table 5.1: Parameters of the tradeoff scheme in [16]. Setting $a = 2$, $M = 1$ leads to results in [10]. In example 1, $a = 2$, $m = \mathcal{O}(\log n)$, in example 2, $a = m = n^{0.5}$

5.3.2 Problem Posed

Canetti et al noted that the choice of $M = \log N$ led to the key storage requirement $\mathcal{O}(\frac{n}{\log_a N})$ which is sub-linear in N. They conjectured [16] to be optimal with communication overhead of $\mathcal{O}(\log N)$, and posed it as an open problem.

5.3.3 Answer to the Problem

We note that the answer to the open problem is *NO*. Proof that the choice of M they proposed does not lead to minimum storage can be presented using the following direct computations. The total number of keys to be stored by the GC. The key distribution structure in [16] has an $a-ary$ rooted-tree with depth b followed by a cluster of M members at each leafs. The total number of keys excluding the SK, is given by

$$\begin{aligned} S &= \sum_{i=0}^{b} a^i && (5.3) \\ &= \frac{a^{b+1}-1}{a-1} \\ &= \frac{aN-M}{M(a-1)} && (5.4) \end{aligned}$$

Setting $a = 2$, $M = 1$ leads to the familiar result in [10] for a binary rooted-

tree in the case that it has (2n -1) keys excluding the SK. The general form of the storage as a function of M is a hyperbola given by the equation

$$S = \frac{\lambda N}{M} + \mu \tag{5.5}$$

where λ, μ are scalar constants.

The result of sub-linear storage was derived in [16] by setting $M = \log N$. Since the storage function is a hyperbola, we note that the selection $M = \log N$ doesn't necessarily yield the minimal point. If the group size N is sufficiently large, then there are plenty of values of M in the range $\log N < M \leq N$ that will require less key storage than that for the choice of $M = \log N$. We present numerical examples in a tabular manner for a binary tree ($a = 2$). For this case the storage function reduces to $\frac{N-M}{M}$.

N	$\log N$	Range for improved results
2^{10}	10	$10 < M \leq N$
2^{15}	15	$15 < M \leq N$
2^{20}	20	$20 < M \leq N$

5.3.4 Further Improvement to the Cluster Based Techniques

We note that the solution presented by Canetti et al need not be optimal in a heterogeneoous network with non-uniform member revocation probabilities. If the member revocation probabilities are non-uniform, using the modeling in the previous section, and assuming that the individual member revocation event are independent, for cluster i, the probability of revocation of the cluster is the sum of the probabilities of the revocation of the cluster members. Using these

probabilities, we can define the *entropy of cluster revocation* in a similar manner as the entropy of member revocation. Moreover, given the individual cluster revocation probabilities, we can solve the problem of optimal number of keys per cluster for the $a-ary$ rooted tree using an identical derivation as in the previous chapter.

We summarize the results without repeating the proofs. Since the set of KEKs assigned to a cluster should be unique, and the KEKs are distributed on the nodes of the tree, the unique indexing requires that the *number of keys* assigned to a cluster should satisfy the Kraft inequality [20, 21]. Denoting the number of keys assigned to a cluster with probability of revocation p_i by l_i, we note

$$\sum_{j=1}^{N} a^{-l_i} \leq 1. \tag{5.6}$$

Minimization of the average number of keys held by a member with the unique indexing leads to the solution that the optimal number of keys assigned to a cluster with revocation probability p_i is given by $l_i = -\log_a p_i$. The following theorem summarizes the optimal number of keys per cluster.

Theorem 5.1. For a rooted-tree based key assignment that satisfies the Kraft inequality, the optimal average number of keys, excluding the root key and the SK, assigned to a cluster is given by the $a-ary$ entropy $H_a = -\sum_{i=1}^{N} p_i \log_d p_i$ of the *member revocation* event. For a cluster i with probability of revocation p_i, satisfying the optimization criteria, the optimal number of keys l_i, excluding the root key and the SK, is given by

$$l_i = -\log_d p_i \tag{5.7}$$

$$H_a = -\sum_{i=1}^{N} p_i \log_d p_i. \tag{5.8}$$

In order to design the system for worst case condition, from the view point

of the GC, the uncertainty as to which cluster is to be revoked next should be maximized subject to the condition $\sum_{i=1}^{N} p_i = 1$. Formally, we have the following optimization problem

$$\begin{aligned} &\text{maximize} H_a = -\sum_{i=1}^{N} p_i \log_d p_i \\ &\text{subject to} \sum_{i=1}^{N} p_i = 1 \end{aligned} \tag{5.9}$$

In this formulation, the aim is to find the set of optimal values to each cluster revocation probabilities.

The optimal result is the well known uniform distribution for each cluster revocation probabilities, and the corresponding entropy is the maximal entropy given by $\log_d N$. Hence, the cluster size should be selected such that the revocation probabilities of each of the cluster is identical. If all the members have same probability of being revoked, the cluster size will also be the same. This though is only a special case. Hence, the results in [16] for key storage can be further improved in more than one way.

Chapter 6

Conclusion

This dissertation addresses key generation and key distribution problems for a single sender-multiple receiver model of secure multicast. Commercial applications such as stock quote distribution and selective new updates belong to the single sender- multiple receiver model.

A new key generation scheme was proposed that allows a set of mutually suspicious members to generate a common secret. The scheme also lets the members generate the common secret without having to expose their individual secrets. In this scheme, we assumed that there is a third party to initiate the key generation procedure. Every key generating member is given an initial pad and a *group binding* parameter that is the sum of all the pads. Members generate individual shares called *fractional keys*, use the individual pads to create a hidden fractional keys, and exchange the hidden fractional keys. Every member then combines the hidden fractional shares to generate the hidden common key/secret. The group binding parameter is then used to remove the *combined effect* of *all* the pads, and extract the new common key/secret. We also provided a mechanism to update the pads of individual members. The common key/secret, which is

the sum of individual shares, is the group binding parameter. Hence, the group binding parameter is updated every time the group keys are computed. Although we presented the key generation scheme for an additive law, variations such as using a multiplicative law for combining the individual secrets are also possible.

In the second part of the thesis, a new approach to key distribution was proposed that made use of basic concepts from information theory. In doing so, we also showed that the *best*, or optimal, strategy that minimizes the number of keys to be stored while minimizing the number of updated messages as well, is equivalent to the optimal selection of codeword length. We further showed that the solution obtained using concepts from information theory does not prevent collusion. This point is demonstrated by considering the recent proposal by researchers at IBM corporation [17], and showing that their results correspond to optimal selection of codeword length selection but lead to member collusion. We also presented the condition that prevents user collusion from compromising a valid member. We then showed that the use of entropy also allows one to group members into clusters with each cluster having equal probability of being revoked. We also showed that it is possible to find the optimal key assignment in the cluster case based on entropy of cluster revocation event.

There are many interesting problems and directions of future research arising from the work presented here.

First and foremost, the results need to be extended to the case of many senders and many receivers which represents all possible multicast applications. Specifically we plan to pursue the following problems: authentication without reducing performance in multicasting; group key generation and distribution; handling membership across groups. What are the appropriate generalizations

of the results presented in this thesis? What is the "correct" information theory analogy in this case of many-to-many.

Second, we plan to investigate implementation of the key distribution system described in the thesis in real world networks. This will help identify any potential practical problems that may have not been recognized in the theoretical/analytical work presented here.

Third, we plan to investigate fast algorithms for group key generation and we are particularly interested in approaches that lead to reduced key lengths, such as methods using elliptic curves. In this context we also intent to investigate applications in conditional access schemes for multicasting of real-time videos and multimedia information streams. We are also interested in exploring applications in mobile networks and in networks with rapidly varying topologies which will cause fast dynamic changes in multicast group membership.

Fourth, a set of parameters identified by our theory is the probabilities of member revocation p_i. Typically these will not be known exactly. How can they be estimated? What is the error of inaccuracies in the estimation? Can we obtain robust schemes and what is the cost of those in terms of scalability? Can we develop universal methods that do not rely on explicit estimates of these probabilities? Are these related to universal coding? Finally, we are interested and plan to investigate attacks that are more general and sophisticated than the collusion attacks described here. Specific problems include intrusion detection, defense against covert channel use, and defense against schemes exploiting biasing of the key space by the contributing member.

Multicast security is a key and central problem in the Internet-centric world. We have investigated some initial problems in a systematic and analytical fashion.

Much exciting and important research remains to be done.

BIBLIOGRAPHY

[1] T. Cover, J. Thomas, Elements of Information Theory, John Wiley & Sons, Inc, NY, 1991.

[2] M. steiner, G. Tsudik, and M. Waidner, "Diffie-Hellman key distribution extended to group communication", 3rd *ACM Conf. on Computer and Communications Security"*, 1996.

[3] A. Fiat and M. Naor, "Broadcast Encryption", *Advances in Cryptology-Crypto'92*, Lecture Notes in Computer Science. vol. 773, pp. 481-491, Springer-Verlag, Berlin Germany, 1993.

[4] D. R. Stinson, and T. V. Trung, "Some New Results on Key Distribution Patterns and Broadcast Encryption", to appear in Design, Codes and Cryptography.

[5] D. R. Stinson, "On some methods for unconditionally secure key distribution and broadcast encryption", to appear in Design, Codes and Cryptography.

[6] M. Brumester and Y. Desmedt, "A Secure and Efficient Conference Key Distribution System", *Advances in Cryptology- Eurocrypt'94*, Lecture Notes in Computer Science. vol. 950, pp. 275-286, Springer-Verlag, Berlin Germany, 1994.

[7] H. Harney and C. Muckenhirn, "GKMP Architecture", *Request for Comments(RFC)* 2093, July 1997.

[8] H. Harney and C. Muckenhirn. "GKMP Specification". Internet RFC 2094, July 1997.

[9] S. Mittra, "Iolus: A framework for Scalable Secure Multicasting", In *Proceedings of ACM SIGGCOM'97*, pages 277–288, September 1997.

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

[11] C. K. Wong, M. Gouda, S. S. Lam,"Secure Group Communications Using Key Graphs", In *Proceedings of ACM SIGCOMM'98*, September 2-4, Vancouver, Canada.

[12] R. Canetti, and B. Pinkas, "A taxonomy of multicast security issues", *Internet draft*, April, 1999.

[13] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, "Multicast Security: A Taxonomy and Efficient Reconstructions", in *Proceedings of IEEE Infocom'99*.

[14] D. A. McGrew and A. Sherman, "Key Establishment in Large Dynamic Groups Using One-Way Function Trees", *Manuscript, 1998*.

[15] G. Caronni, M. Waldvogel, D. Sun, and B. Plattner, "Efficient Security for Large and Dynamic Groups", In *Proc. of the Seventh Workshop on Enabling Technologies*, IEEE Computer Society Press, 1998.

[16] R. Canetti, T. Malkin, and K. Nissim, "Efficient Communication-Storage Tradeoffs for Multicast Encryption", In Eurocrypt 99, pp. 456 - 470.

[17] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, D. Saha, "Key Management for Secure Internet Multicast Using Boolean Function Minimization Techniques", in Proceedings of IEEE Infocom'99.

[18] R. Canetti, P-C. Cheng, D. Pendarakis, J. R. Rao, P. Rohatgi, D. Saha, "An Architecture for Secure Internet Multicast", *Internet Draft*, November 1998.

[19] B. Quinn, "IP Multicast Applications: Challenges and Solutions", *Internet draft*, November 1998.

[20] R. Poovendran, and J. S. Baras, "An Information Theoretic Approach for Design and Analysis of Rooted-Tree Based Multicast Key Management Schemes", Springer Verlag Lecture Notes in Computer Sciences, *Advances in Cryptology*- CRYPTO'99, August 1999, Santa Barbara, USA.

[21] R. Poovendran, and J. S. Baras, "An Information Theoretic Approach to Multicast Key Management", in Proceedings of IEEE Information theory and Networking Workshop, Metsovo, Greece, June, 1999.

[22] R. Poovendran, S. Ahmed, S. Corson, J. Baras, "A Scalable Extension of Group Key Management Protocol", Proceedings of ATIRP Conference, pp 187-191, Feb, 1998, Maryland.

[23] R. Poovendran, S. Corson, J. Baras, "A Dynamic Group ElGamal key Generation with Tight Binding", Proceedings of ATIRP Conference, Feb, 1999, Maryland.

[24] R. Poovendran, S. Corson, J. Baras, "A Private Scheme for Distributed Shared Key Generation", Proceedings of the Information Theory Workshop, June 1999, South Africa.

[25] R. Poovendran, S. Corson, J. Baras, "A Shared Key Generation Procedure Using Fractional Keys", Proceedings of the IEEE Milcom, October, 1998, Boston, MA.

[26] D. Boneh and M. Franklin, "Efficient Generation of Shard RSA Keys", Crypto'98.

[27] M. Yung, "Cryptovirology: Extortion-Based Security Threats and Countermeasures", Proceedings of IEEE Symposium on Security and Privacy, pp. 129-140, 1995.

[28] J. L. Massey, "An Information-Theoretic Approach to Algorithms", Impact of Processing Techniques in Communications, In NATO Advanced Study Institutes Series E91, pp. 3-20, 1985.

[29] J. L. Massey, "Some Applications of Source Coding to Cryptography", In European Trans. on Telecom., Vol. 5, pp. 421-429, July-August 1994.

[30] H. N. Jendal, Y. J. B. Khun, and J. L. Massey, "An Information-Theoretic Approach to Homomorphic Substitution", In Advances in Cryptology-Eurocrypt'89, LNCS-434, pp. 382-394, 1990.

[31] Y. Desmedt, Y. Frankel, and M. Yung, " Multi-receiver/Multi-sender network security: efficient authenticated multicast feedback", *IEEE Infocom'92*, pp. 2045-2054.

[32] A. Menezes, P. van Oorschot, and A. Vanstone, "Handbook of Applied Cryptography", CRC Press, Boca Raton, 1997.

[33] M. Naor and O. Reingold, "From Unpredictability to Indistinguishability: A Simple Construction of Pseudo-Random Functions from MACs", *Advances in Cryptology- Crypto'98*, Lecture Notes in Computer Science. vol. 1462, pp. 267-282, Springer-Verlag, Berlin Germany, 1998.

[34] M. Luby, Pseudo-Random Functions and Applications, Princeton University Press, 1996.

[35] T. Hardjono, B. Cain, and N. Doraswamy, "A Framework for Group Key Management for Multicast Security", *Internet draft*, July 1998.

[36] A. Ballardie. "Scalable Multicast Key Distribution". Internet RFC 1949, May 1996.

[37] N. Koblitz, Algebraic Aspects of Cryptography, pp. 11-12, Springer-Verlag, New York, 1998.

[38] N. Koblitz,"Cryptography as a teaching tool", In Cryptologia, October, 1998.